universität wien

# MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Using Problem Instance Feature Extraction for Automated Metaheuristic Selection and Configuration to solve the Vehicle Routing Problem"

verfasst von / submitted by

Matthias Weinwurm, B.Sc. (WU)

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna 2021

| | |
|---|---|
| Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet: | UA 066 915 |
| Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet: | Masterstudium Betriebswirtschaft / Master's degree programme Business Administration |
| Betreut von / Supervisor: | Univ.-Prof. Dr. Jan Fabian Ehmke |

## Abstract in English

With a steadily rising number of increasingly complex metaheuristics for solving hard optimization problems being presented in the scientific literature, the choice of one metaheuristic and the setting of its parameters for a given problem instance becomes more and more difficult and time-consuming. The automated algorithm selection and configuration problems, respectively, aim to automate these tasks, often using machine learning approaches. While these two domains are treated separately in the literature, this thesis aims to create an integrated model by using components and concepts of the respective domains and combining them into one, while testing the model on a set of vehicle routing problem (VRP) instances. First, a feature set describing VRP instances is constructed using multiple sources and it is thereby shown, that features for the traveling salesman problem can be reused for the VRP with good predictive performance. It is then shown that the proposed model outperforms algorithm selection or algorithm configuration approaches, that are run individually or sequentially, which implies that the domains are in fact interconnected. Furthermore, it is shown that the performance gains can be realized within a very short runtime, which makes the model potentially applicable to real-world scenarios.

*Keywords*: algorithm selection problem, algorithm configuration problem, machine learning, vehicle routing problem

## Abstract in German

Aufgrund der stetig zunehmenden Anzahl an immer komplexeren Metaheuristiken für das Lösen schwerer Optimierungsprobleme, die in der wissenschaftlichen Literatur präsentiert werden, wird die Auswahl einer Metaheuristik sowie das Setzen ihrer Parameter für eine gegebene Probleminstanz immer schwerer und zeitaufwändiger. Das Automated Algorithm Selection sowie Configuration Problem zielt darauf ab, die jeweiligen Aufgaben, zumeist mithilfe von Machine Learning Ansätzen, zu automatisieren. Während diese beiden Domänen in der Literatur jedoch als separat betrachtet werden, versucht diese Arbeit ein integriertes Modell zu erschaffen, das Komponenten und Konzepte der jeweiligen Domänen verwendet und diese zu einem Modell kombiniert, welches dann an einem Set von Vehicle Routing Problem (VRP) Instanzen getestet wird. Zuerst wird ein Feature Set, das VRP Instanzen beschreiben soll, aus mehreren Quellen konstruiert, wobei gezeigt wird, das Features für das Traveling Salesman Problem mit guter Vorhersagekraft für das VRP wiederverwendet werden können. Anschließend wird gezeigt, dass das vorgestellte Modell gegenüber Algorithm Selection oder Algoritm Configuartion Ansätzen, die entweder allein oder hintereinander ausgeführt werden, bessere Ergebnisse erzielt, was impliziert, dass die beiden Domänen tatsächlich miteinander verbunden sind. Des Weiteren wird gezeigt, dass diese Leistungssteigerungen innerhalb einer sehr kurzen Laufzeit realisiert werden können, was das Modell potenziell für eine Anwendung in der realen Welt eignet.

*Keywords*: Algorithm Selection Problem, Algorithm Configuration Problem, Maschinelles Lernen, Vehicle Routing Problem

## Table of Contents

## List of Abbreviations

| | |
|---|---|
| TSP | … traveling salesman problem |
| VRP | … vehicle routing problem |
| ILS | … iterated local search |
| SA | … simulated annealing |
| GA | … genetic algorithm |
| ACO | … ant colony optimization |
| PCA | … principal component analysis |
| MST | … minimum spanning tree |
| VBS | … virtual best solver |
| SBS | … single best solver |
| AAS | … automated algorithm selection |
| AAC | … automated algorithm configuration |

## List of Tables

## List of Figures

# 1. Introduction

For most real-world problems it can be said, that the larger the problem at hand is the harder it is to solve, meaning to find the optimal solution. While for some problems this incline in problem hardness in relation to the problem size is moderate, there are others where the hardness grows exponentially. These problems, for which no algorithm exists than can solve them in polynomial time, are called NP-hard problems. The vehicle routing problem, which will be the focus and the application used within this thesis, is a member of this group. As solving these problems exactly is infeasible in most situations, special solvers called heuristics are developed that do not guarantee to find the optimal solution but aim to find a very good solution within a short period of time. (Groër et al., 2010)

The no free lunch theorem by Wolpert and Macready (1997) states, however, that there cannot be one algorithm outperforming all others over all possible problems. The term algorithm, in this regard, does not only refer to the algorithm or solver itself, but also its configuration, meaning that no single algorithm configuration can perform best over all possible problems. This is especially important, as the correct choice of a solver's parameter setting can prove to be just as vital to its performance as the choice of the solver itself.

This theorem now therefore presents two problems. First, how can an algorithm that works well not only for the type of problem we face, but also for the specific problem instance at hand effectively be chosen? And second, once an algorithm was selected, how can an ideal parameter setting be found that optimizes the algorithm's performance on the problem instance at hand?

In practice, up until today both of these problems are largely solved by hand, which not only requires a large amount of time, but also extensive knowledge about the available portfolio of solvers, their inner workings and the effect of their free parameters on their performance, as well as about the problem domain and the specific problem instance at hand. Even if that required time and knowledge is available, finding a good solver and a good configuration can require multiple trial-and-error runs and practice has shown that even then solving these tasks manually can lead to inferior results. (Kadioglu et al., 2010)

As a result, the domains of automated algorithm selection and automated algorithm configuration emerged, which aim to provide an automated and ideally instance-specific approach to solving these problems. The former tries to find a mapping between problem instances and their ideal target algorithm out of an algorithm portfolio, whereas the latter aims to identify the optimal algorithm configuration for either a set of instances or also for a given instance specifically. (Huang et al., 2020; Rice, 1976)

Interestingly, these two domains are treated separately in the scientific literature. However, as an algorithm's configuration can have an immense impact of the performance of the algorithm, the question arises whether these domains can in fact be separated from each other or if an ideal choice of parameter settings can affect the choice of the ideal algorithm in the first place. Therefore, this thesis aims at creating an integrated approach by combining instance-specific algorithm selection and instance-specific algorithm configuration and compares the performance of the proposed approach with applying approaches from the two domains either individually or sequentially.

Lastly, as the time-consuming nature of these tasks was named as one of the main motivations and with a real-world application in mind, this thesis will also aim to create an approach that can perform these tasks within a reasonable time frame.

The structure of this thesis is organized as follows. Section 2 will provide the theoretical background and introduce the reader into the topic of automated algorithm selection and automated algorithm configuration as well as to some of the data science and machine learning tasks that are used within these domains. Section 3 will present the state of the art in the scientific literature before section 4 will identify some gaps in the literature, detail the focus of this thesis and position it within the current state of the literature. Section 5 will provide a definition of the vehicle routing problem and present the proposed approach of creating an integrated model combining instance-specific algorithm selection and instance-specific algorithm configuration. Section 6 will detail the computational results in terms of solution quality, predictive performance of the model and a runtime analysis before section 7 will close the thesis with some concluding remarks and an outlook on future research topics.

# 2. Theoretical Background

## 2.1. Automated Algorithm Selection

As stated in the introduction to this thesis, according to the no free lunch theorem by Wolpert and Macready (1997) there exists no single algorithm that dominates all others. In practice there are rather multiple algorithms with each one of them outperforming the others on some problem instances. Therefore, the algorithm selection problem does not employ a single algorithm but a pool or portfolio of algorithms together with an algorithm selector, which is supposed to map each instance to its ideal solver. (Lindauer et al., 2019)

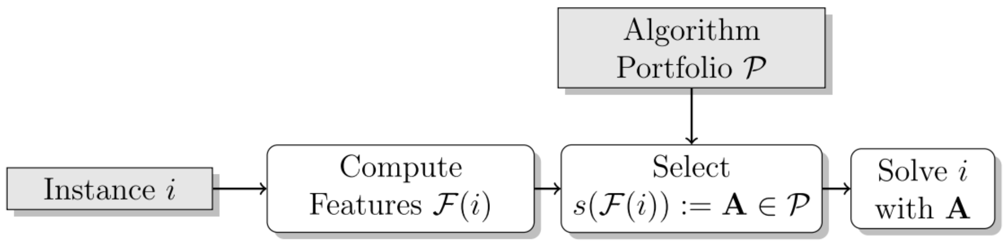
The algorithm selection problem is first introduced by Rice in 1976. In its basic form, the problem contains a problem space, an algorithm space and some performance measure. The goal is to identify and apply any sort of mapping between the problem and algorithm spaces in order to select the algorithm that is expected to achieve the best performance measure for any given instance. As, in most cases, the exact mapping from each instance to its best algorithm is not known beforehand, Rice extends the model by a feature space. Here, features are extracted from problem instances under the assumption that instances with the same or very similar feature values will result in the same performance of any given algorithm. In that aspect, Rice emphasizes that the determination of a good set of features is one of the most important aspects of the algorithm selection problem. (Rice, 1976)

As depicted in the figure below, the instance-specific algorithm selection problem can therefore be formally defined as follows: Given a set of problem instances $I$, a representation of an instance $i \in I$ as a feature vector $F(i)$, an algorithm portfolio $P$ and a performance measure $m : P \times I \to \mathbb{R}$, the aim of the algorithm selector is to find an optimal mapping $s^*$ from the instance space to the algorithm portfolio $s : I \to P$ in order to minimize cost across the instance space without loss of generality: (Lindauer et al., 2019)

$$s^* = \arg\min_{s} \sum_{i \in I} m(s(F(i)), i)$$

**Figure 1**

*Per-Instance Algorithm Selection Problem (Lindauer et al., 2019, p.87)*


Algorithm Portfolio $\mathcal{P}$
Instance $i$
Compute Features $\mathcal{F}(i)$
Select $s(\mathcal{F}(i)) := \mathbf{A} \in \mathcal{P}$
Solve $i$ with $\mathbf{A}$

Given the fact that in practice the algorithm portfolio usually has a manageable size, the most common approach to automated algorithm selection is to solve every instance in the training set with every solver in the algorithm portfolio and document the single best solver for each instance. With those best algorithms now acting as class labels, this provides a labeled dataset, with which a supervised learning approach in the form of classification can be taken to find a mapping from the feature vectors to the algorithms in the portfolio and to assign unseen instances into one of the classes and therefore to one of the algorithms. Alternatively, regression approaches have been employed as well, trying to learn mappings from the feature vector directly to the algorithm performance of each algorithm, and selecting the algorithm with the best predicted performance for an unseen instance. However, classification approaches have shown to generally outperform their regression-based counterparts. (Kerschke et al., 2018)

A noteworthy variant next to the per-instance algorithm selection problem is the per-set algorithm selection problem. In this problem the goal is not to find any mappings between separate instances and their best algorithms but to find one single algorithm which generates the best performance across an entire set of problem instances. Per-set algorithm selection can be used for example for any applications with homogenous problem instances, where an algorithm that has proven to be successful on the entire training set is expected to perform equally well on all unseen instances. (Kerschke et al., 2019)

An alternative approach to automated algorithm selection is the concept of parallel algorithm portfolios. Here all or multiple algorithms from a predefined portfolio of algorithms are run in parallel and as soon as one of the algorithms solved the problem or a given termination criteria is reached, the best solution is returned. While this approach requires powerful, parallel hardware to operate effectively, algorithm portfolios can generate near VBS quality solutions under ideal conditions. (Lindauer et al, 2015a)

## 2.2.Automated Algorithm Configuration

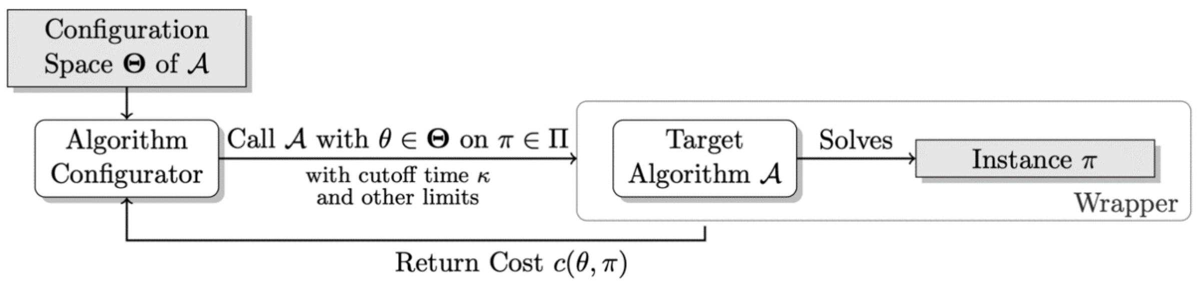
Most of the state-of-the-art algorithms, regardless of in optimization or any other domain, come with a set of free parameters, which allow the users to fine tune the algorithm to their specific needs. It is often stated that the configuration of a solver has an even greater effect on the quality of a found solution than the initial choice of which solver to apply, which makes the algorithm configuration problem an essential part of every solution approach. This task of determining a good set of parameter settings is often done by hand, which represents an enormous time effort, as experience and extensive knowledge of the domain and the specific problem to solve is required, and even then, it can lead to several "trial and error" runs. But not only is this task very time-intensive, it was shown that the manual setting of algorithm parameters frequently leads to inferior results than automatizing this task. (Kadioglu et al., 2010)

As illustrated in figure 2, the algorithm configuration problem can therefore be formally defined as follows: given a target algorithm A, a configuration space Θ, a set of problem instances Π, and a performance metric $c : \Theta \times \Pi \rightarrow \mathbb{R}$, find a configuration $\theta^* \in \Theta$ that minimizes the performance metric $c$ across the set of problem instances Π: (Eggensperger et al., 2019)

$$\theta^* \in \arg\min_{\theta\in\Theta} \sum_{\pi\in\Pi} c(\theta,\pi)$$

**Figure 2**

*Algorithm Configuration Problem (Eggensperger et al., 2019, p.863)*


Configuration Space $\mathbf{\Theta}$ of $\mathcal{A}$
Algorithm Configurator
Call $\mathcal{A}$ with $\theta \in \mathbf{\Theta}$ on $\pi \in \Pi$
with cutoff time $\kappa$ and other limits
Target Algorithm $\mathcal{A}$
Solves
Instance $\pi$
Wrapper
Return Cost $c(\theta, \pi)$

The parameters in this problem domain are commonly classified into 3 types. Categorical parameters can take one of a finite set of independent values, this can for example represent which construction heuristic or which crossover operator to apply. Ordinal parameters can take one of a finite set of values as well, however, these values are not independent but can be compared with each other, for example the values of good, better and best. Lastly, there can be numerical parameters, which are often again split into integer or real numbers. (Hutter & Ramage, 2015)

The algorithm configuration problem itself can be further classified into offline algorithm configuration, often also called parameter tuning, and online algorithm configuration, often called parameter control. While offline algorithm configuration approaches attempt to find good configurations offline, which are then evaluated by running the algorithm with these approaches and collecting the final results, online approaches collect instant feedback during the algorithm run and are able to change parameter settings according to the current state of the algorithm run. (Huang et al., 2020) As this thesis focuses on offline algorithm configuration approaches, their online counterparts will not be covered in detailed here.

Opposed to the algorithm selection domain, the instance-specific algorithm configuration problem remains an open research challenge to date, which is why most scientific works apply an instance-oblivious or per-set algorithm configuration approach. Per-set algorithm configuration generally employs a very resource intensive, trial-and-error based system of generating parameter configuration and then testing them on the provided instance or instance set. Huang et al. (2020) classify these approaches into 3 categories. “Simple Generate-Evaluate Methods” first generate a set of configurations before evaluating them and returning the best among this set, “Iterative Generate-Evaluate Methods” contain several cycles of the “Simple Generate-Evaluate Methods”, albeit often with fewer configurations per iteration, and “High-Level Generate-Evaluate Methods”, where the candidate solutions are generated using external algorithm configurators before those configurations are then evaluated and the best one is returned. (Huang et al., 2020)

While the methods for generating candidate configurations, for example full factorial design, central composite design, Latin hypercube design or even random sampling, have shown to be of little effect to the performance of the configurator and are mostly interchangeable, the methods for evaluating the candidates differ significantly between the approaches. The main evaluation methods are repeated evaluation, racing, intensification, sharpening and adaptive capping. (Huang et al., 2020) Repeated evaluation factors in the stochasticity of many algorithms by running the target algorithm repeatedly and returning the average performance. (Yuan et al., 2012) In racing methods, several instances of the target algorithm running on different candidate configurations are raced against each other and are iteratively evaluated, with candidates being discarded as soon as statistical evidence proves their inferiority. (Birattari & Kacprzyk, 2009) Intensification describes the process of comparing a new candidate to the incumbent by evaluating it on the same instances that the incumbent was evaluated on. If the new candidate shows worse performance than the incumbent, it is discarded, if it outperforms the incumbent on all instances the incumbent was evaluated on, it becomes the new incumbent solution. (Hutter et al., 2009) In sharpening, each candidate is evaluated only on a small set of instances before those candidates that showed promising results during these first steps are evaluated more thoroughly. (Smit & Eiben, 2009) Lastly, adaptive capping describes the approach of prematurely cutting off runs of unsuccessful configurations in order to achieve computational savings. (Hutter et al., 2009)

As described in the previous section, most algorithm selection concepts use classification approaches to map each instance to its best solver. This is, however, only possible with a relatively small set of solvers. In algorithm configuration, especially when the parameters include numerical parameters, there is an exponential or even infinite number of possible configurations, which makes a classification approach virtually impossible. Earlier works employed linear regression approaches to learn mappings from feature vectors and parameter configurations to performance metrics, but this approach did not lead to a significant performance improvement over per-set algorithm configuration. (Hutter & Hamadi, 2005; Hutter et al., 2006)

An alternative approach was introduced by Malitsky and Sellmann (2009) called stochastic offline programming. In this iterative process the authors start by clustering the problem instance based on a distance metric in the instance feature space under the assumption that similar instances behave similarly when solved by the same algorithm. For each of those clusters a good solver is determined using a per-set approach, before the results are then used to adapt the applied distance metric. While this approach is primarily used on general heuristic generation and selection, Kadioglu et al. (2010) evolve this concept to present ISAC, the instance-specific algorithm configuration. ISAC first clusters the problem instances using g-means clustering before solving the per-set algorithm configuration problem for each of the clusters using the GGA configuration approach. ISAC is shown to achieve significant improvements over per-set algorithm configuration and remains one of the best-known approaches to the instance-specific algorithm configuration problem to date.

## 2.3. Data Science

Most definitions of the term data science start with the definition and differentiation of the terms data and knowledge, which can be stated as follows:

**Table 1**

*Definition of Data and Knowledge (Berthold et al., 2020, p.3)*

| data | knowledge |
|---|---|
| • refer to single instances (single objects, people, events, points in time, etc.) | • refers to classes of instances (sets of objects, people, events, points in time, etc. |
| • describe individual properties | • describes general patterns, structures, laws, principles, etc. |
| • are often available in large amounts (databases, archives) | • consists of as few statements as possible |
| • are often easy to collect or to obtain (e.g., scanner cashiers in supermarkets, Internet) | • is often difficult and time consuming to find or to obtain (e.g., natural laws, education) |
| • do not allow us to make predictions or forecasts | • allows us to make predictions and forecasts |

Data science now describes the process of analyzing often large amounts of data in order to extract knowledge from it. (Berthold et al., 2020)

This subsection will focus on this domain by first looking at ways to extract and collect data, with a focus on optimization problems in mind, then the process of data preparation will be covered before some machine learning techniques used in data science will be described.

### *2.3.1. Feature Extraction*

In general, the feature extraction or often called feature construction describes the process of turning a large set of raw input data into a set of useful features. In practice, this has many different applications, for example signal or image enhancements by filtering out noise, pattern recognition in large datasets, extracting representative features from large amounts of health data, calculating color scales or textural features of images, and countless more. Data preprocessing tasks like standardization, normalization, discretization or feature subset selection are often classified under the overarching term feature extraction as well. In short, the aim of feature extraction is to extract a set of representative and meaningful features out of a dataset that maximizes the ability for further interpretation, be it by humans or machine learning algorithms. An always important decision thereby regards the number of features to be extracted. If too few are extracted, useful information might be lost, however, if too many features are extracted, it not only increases computational efforts, but the most relevant features can in fact be overshadowed by the sea of irrelevant, noisy and redundant features. Many works in the literature argue, that being too inclusive in feature extraction should always be the preferred option over risking to discard useful information, especially if further preprocessing steps like feature subset selection are performed, which is also the approach applied in this thesis. (Guyon & Elisseeff, 2006)

In optimization problems feature extraction describes the process of calculating a set of predefined features from a specific problem instance. The problem instance is usually provided in the form of an input file, in routing problems for example this comes in the form of a set of coordinates and demands of all the customers as well as the vehicle capacity and any other relevant information. The goal of the feature extraction process is to calculate a representative and comprehensive set of features out of the instance file, that not only reveals any characteristics of the specific problem that would be hard to detect by looking at the raw data alone, but it also allows for comparability among different instances as it converts all instance descriptions into the same dimensionality. (Rasku et al., 2016)

While, as mentioned above, data preprocessing and feature subset selection tasks can often be classified as feature extraction tasks as well, in this thesis feature extraction only focusses on the calculation of a set of features from the problem instance file, which is why the tasks are split into separate sections.

### *2.3.2. Data Preparation*

Once a dataset is established, performing data preparation is essential in order to maximize the effect and the results of the machine learning process, as in practice, datasets often come with very high dimensionality. These dimensions do not only often contain values in vastly different ranges, which makes it difficult to compare them to one another, but some of them are also more relevant to the purpose in mind than others, while some might even be completely irrelevant. Therefore, while it might make sense to extract many features from problem instances and include a larger number of dimensions in the dataset, this makes the need for a data preparation step even more pressing. (Guyon & Elisseeff, 2003)

Data preparation can be done in many forms and substeps. The first one presented here is data cleaning. While data cleaning is often used synonymously for the entire data preparation process, here it simply refers to the act of improving the quality and readability of the data at hand. This can take on many forms, depending on the type of data, like text, numbers or dates for example. These can include conversions to upper or lower case, removing spaces or punctuation marks, using spell checking, standardizing formats of dates and numbers and many more. (Berthold et al., 2020) In the case of data extracted from optimization problem instances, which are usually in numerical form, the most relevant cleaning action is often normalizing the data, in order to increase comparability.

With a well-maintained data set at hand, the next step in data preparation is feature selection or dimensionality reduction. As mentioned above, datasets often contain a very large number of dimensions with not all of them being equally relevant and some being even completely irrelevant to the problem that is set to be solved. Therefore, feature selection aims at finding an optimal subset that maximizes the predictive quality of the machine learning algorithms. (Berthold et al., 2020)

Filter methods try to select the k top-ranked features according to a specified evaluation criterion, which can be as simple as removing some features with low or even no variance across the dataset. As each feature is evaluated individually in this case, filter methods have a big advantage in terms of computational effort required against some other methods that will be described later. In case of supervised learning, where a class label or a target variable is available, the predictive performance of each individual feature can be evaluated, before all of them are ranked and the best k of them are selected. (Guyon & Elisseeff, 2003)

While simple filter methods present a fast and easy way to select features, these methods select a set of features which perform well individually but they do not guarantee that they also perform well collectively, as some of them might turn out to be redundant. Another type of feature selection approaches therefore tries to select the top-ranked subset of features from the dataset, which is a by far more expensive approach. These approaches include for example calculating the cross products of the features in a feature set, ranking the predictive performance of those different cross products and taking the subset resulting in the best performance. Correlation based filtering describes the process of evaluating the feature subset by looking at both its correlation to a target variable as well as to the correlation of the features with each other. Wrapper methods are another noteworthy type a feature selection approaches, which evaluate feature subsets by performing actual machine learning tasks on them. These feature selection approaches trying to find a top-ranked subset of features usually employ either forward or backward feature selection. Forward feature selection methods start with an empty feature subset and iteratively add the single feature the maximizes the performance of the resulting subset, according to the selected evaluation criterion, until a predefined termination criterion is met, for example a given size of the subset or if the performance of the new subset is worse than the current one's. Backward feature selection methods operate very similarly only that they start with the full set of all features and gradually remove one feature after another. (Berthold et al., 2020)

Principal component analysis (PCA) presents a popular alternative to feature selection in dimensionality reduction, as this approach does not reduce the number of features in the dataset but rather projects them to a lower dimension by calculating new variables as linear functions of the original dataset while preserving as much of the original dataset's variance as possible. In short, PCA comes down to an eigenvector/eigenvalue problem resulting in a set of uncorrelated linear combinations, which are called the principal components, of the dataset. Those principal components are defined as the linear combinations of the dataset as defined by its eigenvectors. The set of components, which contains exactly as many components as the original dataset's dimension, are then usually sorted by its eigenvalues, which represent the variances of the linear combinations, in descending order and the first $k$ of them are selected. (Jolliffe & Cadima, 2016)

#### ***2.3.3. Machine Learning***

As machine learning techniques are commonly used for creating the mappings in both instance-specific algorithm selection and instance-specific algorithm configuration, with the former predominantly employing classification techniques while the latter uses clustering approaches. This section provides a very brief introduction to machine learning and the tasks of supervised and unsupervised learning.

Generally speaking, machine learning algorithms aim to find patterns within large amounts of data and ultimately turn data into knowledge while continuously learning and using experience to improve themselves. As defined above, data refers to single instances, describes individual properties, is often easy to collect, available in large amounts and does not allow for any predictions by itself, whereas knowledge refers to classes of instances by describing general patterns, structures or principles. While knowledge is often difficult and time consuming to obtain, it does allow us to make predictions or forecasts. (Berthold et al., 2020)

However, machine learning algorithms often do not only try to describe data in the form of knowledge, but they also try to use that knowledge in order to make predictions. This potentially leads to an incredible wealth of possible applications of machine learning, including natural language processing, speech recognition, image or face recognition, text classification, autonomous vehicle control, medical diagnosis and many more. (Mohri et al., 2018)

There are several machine learning disciplines, which can be defined based on the data they use and the type of tasks they perform. Supervised learning approaches use a labeled dataset for training, meaning that the correct output value for an instance is known and the algorithm tries to find connections between the data and the output value in order to be able to correctly predict the output value of an unseen instance. Unsupervised learning approaches on the other hand work with unlabeled data, which means that they work with raw data and do not know in advance what to predict or explain. These approaches try to find patterns and similarities within the dataset and often try to cluster similar instances together. Another discipline worth mentioning is that of reinforcement learning. Here, the algorithm, often referred to as an agent, performs an action and receives a reward for it, with the intention to identify the state-action pairs that maximize the reward. (Mohammed et al., 2016) While reinforcement learning approaches have been applied in solving optimization problems, for example by Eiben et al. (2006), they will not be covered beyond this point, as they are not relevant for the remainder of this thesis.

The following subsections will describe the machine learning disciplines of supervised and unsupervised learning in more detail.

#### 2.3.3.1. Supervised Learning – Classification.

In supervised learning the machine learning algorithm works with labeled data, meaning that the correct output for every specific instance is given as a label. This label acts as a supervisor giving instant feedback on the correctness or the degree of error of the model's output. With this information the machine learning model can therefore tweak its internal parameters in order to minimize the total error over the training set. However, the goal of machine learning is not to be able to perfectly replicate the labels of the training set, but rather to be able to correctly predict the output for unseen instances as well. This is achieved by generalization, which means that the machine learning algorithm manages to learn patterns within the training data that are generally applicable even to unseen data. In order to attain generalization, it has to be avoided that the algorithm over-analyzes the training data, which leads to the fact that algorithm simply learns the training data and its labels by heart instead of learning useful information needed for predicting the output of unseen data. This phenomenon is called overfitting. (Bonaccorso, 2018)

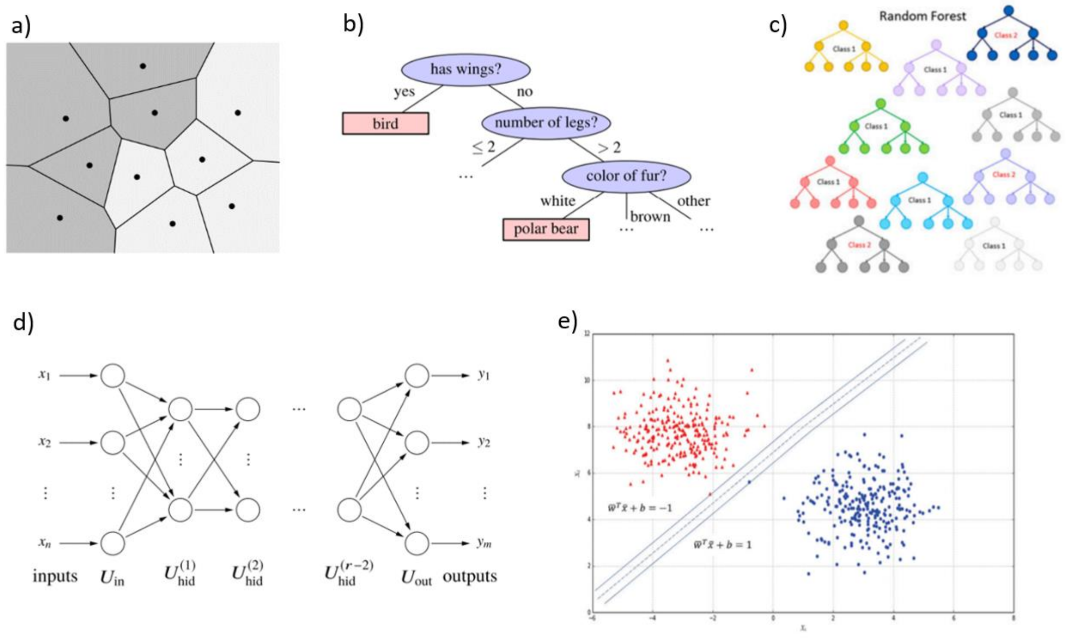
Supervised learning approaches can generally be further distinguished into regression and classification models based on the type of the data labels. The task of producing continuous output values, for example predicting the temperature or stock levels, is called regression whereas the task of predicting discrete output values, for example detecting whether an image shows a cat or a dog, is called classification. Some works also further distinguish supervised learning approaches into descriptive and predictive learning models. Descriptive learning approaches aim to find explanations of how and why the target value depends on the respective input variables, whereas in predictive learning the goal is not primarily to understand the causality between the data and the outcome values, but rather to maximize the model's ability to correctly predict the output values for unseen instances. (Berthold et al., 2020; Bonaccorso, 2018)

No matter whether the model is tasked with classification, regression, descriptive or predictive learning, supervised learning models thereby come in a variety of different shapes. Some of the most popular ones will be described here very briefly and are depicted in figure 3. These include decision trees, which try to find hierarchical structures within the data in order to explain how different areas correspond to different outcomes. Decision trees often only use a subset of all attributes of the dataset which makes them well suited for datasets containing many attributes, that can also be not normalized, noisy or correlated. The Naïve Bayes classifier is a very simply model exploiting the Bayes' rule to model conditional probability distributions of different classes based on some simplified assumptions. While Naïve Bayes classifier provide a good baseline for classification tasks, they are usually outperformed by other models. Furthermore, k-Nearest-Neighbor predictors take a simple approach to learning as well, as they do not even try to find explanations between the data and the correct output values, but they rather just look at historic cases and training instances. These models try to find the k, which is a free parameter, most similar instances based on a predefined distance metric before assigning the new instance to the class that has the highest number of representatives within the sample, or in case of a regression analysis, to the, possibly weighted, average of the k values. Support Vector Machines claim that linear separation or linear regression is sufficient for most supervised learning tasks. However, often the classes are not linearly separable with their current feature space, which is why support vector machines used kernel methods to project the data into a different dimensionality. While support vector machines usually produce good prediction qualities, they do not explain the causality with the data and how they reached these predictions. Artificial neural networks take inspiration from the human brain by creating a model consisting of often multiple layers of neurons connected with each other by synapses.

These networks have an input neuron layer, where data is fed into, an output layer where the predictions are presented, and possibly one or more hidden layers in between. After producing an output value, the degree of error to the correct label is calculated and used to adapt the weights and strengths of the synapses, starting from the output layer flowing back to the input layer, a process called backpropagation. Finally, ensemble methods employ several instances of a type of classifiers in order to create a more robust model. Each individual model produces its own output value, which the ensemble methods then consolidate into one final output value, using a majority vote are any other function. The most popular representative of this group is the random forest model. (Berthold et al., 2020)

**Figure 3**

*Supervised Learning Methods; a) kNN-Classifier (Berthold et al., 2020, p.275); b) Decision Tree (Berthold et al., 2020, p.221); c) Random Forest (Berthold et al., 2020, p.310); d) Artificial Neural Network (Berthold et al., 2020, p.284); e) Separation with Support Vectors (Bonaccorso, 2018, p.209)*



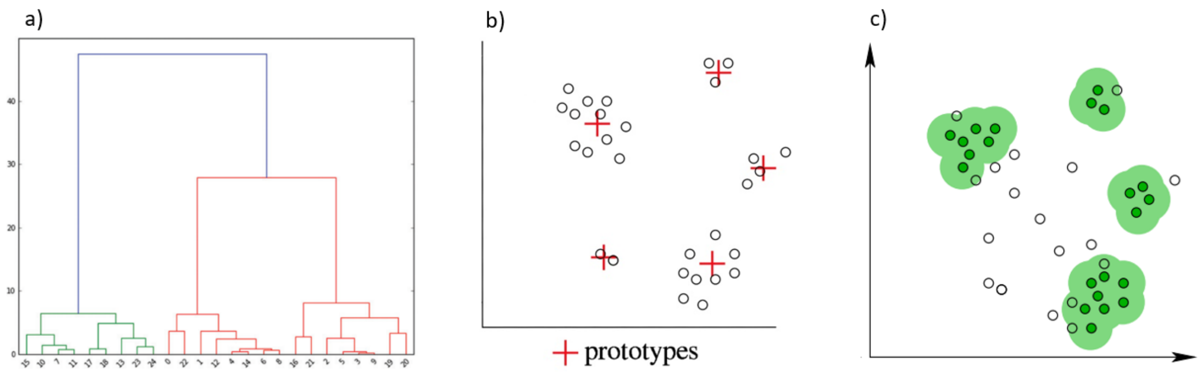
**2.3.3.2. Unsupervised Learning – Clustering.**

In many scenarios however, there are no data labels and therefore no error values for any predictions available for specific data instances. In these cases, machine learning algorithms still try to find patterns in a dataset, extract meaningful information and group individual data instances together based on those patterns. These approaches are called unsupervised learning or often clustering approaches. (Bonaccorso, 2018)

Just as with supervised learning tasks, there is an immense portfolio of different unsupervised learning approaches available, some of which are depicted in figure 4.

**Figure 4**

*Unsupervised Learning Approaches; a) Hierarchical Clustering (Bonaccorso, 2018, p. 344); b) Prototype-Based Clustering (Berthold et al., (2020) p. 174); c) Density-Based Clustering (Berthold et al., 2020, p. 182)*



Hierarchical clustering is thereby one of the oldest and simplest approaches, which starts by assigning each instance to its own cluster, before iteratively finding the two clusters closest to each other within the partition, which are then merged and reinserted into the partition as one combined cluster. This process is usually continued until either the number of clusters is reduced to a predefined number or the minimum distance between any two clusters exceeds a predefined threshold. Another group of clustering approaches is called prototype- or model-based clustering, where an entire cluster is presented by a single prototypical data instance. In k-Means clustering for example, each instance is assigned to its closest out of k randomly selected prototypes, effectively forming k clusters. In an iterative process, the prototype is then relocated to the center of its current cluster before all data instances are reassigned to their nearest prototype. Density-based clustering approaches aim to mitigate the fact, that single linkage clustering approaches, like hierarchical clustering for example, are very sensitive to noise and outlier data points. These methods therefore only consider data points as a cluster if the resulting cluster has a specific density. The DBSCAN algorithm for example uses a minimum number of points and a minimum distance between the points to form a cluster, before expanding the cluster to any points within the minimum distance of its edge. Density-based clustering methods however do not necessarily assign all data points to a cluster, as some outlier points may remain. Other popular clustering approaches include self-organizing maps, clustering by association rules or deviation analysis, however these approaches are not described here in detail, as they are not relevant for the remainder of this thesis. (Berthold et al., 2020; Bonaccorso, 2018)

# 3. Literature Review

As already mentioned in the previous section, the automated algorithm configuration problem was first introduced by Rice (1976) as an instance-specific approach, calculating instance features and mapping the resulting feature vector to a target algorithm. Even though the methods used at these steps have changed over the years, the basic process largely remains unchanged in state-of-the-art algorithm selection approaches today.

The scientific literature on automated algorithm selection is usually split based on the problem domain these approaches are applied to. In constraint satisfaction problems, propositional satisfiability problems and other AI applications for example, there exists a wide array of finished algorithm selectors. These include SATzilla by Xu et al. (2012), claspfolio 2 by Hoos et al. (2014) or AutoFolio proposed by Lindauer et al. (2015), which uses the automated algorithm configurator SMAC to automatically configure the selectors hyperparameters. Within this problem domain there also have been several algorithm selection competitions based on the algorithm selection benchmarking library ASlib by Bischl et al. (2016).

For many other problem domains, including routing problems like the traveling salesman problem (TSP) or the vehicle routing problem (VRP), the literature on automated algorithm selection is much more conceptual, focusing on variations of used feature sets or applied machine learning approaches. There are several works proposing feature sets for the TSP including Smith-Miles and van Hemert (2010, 2011), Kanda et al. (2011), or Mersmann et al. (2013), which however mainly aim at characterizing TSP instance difficulty and thereby predicting the effectiveness of single local search heuristics or metaheuristics. Pihera and Musliu (2014) built on these works and proposed using local search probing features as well as constructing a k-nearest-neighbor graph to better characterize instances before using their feature set for algorithm selection and comparing several supervised learning methods, concluding random forests provides the best results. While more recent works, for example by Kerschke et al. (2018), still continue to propose new features for describing the TSP instances, their focus often lies on testing these feature sets on state-of-the-art solvers and using various feature selection approaches trying to identify the most characteristic features. Even on a portfolio of state-of-the-art solvers, these approaches have proven to be highly successful.

As specifically for the vehicle routing problem, Rasku et al. (2016) aimed to create a comprehensive list of features, most of which were adapted from proposed TSP feature sets and tested this feature set on algorithm selection tasks for the VRP. Rasku et al. (2019) later conducted a comparative analysis of different feature selection approaches together with different classification methods for automated algorithm selection using their comprehensive feature set, where they found that while random forest classifiers generally provide good results, different feature selection approaches outperform each other in different scenarios. Overall, however, they concluded that the use of a comprehensive feature set together with feature selection and algorithm selection provides good results for solving VRP instances.

In contrast to those automated algorithm selection approaches, most of the automated algorithm configuration approaches operate independently from any specific problem domain. They usually only require a target algorithm, a set of parameters, some ranges or rules on how to set these parameters and a performance metric. Therefore, while some of the following works have been successfully applied to the TSP or the VRP, they are all drawn from a general context.

Birattari and Kacprzyk (2009) use an F-Race evaluation algorithm, which applies the non-parametric Friedman test during each evaluation round to look for evidence that any candidates are inferior to the others with statistical significance and should be discarded. Birattari and Kacprzyk state that using random sampling for generating candidate evaluations significantly improves performance over methods that require manual interaction such as full factorial design. Among the iterative generate-evaluate methods of the algorithm configuration approaches, the most common methods can be grouped into heuristic search based methods and model based optimization approaches. In the first category falls the iterated F-Race approach proposed by Balaprakash et al. (2007), which is an iterative approach to the method designed by Birattari and Kacprzyk, where after each iteration the probabilistic model for generating the candidate solutions is updated. López-Ibáñez et al. (2016) extended and improved the iterated F-Race approach and provided the software package irace, which has been applied in different scenarios and shows state of the art results until today. Barbosa and Senne (2017) propose HORA, a heuristic oriented racing algorithm, which is an iterative approach of using Design of Experiments in order to define the parameter search space before applying racing to find good configurations. Meta evolutionary algorithms, which were first introduced by Mercer and Sampson (1978), have shown to provide competitive results as well. After several experiments by Grefenstette (1986) or Smit and Eiben (2009), Ansótegui et al. (2009) proposed GGA, the gender based genetic algorithm, which is the best-known variant and still shows state-of-the-art performance. GGA divides the population of candidate configurations into 2 genders, with competitive selection and fitness evaluation happening in only one of the gender groups, which leads to significant reductions in computing time and allows for bigger populations. REVAC, the parameter relevance estimation and value calibration, proposed by Nannen and Eiben (2007), is another representative of this group, which tries to estimate the expected performance of a configuration by measuring the relevance of the parameter settings. The ParamILS framework by Hutter et al. (2009) uses iterated local search to search the configuration space, with a one-exchange neighborhood, meaning that one parameter is changed at a time in each local search or perturbation step. Model based optimization approaches employ response surface models or surrogate models to model the relations between algorithm performance and parameter configurations in order to sequentially guide the sampling of new configurations. The most common approaches in that category are SMAC, which stands for sequential model based optimization for general algorithm configuration, by Hutter et al. (2011) and SPO (sequential parameter optimization) by Bartz-Beielstein et al. (2005). As SMAC is the algorithm configuration tool employed in the approach presented in this thesis, the reader is referred to the following sections for a more detailed description. In a recent algorithm configuration survey by Huang et al. (2020) the authors name SMAC as the most powerful parameter configurator while claiming that the iterated F-Race and ParamILS approaches provide state-of-the-art results as well.

All these automated algorithm configurators however work on a per-set basis, meaning that they find a good algorithm configuration for an entire set of instances. As mentioned in the previous section, truly instance specific algorithm configuration remains an open challenge due to the often infinite number of possible algorithm configurations. For the most part, there are two promising approaches tackling this problem, which are Hydra by Xu et al. (2010) and ISAC by Kadioglu et al. (2010). Hydra by Xu et al. (2010) first determines a default configuration for the entire training set. It then iteratively builds a portfolio of optimized configurations by modifying the performance metric so that it judges a newfound configuration with its actual performance, if it outperforms the portfolio, or with the portfolio's best performance, if it performs worse than the portfolio, thereby rewarding configurations working well on a subset of instances without penalizing them if they do not perform well on other instances. The ISAC approach proposed by Kadioglu et al. (2010) on the other hand is based on stochastic offline programming. Here, instances are first clustered and the algorithm is then configured for each cluster using the GGA configurator by Ansótegui et al. (2009), with unseen instances then simply being assigned to one cluster. Configuring the algorithm on clusters of instances thereby has the advantage that it prevents overfitting and provides generalized configurations that work well on similar instances as well. While the basic concept of ISAC is still regarded as state-of-the-art today, it has been tweaked and improved over the years. Malitsky et al. (2014) for example propose evolving the portfolio of configurations found by

ISAC over time, whereas Ansótegui et al. (2016) propose using automated algorithm selection based on classification tasks to assign instances to the portfolio of configurations found by ISAC instead of simply assigning them to the separate clusters, naming their approach ISAC++.

A very recent approach proposed by Ansótegui et al. (2021) used instance oblivious algorithm configurators like SMAC or GGA to configure the algorithm on the entire training set, however, they do not only collect the final result of the configurator but save every single configuration tried in the search process. They then solve every training instance using every one of the saved configurations and add the ones which outperform all others on at least one instance to the final portfolio of configurations before assigning unseen instances to configurations in the portfolio using algorithm selection approaches.

## 4. Research Focus

With the theoretical background as well as the current state of the scientific literature laid out, this section now identifies any gaps in the literature and defines the position of this thesis in trying to close these research gaps.

First, as mentioned in the literature review, the scientific literature on algorithm selection problems is largely problem domain specific and while there exist several papers on algorithm selection for the traveling salesman problem, the literature on algorithm selection for the vehicle routing problem is relatively sparse. However, Rasku et al. (2016) claim that most features defined for the traveling salesman problem can be adapted and reused to describe the vehicle routing problem as well, as the latter acts as a generalization of the former. As this paper by Rasku et al. (2016) was the only one found presenting a comprehensive set of VRP instance features, the first focus of this thesis is therefore to test this claim by collecting a set of features describing vehicle routing problem instances from different sources within the literature that can not only be used for instance specific algorithm selection, but also instance specific algorithm configuration, thereby expanding the scope of the works by Rasku et al. (2016, 2019).

1. *Can a representative set of instance features describing a vehicle routing problem instance be found that can reliably predict its best performing metaheuristic as well as the selected metaheuristic's best performing configuration?*

As described in the previous section, there already exists extensive literature on both automated algorithm selection and automated algorithm configuration. While many papers and literature surveys in both domains mention each other as a related problem, the two domains are interestingly still treated as two separate ones and there is, to the best of my knowledge, no scientific literature on combining the two domains into one approach. Therefore, the second focus of this thesis aims at verifying, whether the two domains can in fact be regarded as separate or whether a combined and integrated approach can improve the performance compared to using each one individually or even sequentially.

2. *Can an integrative approach combining instance-specific algorithm selection and instance-specific algorithm configuration improve the average solution quality over a set of VRP instances compared to using these approaches individually or sequentially?*

Lastly, there are many aspects and processes within these domains that are very time consuming, for example the calculation of some instance features like local search probing features or the algorithm configuration process. Having a time constrained, real-world application in mind, the third focus of this thesis lies in trying to achieve these prediction qualities and performance gains within a reasonable time frame, which is measured in the model runtime of solving an unseen instance.

3. *Assuming a time constrained environment, can the points above be achieved within a reasonable time frame?*

# 5. Methodology

## 5.1.Problem Description – Vehicle Routing Problem

The vehicle routing problem was first introduced by Dantzig and Ramser (1959) as the truck dispatching problem. It is described as a generalization of the traveling salesman problem, by introducing additional conditions. Like in the TSP, the VRP consists of a depot and a set of customers that have to be visited. However, deliveries with a specified weight are made at each node other than the depot and the salesman has a predefined capacity. If the capacity of the salesman is larger than the sum of all demands, the problem remains a classic travelling salesman problem, otherwise the solution is allowed to include multiple trips, which can be interpreted as either the salesman returning to the depot to refill, or more commonly as multiple salesmen or vehicles being used in the solution.

The formal definition of the vehicle routing problem below is based on, however not replicating exactly, the formulations by Dantzig and Ramser (1959) and Bai et al. (2021):

**Sets and constants:**

$$A \ldots set\ of\ arcs$$

$$V \ldots set\ of\ vertices$$

$$0 \ldots depot$$

$$n \ldots number\ of\ customers$$

$$d_i = demand\ of\ customer\ i$$

$$c_{ij} = costs\ of\ traversing\ arc\ (i,j)$$

$$C = vehicle\ capacity$$

**Decision variables:**

$$x_{ij} = \begin{cases} 1 & if\ arc\ (i,j)\ is\ part\ of\ the\ solution \\ 0 & otherwise \end{cases}$$

$$y_{ik} = \begin{cases} 1 & if\ customer\ i\ is\ served\ by\ vehicle\ k \\ 0 & otherwise \end{cases}$$

$$m \ldots number\ of\ vehicles\ used$$

**Objective function:**

$$\sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \rightarrow min \qquad (1)$$

**Constraints:**

$$\sum_{i \in V\backslash\{j\}} x_{ij} = 1 \qquad \forall j \in V\backslash\{0\} \qquad (2)$$

$$\sum_{j \in V\backslash\{i\}} x_{ij} = 1 \qquad \forall j \in V\backslash\{0\} \qquad (3)$$

$$\sum_{i \in V\backslash\{0\}} x_{0i} = \sum_{j \in V\backslash\{0\}} x_{j0} = m \qquad (4)$$

$$\sum_{i} y_{ik} * d_i \leq C \qquad \forall k \in \{1, \dots, m\} \qquad (5)$$

The objective (1) of the vehicle routing problem is to minimize the total distance traveled across all routes, whereby the number of vehicles and therefore the number of routes is not fixed in advance. The solution thereby has to satisfy a set of constraints. Equation (2) ensures that exactly one vehicle arrives at every node except the depot. Constraint (3) on the other side ensures that exactly one vehicle also leaves every node except the depot. Constraints (2) and (3) also implicitly ensure the continuation of flow throughout the route, as there cannot be a node where one vehicle arrives but none leaves, or vice versa. Constraint (4) enforce that the number of vehicles leaving the depot to any other node equals the number of vehicles arriving at the depot from any other node, which both equal the chosen number of vehicles $m$. Therefore, every route must start and end at the depot. Constraints (2) – (4) therefore also prevent the formation of subtours or loops. Lastly, constraint (5) ensures that the sum of the demands of the customers visited by vehicle k does not exceed the vehicle capacity.

## 5.2.Solution Heuristics

For solving combinatorial optimization problems there generally exist two types of algorithms, exact solvers and approximate solvers, also called heuristic methods. Exact solvers aim to explore the entire search space of possible solutions and guarantee to find the optimal solution to the problem. However, as many combinatorial optimization problems like the vehicle routing problem are classified as $NP - hard$ problems, meaning that there exist no algorithms capable of solving it in polynomial time, exact solvers can have, in the worst-case, an exponential runtime in regard to the problem size. As this would not be feasible for large instances, heuristic methods were developed, which aim to find good or near-optimal solution within a reasonable time frame, however without being able to guarantee the optimality of a found solution. Heuristic methods can be generally classified into construction heuristics, improvement heuristics and metaheuristics. These 3 classes, as well as their respective members used in this approach, will be introduced in this section. (Blum & Roli, 2003)

### *5.2.1. Construction Heuristics*

Construction heuristics aim to construct good solutions within a reasonable time frame. In this approach there are 4 methods applied, a simple nearest neighbor heuristic, the savings algorithm by Clarke and Wright (1964), a cluster first, route second approach based on the sweep algorithm by Gillet and Miller (1974) and a random approach, that is merely acting as a baseline. The nearest neighbor heuristic is one of the simplest one at that. In regards to the vehicle routing approach, it starts by marking all customer nodes as unvisited, and then, starting at the depot, always visits the nearest unvisited customer from the current position and flagging the respective one as visited. As soon as the next customer would exceed the vehicle capacity, the vehicle returns to the depot instead and starts again from there until all customers have been visited. (Laporte & Semet, 2002) The savings algorithm proposed by Clarke and Wright (1964) starts by initially assigning each customer to a sperate vehicle, which starts from the depot, goes to the customer and returns back home to the depot. Next, for each pair of nodes $i$ and $j$, where $i$ has to be different from $j$, a savings value $s_{ij} = d_{i0} + d_{0j} - d_{ij}$ is computed, that could be realized by combining the two tours going through these points into one. The list of all savings is then sorted and the feasible merge leading to the highest savings is performed. Feasible in that regard means that the resulting tour must not exceed the vehicle capacity and that both $i$ and $j$ must be at the beginning or end of their respective tours. As soon as no feasible merge resulting in cost savings is possible, the heuristic ends. (Clarke & Wright, 1964) As stated above, the cluster first, route second heuristic is based on the sweep algorithm proposed by Gillet and Miller (1974). In the first step, the locations of all customer nodes are calculated in polar coordinates with the depot acting as the central point and are ranked by their polar angles. The sweep algorithm goes through the entire list in a clockwise manner, adding nodes to a cluster until the vehicle capacity would be violated, in which case a new cluster is formed. In the second step, the routing problem within each cluster, which is now effectively a traveling salesman problem is solved. In the approach in this thesis, a nearest neighbor algorithm is used for solving the routing problems. (Gillett & Miller, 1974) Lastly, the random construction heuristic simply adds a random unvisited node to a route until the capacity limit is reached before continuing this approach with a new route until all nodes have been visited.

In order to allow the use of these construction heuristics in population-based metaheuristics, a randomized version of the 3 mentioned construction heuristics is implemented as well. In the randomized version of the nearest neighbor heuristic, a random starting point for the tours is selected. In the randomized savings algorithm, one of the top 3 merges resulting in the highest savings is selected at random, and in the cluster first, route second approach a random starting point for the sweep algorithm is selected.

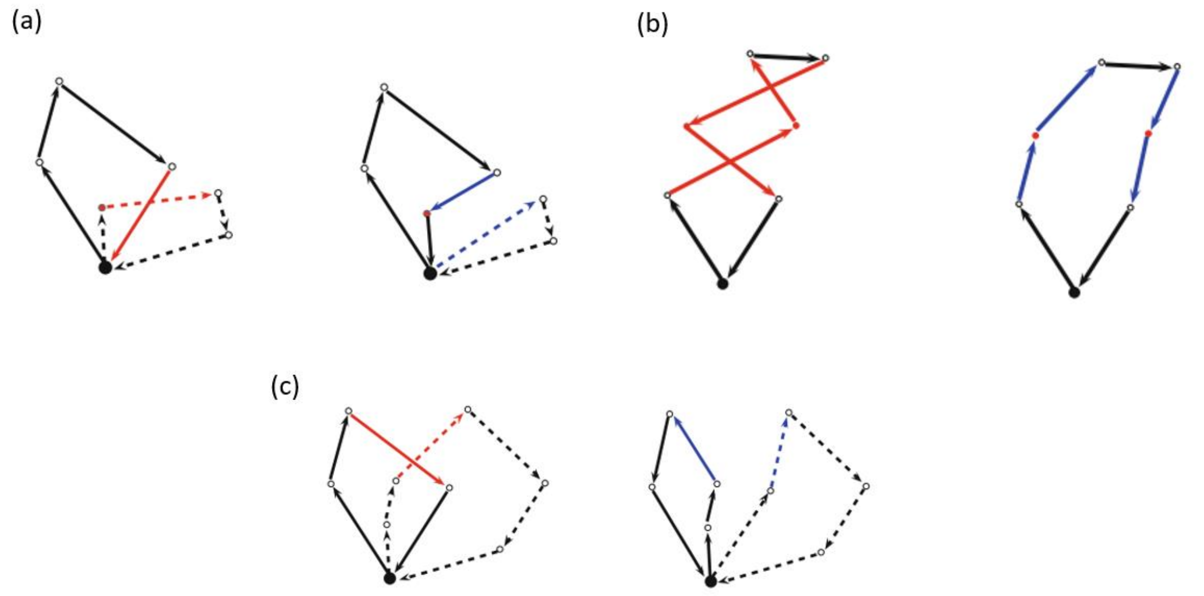
### *5.2.2. Local Search Heuristics*

Improvement or local search heuristics take an already constructed solution and aim to improve it in an iterative approach until no more improvement is possible. This is achieved by exploring the neighborhoods $N(x) \subset S$, which is a subset of the feasible solution space $S$, of the current solution $x$. When choosing the neighboring solution $x' \in N(x)$ there are two options, either choosing the first improvement, meaning to select the first found neighbor where $f(x') < f(x)$, or best improvement, where the algorithm evaluates all members of the neighborhood before selecting the best possible option. As soon as no more improvement can be found within the defined neighborhood, meaning that $\forall x' \in N(x) : f(x) \leq f(x')$ for a current solution $x$, the search has reached a local optimum. (Blum & Roli, 2003; Groër et al., 2010)

With the definition above in mind, there are 2 major characteristics of different local search methods, the definition of the applied neighborhood structure and the choice between first improvement and best improvement. The search neighborhood is defined as a subset of the feasible solution space $S$, that can be reached by performing a single step of a specified operator from the current solution. The operators and therefore the neighborhoods used in this thesis are the move operator, the swap operator and the 2-opt operator. The move operator describes moving a single node to any other point in the solution, either within the same route or into a different route. The swap operator is defined as swapping the position of 2 nodes in the solution, once again either within the same route or between different routes. Lastly, the 2-opt operator describes the process of cutting 2 edges in the solution before reconnecting the now loose ends in a different way. (Groër et al., 2010) Figure 5 shows an example for each of those three operators. In the implementation used in this approach each of those methods are included in both their first improvement and their best improvement variation, resulting in a total of 6 different local search methods.

**Figure 5**

*Local Search Methods: (a) Move, (b) Swap, (c) 2-Opt (Groër et al., 2010, p. 82)*



### *5.2.3. Metaheuristics*

Local search heuristics do however have some disadvantages. First, they are problem specific, which means that a local search operator designed for one problem domain cannot be used on solving an instance of a different problem domain. Second, local search heuristics always descend into a local optimum, where they end. There is however no guarantee that this is also the global optimum. Metaheuristics aim to compensate these disadvantages. They try to be problem-independent by often incorporating a modular structure and simply employing problem-specific construction or local search heuristics. Next, they often try to escape local optima and try to guide the search into promising areas of the search space. (Gendreau & Potvin, 2010) After a formal definition of metaheuristics, this section will introduce the 4 metaheuristics that were used in this approach.

Blum and Roli (2003) collected pieces from several definitions of metaheuristics from the literature to create the following list of characteristics that they see as most defining for a metaheuristic:

- "Metaheuristics are strategies that "guide" the search process.
- The goal is to efficiently explore the search space in order to find (near-) optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.

- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Todays more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.” (Blum & Roli, 2003, pp. 270-271)

Metaheuristics can thereby be classified into single-solution-based metaheuristics and population-based metaheuristics. Single-solution-based metaheuristics work on a single solution, which they modify, perturb, and improve while trying to find the optimal solution. Population-based metaheuristics work on an entire set of solutions, often creating a new population in every iteration by taking promising parts of existing solutions. (Blum & Roli, 2003)

As the focus of this thesis lies on the ability to characterize problem instances with a representative feature set and using this set for a combined approach of instance-specific algorithm selection and configuration, the overall performance and final solution quality of the solvers is not the main priority, but rather that they are designed in a way that allows for testing the proposed approach. Therefore, the decision was made to use a set of 4 metaheuristics, that were implemented from scratch in the programming language Python, instead of integrating state-of-the-art solvers. Self-implementing the metaheuristics provides the possibility of fine tuning the algorithms, such that each solver is competitive within the group and each solver outperforms all others on a subset of the instances. Another advantage of self-implemented the solvers is that the number of free parameters can be controlled. A decision within this implementation process was made to use 4 different metaheuristics, the single-solution based algorithms “Iterated Local Search” and “Simulated Annealing” as well as the population-based algorithms “Genetic Algorithm” and “Ant Colony Optimization”. These 4 solvers will be briefly introduced in this section. For a complete list of the solvers’ parameters including their types and allowed ranges, the reader is referred to the appendix.

Iterated Local Search (ILS) is one of the oldest and simplest single-solution-based metaheuristics. The basic idea first introduced by Baxter in 1981, ILS was continuously developed and modified over the years and continues to be used today. As can be seen in figure 6, ILS starts by constructing a solution using any given construction heuristics before performing local search to lead the solution into a local optimum. From now on and until a predefined termination criterion is met, which is a 30 second runtime limit in this case, the solution is perturbed, which means that it is changed just enough to escape the local optima before reapplying the local search algorithm. After making a decision on whether the newfound solution is accepted or discarded, in this case all new solutions are accepted, it is checked whether a new best solution was found before the next iteration starts. The degree of perturbation is thereby one of the most important aspects in designing and configurating the metaheuristic. If the solution is not perturbed enough, it might not escape the current local optimum, however if the solution is perturbed too much, this might equal a random restart and would results in losing any promising parts in the already found solution. (Lourenco et al., 2001) In the implementation used in this approach, the iterated local search metaheuristic has 4 free parameters. These are the choice of the construction heuristic to apply, the used local search heuristic and the neighborhood as well as the number of steps that are used for perturbation.

**Figure 6**

*Pseudocode for the Iterated Local Search Metaheuristic*

```
$IteratedLocalSearch(\,)$
1:   $x \leftarrow ConstructionHeuristic(\,)$
2:   $x \leftarrow LocalSearch(x)$
3:   $x^* \leftarrow x$
4:   $\boldsymbol{while}\ total\ runtime < 30s\ \boldsymbol{do}$
5:           $x' \leftarrow Perturb(x)$
6:           $x' \leftarrow LocalSearch(x')$
7:           $x \leftarrow Accept(x', x)$
8:           $\boldsymbol{if}\ z(x) < z(x^*)\ \boldsymbol{do}$
9:               $x^* \leftarrow x$
10:          $\boldsymbol{end\ if}$
11:  $\boldsymbol{end\ while}$
12:  $return\ x^*$
```

The simulated annealing (SA) algorithm introduced by Kirkpatrick et al. (1983) uses the analogy of annealing, which is the process of melting a material, before slowly cooling it down in order for it to reach its most stable state. The material has to be cooled slowly and in a controlled way, as in higher temperatures the material has a higher chance to diverge into an unwanted state. In SA, this idea is applied to solving combinatorial optimization problems, as can be seen in figure 7. First, a solution is constructed using a heuristic method and the temperature is set to an initial value. Until the time limit is reached, a random neighbor, using a predefined neighborhood, is selected and a decision is made on whether the neighbor is selected as the new solution or not. This non-deterministic decision is based on the current solution, the neighbor and the temperature, with a neighbor worse than the current solution having a higher probability to be accepted, if the temperature is high. After checking whether the new solution is better than the current best solution, the temperature is reduced using the cooling rate. (Kirkpatrick et al., 1983) Just as for the iterated local search metaheuristic the created implementation of the SA algorithm also has 4 free parameters including the applied construction heuristic, the neighborhood used for selecting a random neighbor, the initial temperature value as well as the cooling rate.

**Figure 7**

*Pseudocode for the Simulated Annealing Metaheuristic*

```
$SimulatedAnnealing(\,)$
1:   $T \leftarrow T_0$
2:   $x \leftarrow ConstructionHeuristic(\,)$
3:   $x^* \leftarrow x$
4:   $\textbf{while}\ total\ runtime < 30s\ \textbf{do}$
5:           $x' \leftarrow RandomNeighbor(x)$
6:           $x \leftarrow Accept(x, x', T)$
7:           $\textbf{if}\ z(x) < z(x^*)\ \textbf{do}$
8:               $x^* \leftarrow x$
9:           $\textbf{end if}$
10:          $T \leftarrow Update(T)$
11:  $\textbf{end while}$
12:  $return\ x^*$
```

The genetic algorithm (GA) is modeled after the concept of survival of the fittest in evolution. As depicted in figure 8, the algorithm starts by constructing an initial population. The members, often called chromosomes, are usually constructed by random construction. In this approach, randomized versions of more sophisticated heuristics are available next to a purely random construction. Once an initial population is constructed, the algorithm enters a loop and in each iteration a new population is created. If the algorithm is configured to use elite members, the best predefined number of members from the current population are also added to the new population. Next, a probabilistic selection operator is applied to select members of the current population acting as parents, with members with a higher fitness value having a higher chance to be selected. A crossover operator then combines the parts from the parents into new offspring solutions, which then have a chance to be mutated, meaning that a few random steps in a predefined neighborhood are performed. If the algorithm is configured to perform local search, a local search heuristic is applied to the new solutions at this point. This process of selection, crossover and mutation tries to select some of the best solutions within the population, reuse their most promising elements to form new solutions and potentially perturb these solutions to escape any local optima. This process is performed until the new population contains enough offspring solutions to have reached its defined population size. After checking whether the best solution of the current population is the best solution found so far, the algorithm continues with its next iteration, until the time limit is reached. (Whitley, 1994) The implementation of the GA exposes a set of 11 free parameters. These are the construction and local search heuristics to apply, the Boolean decision values on whether to perform local search on the initial solutions as well as any new solution, the population size, the number of elite individuals in the population, the selection and crossover operators to apply, the mutation rate and the neighborhood used for perturbation along with the number of perturbation steps.

**Figure 8**

*Pseudocode for the Genetic Algorithm Metaheuristic*

```
$GeneticAlgorithm(\,)$
1:  $p \leftarrow InitialPopulation(\,)$
2:  $\boldsymbol{if}$ $perform\ initial\ local\ search$ $\boldsymbol{do}$
3:      $p \leftarrow LocalSearch(p)$
4:  $\boldsymbol{end\ if}$
5:  $x^* \leftarrow BestSolution(p)$
6:  $\boldsymbol{while}$ $total\ runtime < 30s$ $\boldsymbol{do}$
7:      $newPopulation \leftarrow \emptyset \cup GetElites(p)$
8:      $\boldsymbol{while}$ $|newPopulation| < populationSize$ $\boldsymbol{do}$
9:          $p_1, p_2 \leftarrow Selection(p)$
10:         $o_1, o_2 \leftarrow Crossover(p_1, p_2)$
11:         $\boldsymbol{if}$ $random(\,) < mutationRate$ $\boldsymbol{do}$
12:             $o_1 \leftarrow Mutation(o_1)$
13:         $\boldsymbol{end\ if}$
14:         $\boldsymbol{if}$ $random(\,) < mutationRate$ $\boldsymbol{do}$
15:             $o_2 \leftarrow Mutation(o_2)$
16:         $\boldsymbol{end\ if}$
17:         $\boldsymbol{if}$ $perform\ local\ search$ $\boldsymbol{do}$
18:             $o_1 \leftarrow LocalSearch(o_1)$
19:             $o_2 \leftarrow LocalSearch(o_2)$
20:         $\boldsymbol{end\ if}$
21:         $newPopulation \leftarrow newPopulation \cup \{o_1, o_2\}$
22:     $\boldsymbol{end\ while}$
23:     $p \leftarrow newPopulation$
24:     $\boldsymbol{if}$ $z\big(BestSolution(p)\big) < z(x^*)$ $\boldsymbol{do}$
25:         $x^* \leftarrow BestSolution(p)$
26:     $\boldsymbol{end\ if}$
27: $\boldsymbol{end\ while}$
28: $return\ x^*$
```

The ant colony optimization (ACO) metaheuristic as introduced by Dorigo and Di Caro (1999) takes real ant colonies as an analogy and uses pheromone trails laid by ants as a long-term memory in order for the ant population to collectively solve an optimization problem. In real life ants lay pheromones along their paths, which evaporate over time, and following ants are more likely to take a path with a higher pheromone level. This concept is applied in this metaheuristic as well. For example, in routing problems a complete graph containing all nodes is constructed, with each edge between any two nodes having a pheromone value. When constructing a new solution, the algorithm is more likely to choose edges with a higher pheromone value. As depicted in figure 9, the algorithm starts by initializing the pheromone values of all edges to a predefined value. From there and until the time limit is reached, the algorithm iterates through multiple generations. In each generation a new population is constructed consisting of a predefined number of ants. In the construction of an ant, an adapted nearest neighbor heuristic is applied. At each node in the solution process a weight for any possible following node is calculated based on the pheromone value $\tau$ and distance $d$ of the respective edge, as well as the free parameters $\alpha$ and $\beta$. The edge to be traversed is then selected using a roulette wheel, with a high pheromone value or a low distance giving an edge a higher chance of being selected. As soon as a solution has been constructed, a local search heuristic can be applied before all the pheromone values are updated. In the update all pheromones evaporate by a predefined rate before new pheromones are laid along the routes taken in the new solution, whereby the amount of pheromones laid depend the quality of the solution. Once the time limit is reached, the best-found solution along this process is returned. (Dorigo & Di Caro, 1999) The implementation of the ant colony optimization metaheuristic exposes 7 free parameters. These are the initial pheromone value, the number of ants in the population, the values $\alpha$ and $\beta$, which control the respective importance of the pheromone value and the distance in calculating the weights used in the roulette wheel selection, the Boolean value of whether to perform local search on new solutions and, if so, the local search heuristic to apply, as well as the evaporation rate and a value $Q$, which controls the importance of the solution quality in calculating the amount of pheromones laid by an ant.

**Figure 9**

*Pseudocode for the Ant Colony Optimization Metaheuristic*

```
$AntColonyOptimization(\,)$
1:  $x^* \leftarrow null$
2:  $\tau \leftarrow InitializePheromone(\,)$
3:  $\boldsymbol{while}$ $total\ runtime < 30s$ $\boldsymbol{do}$
4:          $solutions \leftarrow \emptyset$
5:          $\boldsymbol{while}$ $|solutions| < populationSize$ $\boldsymbol{do}$
6:              $ant \leftarrow ConstructSolution(\tau, \alpha, \beta)$
7:              $\boldsymbol{if}$ $perform\ local\ search$ $\boldsymbol{do}$
8:                  $ant \leftarrow LocalSearch(ant)$
9:              $\boldsymbol{end\ if}$
10:             $solutions \leftarrow solutions \cup \{ant\}$
11:             $\tau \leftarrow UpdatePheromone(\tau, solutions, \rho)$
12:             $\boldsymbol{if}$ $z\big(BestSolution(p)\big) < z(x^*)$ $\boldsymbol{do}$
13:                 $x^* \leftarrow BestSolution(p)$
14:             $\boldsymbol{end\ if}$
15:         $\boldsymbol{end\ while}$
16: $\boldsymbol{end\ while}$
17: $return\ x^*$
```

## 5.3.Instance Analysis - Feature Set

As previously stated, the quality of the feature set describing a problem instance is one of the most important aspects in automated algorithm selection or configuration. The goal is to find a set of representative features while keeping the dimension of the resulting dataset in mind in order to avoid the curse of dimensionality in the machine learning process. The trend in the scientific literature in the recent past has been towards including a more and more comprehensive set of features before using principal component analysis to project the data into a lower dimension. While many scientific works state that various feature selection approaches could be applied instead of or in addition to PCA, many of them still rely solely on PCA. (Rasku et al., 2019) This section of the thesis introduces the full feature set used in this approach together with the applied methods used to construct and calculate these features.

While the scientific literature regarding feature sets and feature extraction from vehicle routing problems is relatively sparse, there exist quite a few works on the characterization of traveling salesman problem instances. Due to the fact that the VRP poses as a generalization of the TSP, most of the features in the proposed feature sets can however be reused or adapted to describe VRP instances as well. As written above, feature sets in the scientific literature have been gradually expanded over the past years, which is why no new features were introduced in this thesis, but the set, depicted in figure 10, was rather composed of pieces from several different approaches from the literature.

**Figure 10**

*Full VRP Feature Set*

| VRP-Domain specific features | | |
|---|---|---|
| vr-1 | number of clients | 1 |
| vr-2 | client demands | 7 |
| vr-3 | avg clients per vehicle | 1 |
| vr-4 | min number of vehicles | 1 |
| vr-5 | capacity tightness | 1 |
| vr-6 | ration of max demand to vehicle capacity | 1 |

| node distribution features | | |
|---|---|---|
| nd-1, 2 | Depot location (x, y) | 2 |
| nd-3 | distances from clients to depot | 7 |
| nd-4 | values in distance matrix | 7 |
| nd-5 | fraction of unique distances in distance matrix (1f, 2f, 3f, 4f) | 4 |
| nd-6, 7 | location of the centroid (x, y) | 2 |
| nd-8 | distance depot-centroid | 1 |
| nd-9 | distances from clients to centroid | 7 |

| cluster features | | |
|---|---|---|
| cl-1, 2 | number of clusters (abs., rel.) | 2 |
| cl-3 | ratio of max. cluster demand to vehicle capacity | 1 |
| cl-4 | ratio of cluster outlier demand to vehicle capacity | 1 |
| cl-5 | relative number of core nodes | 1 |
| cl-6 | relative number of edge nodes | 1 |
| cl-7 | relative number of outlier nodes | 1 |
| cl-8 | relative cluster sizes | 7 |
| cl-9 | cluster reach | 7 |

| geometric features | | |
|---|---|---|
| geo-1 | area of the enclosing rectangle | 1 |
| geo-2 | area of the convex hull | 1 |
| geo-3 | ratio of points on the convex hull | 1 |
| geo-4 | convex hull edge lengths | 7 |
| geo-5 | min distance from non-hull nodes to the hull | 7 |

| Minimum Spanning Tree (MST) features | | |
|---|---|---|
| mst-1 | sum of MST edge distance | 1 |
| mst-2 | distances of MST edges | 7 |
| mst-3 | node degrees of MST | 7 |
| mst-4 | MST depth from the depot | 7 |

| Nearest Neighbor features | | |
|---|---|---|
| nn-1 | distance to 1st nearest neighbor | 7 |
| nn-2 | angle between 2 nearest neighbors | 7 |
| nn-3 | cosine similiarity between 2 nearest neighbors | 7 |
| nn-4 | node degrees in 2NN graph | 7 |
| nn-5, 6 | number of 2NN graph components (abs., rel.) | 2 |
| nn-7 | size of 2NN components | 7 |
| nn-8 | ratio of max. component demand to vehicle capacity | 1 |
| nn-9 | reach of components in 2NN graph | 7 |

| Computational effort | | |
|---|---|---|
| cpu | CPU time for computing all features | 1 |

Before calculating all the features listed here, the coordinates of the instance file are normalized to create better comparability between the different instances. While many approaches in the literature, for example Hutter et al. (2014) or Pihera and Musliu (2014), squeeze the coordinates into a predefined square or rectangle, this approach is not taken in this model as it has the potential to contort the dimensions. Instead, all x-coordinates were scaled into the range from 0 to 400, before using the same scaling factor used on the x-axis to all y-coordinates as well in order to maintain the correct aspect ratio, similar to the proposed procedure by Rasku et al. (2016).

The instances used in this thesis are grouped into 6 different groups plus the additional feature of the computational effort, measuring the CPU time for computing the features in the 6 main groups, thereby loosely following the structure of Rasku et al. (2016). The tables detailing the 6 groups in figure 10 consist of 3 columns. The left one shows the title of the respective feature or feature subgroup, the middle column shows a short description of the features and the column on the right show the number of actual features in the given subgroup. The features subgroups containing 1, 2 or 4 features are self-explanatory from their description, the groups containing 7 features consist of 7 statistical metrics of the given list of values. These metrics are the mean, minimum, maximum, standard deviation, skewness, kurtosis and the coefficient of variation.

The 6 main groups of features can be briefly described as follows:

- **VRP-domain specific features**: This group, as proposed for example by Steinhaus (2015), describes features specific to the domain of the vehicle routing problem, which can be calculated directly from the instance file. The minimum number of vehicles is calculated by dividing the total demand by the vehicle capacity. As in the model used here the number of vehicles is not predefined, the minimum number of vehicles is used to calculate the average number of customers per vehicle and the capacity tightness, which is the total demand divided by the total capacity, as well.
- **Node distribution features**: The node distribution features are a group of features that can describe the locations of the single nodes as well as the distances between them and can be calculated from the distance matrix of the problem instance. Smith-Miles and van Hemert (2011) proposed taking the relative number of distinct distances in the distance matrix with 1, 2, 3 or 4 decimals. They also proposed calculating the centroid of all nodes, which is placed at the average of all points on each axis. (Rasku et al., 2016; Smith-Miles & van Hemert, 2011)
- **Cluster features**: The existence of clusters in the distribution of the customers, or the lack thereof, have shown to have a significant impact on the performance of many heuristics, which is why a separate feature group containing a set of cluster features was created. The DBSCAN approach, with an epsilon value of $\varepsilon = \frac{400}{\sqrt{n}-1}$ , n being the number of nodes, and a minimum sample size of 4, was used here to create clusters of nodes. DBSCAN is a density-based clustering approach that looks at individual nodes and checks how many nodes lie within the $\varepsilon$-distance of the original one. If the number is greater than or equal to the minimum sample size, a cluster is formed, otherwise the

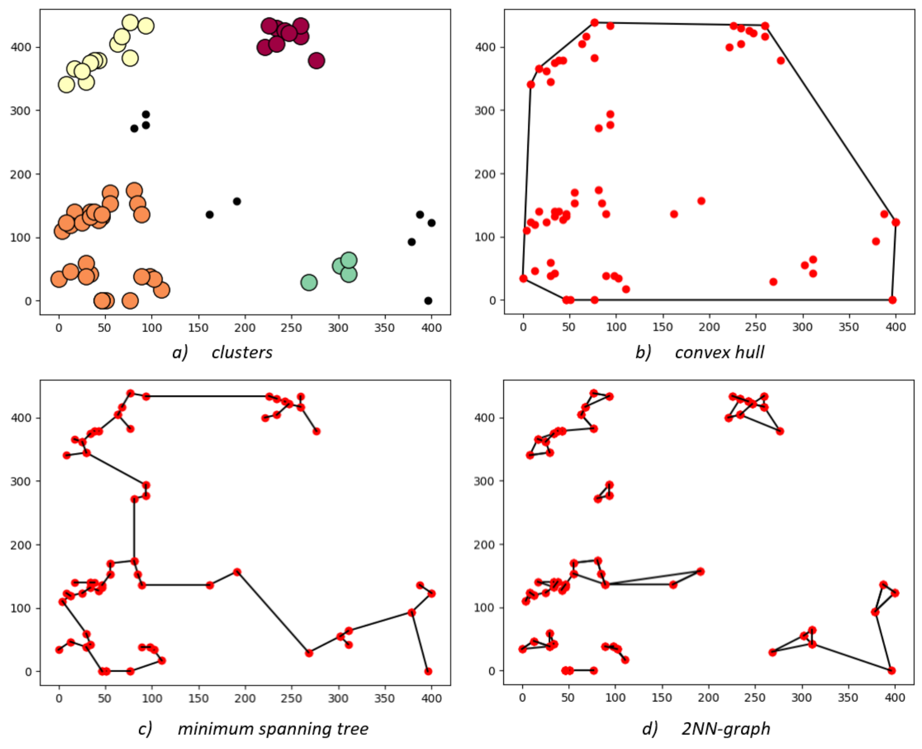
node is flagged as noise or an outlier. If a cluster was formed, the first node is flagged as a core node with all nodes in its vicinity being flagged as edge nodes. The algorithm then tries to expand the cluster by looking for additional nodes lying within the $\varepsilon$-distance of the edge nodes. If new nodes are found, the respective edge node becomes a core node and the new nodes become edge nodes. (Berthold et al., 2020) The relative cluster sizes describe the number of nodes within each cluster divided by the total number of nodes and the cluster reach defines the maximum distance of cluster node to its cluster centroid. (Smith-Miles & van Hemert, 2011; Steinhaus, 2015) An example of this clustering can be seen in figure 11)a).

- **Geometric features**: The group of geometric features describes the overall distribution of all nodes by looking at the enclosing rectangle, which is the area defined by the maximum distance along the x- and the y-axis respectively, and the convex hull, which is the minimum convex shape along a subset of nodes enclosing all other nodes, as can be seen in figure 11)b). (Pihera & Musliu, 2014)
- **Minimum spanning tree (MST) features**: The minimum spanning tree, depicted in figure 11)c) is defined as the connected graph, meaning it contains all nodes, with no cycles and with minimum costs. The node degrees describe for each node to how many different nodes it is connected. With the depot defined as the MST root node, the list of MST depths describes the respective number of edges that need to be traversed in order to get from the root node to each leaf node. (Hutter et al., 2014; Rasku et al., 2016)
- **Nearest neighbor features**: While for the first 3 features of this group only the distance to the first nearest neighbor and the angle between the 2 nearest neighbors are considered, for the rest of the group a 2-nearest-neighbor graph has to be constructed. In this graph, each node has outbound connections to its 2 nearest neighbors, as illustrated in figure 11)d). The node degrees however do consider all connections, not only the outbound ones, therefore the values can be greater than 2. The choice for setting the number of neighbors in constructing the k-nearest-neighbor graph to 2 was made on a trial-and-error basis. If k was set to a higher number than 2, many instances used in the instance set contained only one single connected component in the entire graph, which results in a very low variance within the dataset. On the other side, if every node would only be connected to its 1st nearest neighbor, it turned out that most instances contained a high number of components containing only 2 nodes connected to each other, which makes the number of graph components directly correlated to the

number of nodes, therefore k was set to 2. Lastly, just as for the clusters the component reaches are defined as the maximum distance from to component centroid to any of its nodes. (Pihera & Musliu, 2014)

**Figure 11**

*Example of Node Clusters (a), Convex Hull (b), MST (c) and 2NN-Graph (d)*



Several instance characterization approaches from the literature, which focus on the traveling salesman problem, also include probing features like local search probing or branch and cut probing in the feature set. Local search probing features often start by constructing a solution and recording its solution quality, before performing one or multiple runs of local search heuristics and recording metrics like the solution quality after running the local search heuristic, the number of steps performed by the heuristic, the improvement per step or the total improvement achieved. Branch and cut probing features usually imply a predefined cut-off time and record the values of the bounds and the improvements per cut among other features. (Hutter et al., 2014) However, as these two types of features are usually responsible for the majority of the runtime in the feature extraction phase and due to the fact that this thesis has a time minimizing approach in mind, they were not included in the feature set for this approach. Future research might however experiment with them and analyze if the added explanatory power of these features is worth the increased computational effort.

## 5.4.Approach for Combined Algorithm Selection and Configuration

As mentioned in the introductory section, the goal of this thesis is to create an integrated approach combining both instance-specific algorithm selection and instance-specific algorithm configuration into one model. This is done with a 2-step approach of first creating a base model and then using the created data containing instance labels to perform feature selection and rerunning the entire process to create the final model. This section will describe the entire end to end process as well as used techniques and approaches.

### *Data selection and first data preprocessing*

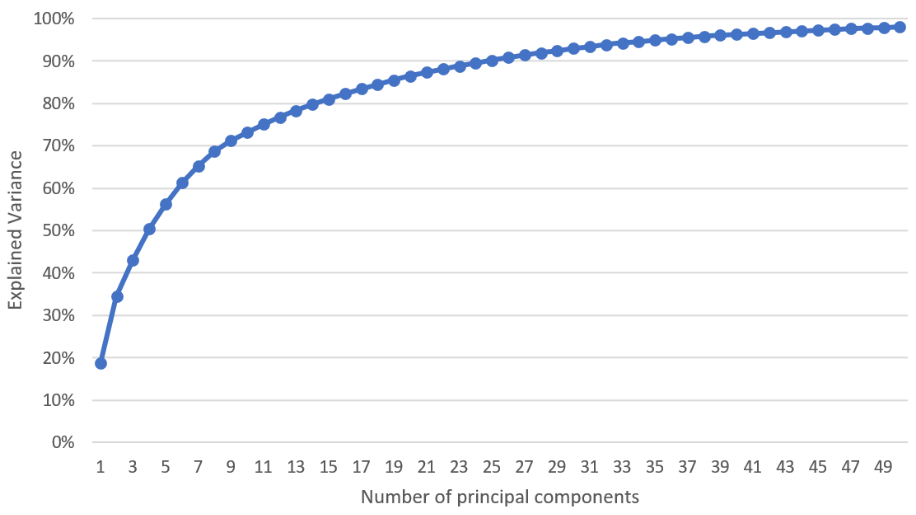
The process starts by creating the data sets used in the approach. As will be further detailed in a later section, a set containing 250 problem instances is used which is split into 200 training instances and 50 test instances. For all the training instances the entire feature set described above is calculated and saved into a data set. The 200 values for each feature in this data set are then normalized using a standard normal distribution. The respective parameters used in the normalization process are saved, as any unseen instance solved later in the evaluation step will need to be normalized according to the distribution of the values in the training set.

Once the 200 normalized feature vectors are available, a first step of data preprocessing is conducted. Many approaches from the literature only use methods like principal component analysis to project the vectors into a lower dimensionality. Since there are no data labels available yet that could be used for sophisticated feature selection methods, the procedure of these approaches is followed at this point. However, first it is checked whether the data set contains any features with no variability, which are removed before continuing.

Principal component analysis is now performed on the dataset. The number of principal components selected is chosen based on a predefined explained variance threshold of 80%, resulting in 15 principal components being selected, as can be seen in figure 12.

**Figure 12**

*Explained Variance by the Number of Principal Components*



***Instance-specific algorithm configuration***

Now that the first version of the preprocessed data is available, the model proceeds to performing instance-specific algorithm configuration. Here, ISAC, the state-of-the-art approach by Kadioglu et al. (2010) is used as a reference. ISAC builds on the stochastic offline programming model introduced by Malitsky and Sellmann (2009) by first clustering all instances before computing ideal parameter settings for each cluster. This approach of configuring the algorithms for clusters of instances and thus solving a per-set algorithm configuration problem has the advantage that the resulting configuration is robust, meaning that it prevents over-tuning and allows the configuration to generalize to similar instances. (Kadioglu et al., 2010)

In this approach a k-Means clustering method is used for clustering the training instances using the scikit-learn library within Python. As briefly described in the theoretical background section, k-Means clustering is a prototype-based clustering technique. While “k” stands for the number of clusters that are to be found, “Means” stands for the fact that spherical cluster distributions are assumed around the mean value of all its members. The objective of the algorithm is to minimize the total intra-cluster variances, which is usually calculated as the sum of the squared distance from all members to the cluster centroid. The algorithm starts by randomly selecting k cluster centroid points, before assigning each data point to its closest cluster centroid. Assuming equal dimensionality across the entire dataset, the distance is usually calculated using the Euclidean distance. After all points are assigned to a cluster, the cluster centroids are now recalculated as the means of each cluster. With the new cluster centroids calculated, each point is once again assigned to its nearest cluster centroid. This iterative process continues until some termination criterion, usually an iteration without any instance being assigned to a different cluster or a predefined number of iterations, is met. (Berthold et al., 2020)

However, one problem with prototype-based clustering methods is that the number of clusters has to be known beforehand. There are some methods for determining the optimal number of clusters, like the elbow method or the silhouette scores, which will be described later. At this point the number of clusters used is set to a manually defined default value of 4.

With the training instances grouped into 4 clusters, every metaheuristic is now automatically configured for every cluster. This is done by employing the state-of-the-art algorithm configurator SMAC by Hutter et al. (2014). SMAC, standing for sequential model-based algorithm configuration, is based on sequential model-based optimization, which iterates between fitting models and using them for choosing configurations to explore. Model-based optimization approaches use response surface models for modelling the dependence of an algorithm's performance on its parameter settings. SMAC is thereby the first model-based algorithm configurator allowing for not only numerical, but also ordinal and categorical parameters, which is achieved by using a random forest algorithm, as well as optimizing configurations for entire sets of instances by including instance features in the optimization process instead of optimizing only on single instances like similar approaches that came before. Therefore, SMAC is not limited to a single problem domain but can be applied to any general algorithm and is up to date still regarded as the most powerful algorithm configurator available. (Huang et al., 2020; Hutter et al., 2011)

SMAC was configured to perform 500 algorithm runs for each metaheuristic and each cluster, while being provided the preprocessed data set as an instance file and aiming to minimize the mean relative delta to the otimal or the best know solution.

***Collection of intermediate performance data and base model***

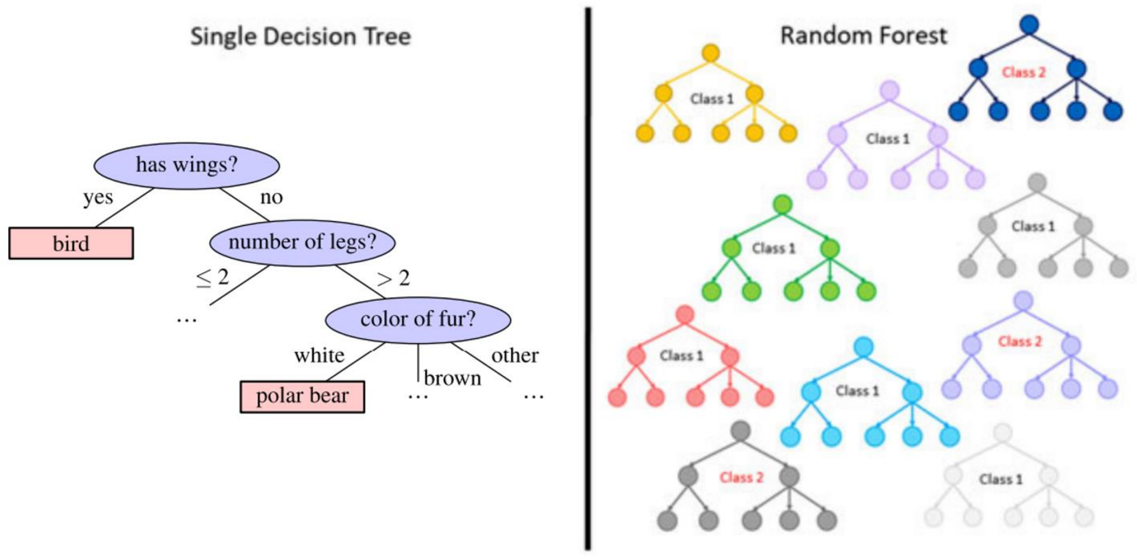
In this next phase, performance data over the entire training set is collected. Every metaheuristic has now been configured with 4 different parameter settings, one for each cluster. The approach therefore now goes through the entire list of training instances and solves every one with all 4 metaheuristics, which are configured with the parameter settings for the respective instance's cluster. These 4 results are then compared with each other and the metaheuristic providing the best result is set as the class label for this instance.

With the instance labels created, it is now possible to create the base model of this approach and collect the intermediate results. Several different classification methods were tested at this point. Among them were decision tree classifiers, random forest classifiers, support vector machines, neural networks or naïve bayes classifiers, once again using their implementations from the scikit-learn library in Python. Out of those it was shown that the random forest classifier consistently provided the best prediction quality. Pihera and Musliu (2014) for example, who tested a Bayesian network, decision tree, random forest, k-nearest-neighbor classifier and support vector machine while performing automated algorithm selection for the TSP, also came to the conclusion, that the random forest classifier yields the best results. (Pihera & Musliu, 2014) A random forest classifier is an ensemble method consisting of multiple decision trees. Decision trees, which can be used in both classification and regression tasks, aim to predict the target value by extracting simple decision rules from the data set. This is done in a hierarchical, tree-like decision structure that makes the classification process easily readable by going through the tree from the root to its leaves, as can be seen in figure 13 on the left.

In a random forest, depicted in figure 13 on the right, a number of n decision trees are constructed and every one of them is trained on a random subset of the training data set. With all those trees being trained slightly differently, they can all produce slightly different predictions as well. For the final prediction, the prediction of every tree is taken into account. Given the type of prediction that is required, there are different approaches to merge the predictions. In regression methods, where a continuous value is required, usually the average of all predictions is taken as the final prediction. In classification tasks popular approaches are to either take the class that received the most votes or to compute an average of the class probabilities of each tree and return the class with the highest average class probability. (Berthold et al., 2020) In the implementation used in this approach, the latter approach is used for class prediction of the random forest.

**Figure 13**

*Single Decision Tree and Random Forest (Berthold et al., 2020, p. 221 (left), p. 310 (right))*



With all the necessary parts of the base model set up, the features of the test instances could now be extracted, the instance could be assigned to one class and one cluster and be solved by the resulting metaheuristic and configuration. A more detailed description of this process flow for unseen instances is provided in the next section.

### ***Feature Selection***

As previously stated, many automated algorithm selection approaches in the literature only use principal component analysis for preprocessing the data, which means that they stop at the stage of the base model. However, with the instance labels of the training set now available, it is possible to run additional feature selection methods and try to increase the predictive performance and as a result the overall performance of the model.

SMAC was used again for configuring the feature selection method. The parameters included the variance threshold, meaning how many features with no or low variance should be removed, the number of features that should be selected based on a “Select K Best” feature selection method, and the question whether PCA should be performed again and if so, what the variance threshold in determining the number of principal components should be. The select-k-best method ranks all features based on the ANOVA f-test, which is a popular approach in cases where the input variables are numerical and the output value is categorical. As ANOVA stands for analysis of variance, this method basically assigns an f-value to each feature, by analyzing the variances of the values of that feature in respect to the target class variable. It does that by comparing the within-class variance, which describes the variance of all values within one class, with the between-class variance, which is the variance of the mean values of each class. A good ranking feature would thereby have a low within-class variance and a high between-class variance, resulting in a high f-value. (Scheffe, 1999) SMAC, configured to perform 1000 algorithm runs and to minimize the prediction error, revealed that only features with no variance should be removed before the 24 best features, according to the ANOVA f-test, should be selected from the remaining features. Interestingly SMAC found that performing PCA on these remaining 24 features does not improve performance anymore, and therefore recommended not to perform PCA at all. This feature selection process resulted in the reduced feature list as shown in figure 14. A detailed list of these 24 features including their statistical metrics can be found in the appendix. Interestingly, at least one feature out of all 6 of the main feature groups was selected.

**Figure 14**

*Feature Set After Feature Selection*

| VRP-Domain specific features | | |
|---|---|---|
| vr-1 | number of clients | 1 |
| vr-2 | client demands | 7 |
| vr-3 | avg clients per vehicle | 1 |
| vr-4 | min number of vehicles | 1 |
| vr-5 | capacity tightness | 1 |
| vr-6 | ration of max demand to vehicle capacity | 1 |

| node distribution features | | |
|---|---|---|
| nd-1, 2 | Depot location (x, y) | 2 |
| nd-3 | distances from clients to depot | 7 |
| nd-4 | values in distance matrix | 1/7 |
| nd-5 | fraction of unique distances in distance matrix (1f, 2f, 3f, 4f) | 4 |
| nd-6, 7 | location of the centroid (x, y) | 2 |
| nd-8 | distance depot-centroid | 1 |
| nd-9 | distances from clients to centroid | 7 |

| cluster features | | |
|---|---|---|
| cl-1, 2 | number of clusters (abs., rel.) | 1/2 |
| cl-3 | ratio of max. cluster demand to vehicle capacity | 1 |
| cl-4 | ratio of cluster outlier demand to vehicle capacity | 1 |
| cl-5 | relative number of core nodes | 1 |
| cl-6 | relative number of edge nodes | 1 |
| cl-7 | relative number of outlier nodes | 1 |
| cl-8 | relative cluster sizes | 4/7 |
| cl-9 | cluster reach | 2/7 |

| geometric features | | |
|---|---|---|
| geo-1 | area of the enclosing rectangle | 1 |
| geo-2 | area of the convex hull | 1 |
| geo-3 | ratio of points on the convex hull | 1 |
| geo-4 | convex hull edge lengths | 1/7 |
| geo-5 | min distance from non-hull nodes to the hull | 7 |

| Minimum Spanning Tree (MST) features | | |
|---|---|---|
| mst-1 | sum of MST edge distance | 1 |
| mst-2 | distances of MST edges | 2/7 |
| mst-3 | node degrees of MST | 1/7 |
| mst-4 | MST depth from the depot | 7 |

| Nearest Neighbor features | | |
|---|---|---|
| nn-1 | distance to 1st nearest neighbor | 2/7 |
| nn-2 | angle between 2 nearest neighbors | 7 |
| nn-3 | cosine similiarity between 2 nearest neighbors | 7 |
| nn-4 | node degrees in 2NN graph | 1/7 |
| nn-5, 6 | number of 2NN graph components (abs., rel.) | 1/2 |
| nn-7 | size of 2NN components | 5/7 |
| nn-8 | ratio of max. component demand to vehicle capacity | 1 |
| nn-9 | reach of components in 2NN graph | 1/7 |

| Computational effort | | |
|---|---|---|
| cpu | CPU time for computing all features | 1 |

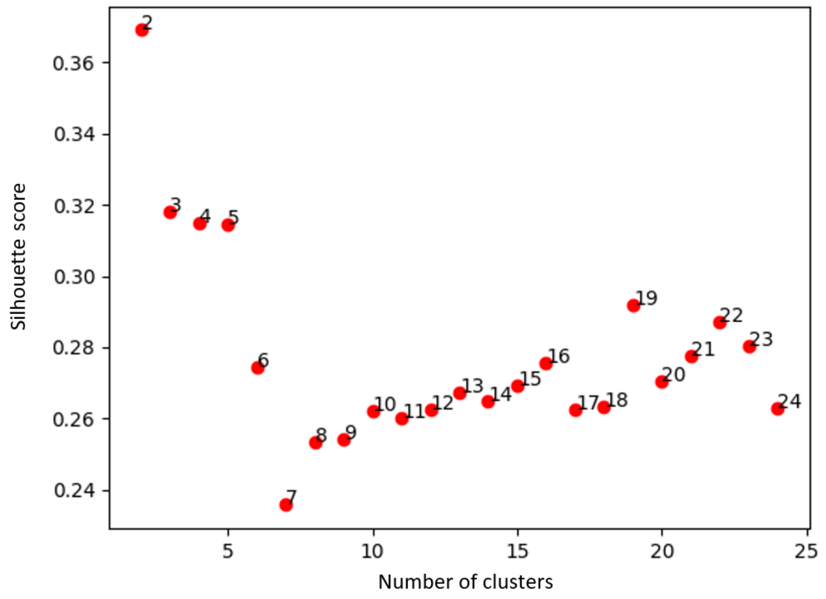
***Second iteration and final model***

In the second step of this proposed approach the training features have to be clustered and group together again, now in regard to the new and reduced feature vector. As opposed to the initial method of using a default value of 4 clusters, at this stage the choice of the number of clusters is more sophisticated. The decision is made based on 3 factors. First, the number of clusters should not be too small to allow for instance specific configurations. As stated previously the basic idea is that similar instances behave similarly with the same configuration, therefore the goal is to have clusters with relatively homogenous instances while still maintaining generality of the configurations, meaning that they also perform well on any new instances that are assigned to that cluster. Second, it should be ensured that the training instances are split into clusters of relatively even cluster sizes. And third, the silhouette score of the resulting clusters is considered, as can be seen in figure 15. Introduced by Rousseeuw (1987), the silhouette score measures how similar all instances are to their respective cluster compared to the instances of all other clusters. This is done by looking at each instance individually and computing the average distance to all other instances within the same cluster. Next, for each different cluster from the one the instance is assigned to, the average distance to the points in that cluster is calculated as well. The silhouette score of an instance now looks at the ratio between the calculated value of its own cluster and the minimum value of all other clusters. The score can range between -1 and +1, with a score close to -1 saying that the best alternative cluster is much closer than the one the instance is assigned to, and a score close to +1 saying that its own cluster is much closer than the best alternative. The overall silhouette score of the entire clustering is then simply calculated as the average over all instances, with a value of +1 therefore still as the aspired value. (Rousseeuw, 1987)

Based on these 3 criteria a number of 5 clusters was selected. The option of 2 clusters was discarded due to the first criteria, based on the silhouette scores the options of 6 or more clusters were discarded as there is a noticeably drop off in the scores, and out of the options for 3, 4 or 5 clusters, 5 was chosen as is it provided the highest number of clusters between these options while also providing the most even distribution of instances among these clusters. The 200 training instances were therefore clustered into 5 groups of 96, 36, 38, 6 and 24 instances, respectively.

**Figure 15**

*Silhouette Scores of Clusters of Training Instances Formed with Reduced Feature Set*



Just as in the first iteration, all 4 metaheuristics are now configured for each of the 5 clusters using the SMAC algorithm, again configured to perform 500 algorithm runs per configuration, now providing the reduced version of the preprocessed dataset containing only the 24 selected features as an instance file. All the training instances are then solved by each metaheuristic, configured with the determined configuration of the respective instance's cluster. Once again, the best performing metaheuristic is set as the instance's class label and with this labeled training data at hand, a new random forest classifier is trained to complete the final model.

## 5.5.Flow for Unseen Instances

This section will briefly describe the process of solving a previously unseen problem instance with the final model described above. This process contains 3 separate subprocesses, the reading of the instance file, the overhead, which is the algorithm selection and configuration model proposed in this thesis, and lastly the solving of the problem instance itself. The overhead starts by calculation the instance features. For the final live flow, the feature extraction algorithm has been adapted to calculate only the specific features that were selected during the feature selection step, which not only aims to improve the predictive performance of the model but also reduces the computational effort of computing all the instance features. Once the feature vector is computed, it is normalized according to the same distribution of the training instances used in the model. This is necessary in order to allow for comparison of the feature vectors. With the normalized feature vector at hand, it can now be used to load the classifier that was trained when building the model and to assign the instance into one class and therefore to one metaheuristic. Likewise, the clusterer that was built in the model creation phase is loaded and the algorithm's configuration is determined by assigning the instance to one cluster. With this overhead process completed, the problem instance can be solved with the selected metaheuristic and the selected configuration.

As mentioned before, one of the aims of this thesis is to test, whether this approach can yield beneficial results in a time-constrained environment. Therefore, a 30 second time limit is enforced as the termination criterion for the solving of the problem instance. Of course, the runtime of the overhead has to be controlled in that regard as well. First, by assuming time as a constrained resource, a long overhead runtime would greatly dampen the value of the model, and second, if the overhead runtime is much longer than the solver runtime itself, it could in fact be easier and more practical to simply run the solver a couple of times with different algorithms and configurations and take the best result provided. Therefore, it is defined that the overhead, consisting of feature extraction, normalization, classification and clustering, must not take longer than the solver and a hard 30 second time limit is imposed on the overhead as well.

## 5.6.Instance Set and Experimental Setup

The VRP problem instances used for the computations were drawn from several sets proposed in the scientific literature, collected in the "Capacitated Vehicle Routing Problem Library" (CVRPLIB). (Xavier, 2014) The CVRPLIB includes a total 267 problem instances from different sources from the literature. Instances, which either used different distance formats or contained exceedingly large numbers of customers, were removed from the set. The resulting set of problem instances that is used in this approach therefore contains 250 instances with a range of customers from 15 to 1040. Table 2 shows a detailed list of the sources for these instances, the number of instances drawn from these sources as well as the ranges of customers used in them.

**Table 2**

*List of Problem Instances*

| | **# of instances** | **Min. # customers** | **mean # customers** | **max. # customers** |
|---|---|---|---|---|
| Set A (Augerat, 1995) | 27 | 31 | 48 | 79 |
| Set B (Augerat, 1995) | 23 | 30 | 50 | 77 |
| Set E (Christofides & Eilon, 1969) | 11 | 21 | 59 | 100 |
| Set F (Fischer, 1994) | 3 | 44 | 83 | 134 |
| Set M (Christofides et al., 1979) | 5 | 100 | 154 | 199 |
| Set P (Augerat 1995) | 24 | 15 | 47 | 100 |
| CMT (Christofides et al., 1979) | 14 | 50 | 113 | 199 |
| (Rochat & Taillard, 1995) | 13 | 75 | 130 | 385 |
| (Golden et al., 1998) | 20 | 200 | 347 | 483 |
| (Li et al., 2005) | 10 | 560 | 780 | 1040 |
| Set X (Uchoa et al., 2014) | 100 | 100 | 412 | 1000 |
| **Total** | **250** | **15** | **258** | **1040** |

The instance set was then split into a training set containing 200 instances and a test set containing 50 instances by selecting 20% of the instances from every set listed in table 2 for the test set.

All computations were performed on a machine equipped with an Intel Core i7-10510U CPU and 16GB of RAM.

# 6. Computational Results – Analysis

## 6.1. Solution Quality

A common approach in analyzing the solution quality of an algorithm selector is looking at the gap between the virtual best solver (VBS) and the single best solver (SBS), in short, the SBS-VBS gap. The virtual best solver is defined as the perfect per-instance algorithm selector, which always selects the best algorithm available. The single best solver is the single algorithm that achieves the best performance across the entire instance set and is therefore the solution of the per-set algorithm selection problem. While in theory the performance achieved by using an algorithm selector could be worse than the performance of the single best solver, the VBS and the SBS performance should serve as lower and upper bounds in the analysis of any reasonable algorithm selector. The SBS-VBS gap then gives an indication of the potential performance gains that could be achieved by using an algorithm selector and the fraction of this gap that can be closed serves as a representative performance measure for the selector. (Bischl et al., 2016; Kerschke et al., 2019)

Even though the concept of the SBS-VBS gap is only used for evaluating the performance of an algorithm selector, it can be easily adapted and used to analyze the approach presented in this thesis as well. In order the get the SBS, each test instance is solved using every metaheuristic with a default configuration and the metaheuristic achieving the best average performance across all test instances is then chosen as the SBS. The default configurations are given by the solution of the per-set algorithm configuration problem on the training set, which is solved for every metaheuristic using SMAC and 500 algorithm runs. That means that every metaheuristic is configured in a way that aims to minimize the mean solution quality over all training instances. In this case the best performing metaheuristic on the test set is the genetic algorithm.

As the VBS is defined as the best solver for each instance, every test instance is solved with every metaheuristic and the configuration of every cluster in order to find the best solution. As there are 4 different metaheuristics and 5 clusters in the final model, there are 20 solutions to each test instance, of which the best one is chosen in order to get the solution quality of the VBS.

In order to create additional benchmarks allowing for a better comparison of the final performance of the proposed approach, automated algorithm selection (AAS) and automated algorithm configuration (AAC) approaches were also used individually as opposed to the proposed integrated approach. For the “AAS only” benchmark all 4 metaheuristics were first configured with the same default configurations that were used in determining the SBS, as described above. Every training instance is then solved by every metaheuristic with the metaheuristic producing the best solution once again serving as the class label for a given instance. For collecting the "AAS only" benchmark results every test instance is assigned to one class and is then solved with the respective metaheuristic using its default configuration.

For the "AAC only" benchmark performance, first the algorithm producing the best performance on the test set using the default configurations described above was selected, which was the genetic algorithm in this case. In the next step, the instance clusters and the optimized cluster configurations from the proposed approach were used to assign each test instance to a cluster and configure it with the respective configuration.

Lastly, the "AAS first, AAC second" benchmark is collected by running both AAS and AAC sequentially. For every test instance a metaheuristic is first chosen by applying AAS using default configurations, before assigning the instance to a cluster setting the parameters of the chosen algorithm according to the cluster configuration.

As described in the previous section the proposed approach is able to produce first results using the entire set of instance features and only using principal components analysis to project the features into a lower dimension. The results collected at this point are listed under the name "base model" in the analysis. After feature selection is performed and the rest of the process is completed as described in the previous section, the results of the "final model" are collected.

**Table 3**

*Analysis of the Solution Quality*

| | **avg. delta to opt** | **SBS-VBS gap closed** |
|---|---|---|
| **SBS** | 13.21% | |
| **VBS** | 4.23% | |
| **AAS Only** | 9.91% | 36.72% |
| **AAC Only** | 8.67% | 50.55% |
| **AAS first, AAC second** | 9.61% | 40.09% |
| **Base model** | 6.45% | 75.30% |
| **Final model** | 6.28% | 77.16% |

As mentioned in the beginning of this section, the aim of these different approaches is to close as much of the SBS-VBS gap as possible. The percentage of the gap that could be closed is shown in the right column of table 3. This gap and the resulting percentage are calculated with the average deltas to the optimum over the entire test set. It has to be noted however, that a proven optimum has not been found for all instances in the test set so far, therefore, in the cases where no optimum has been found, the best-known solution is taken as the optimum.

What is interesting in the middle part of the results shown in table 3 is first, that the performance of using AAC only is better than the performance of using only AAS, which underscores the point made in previous sections that the configuration of a given algorithm can be at least as important if not even more important than the selection of the algorithm itself. Of course, this outcome is highly dependent on the specific metaheuristics and their implementations, so while using AAC only outperforms AAS only in this case, using a portfolio of state-of-the-art solver might show a different picture. However, while using AAS and AAC sequentially improves the performance over using AAS alone, it cannot close the gap to using only AAC. This fact could imply that the performance of an algorithm with a default configuration is not necessarily a good indication on its performance with a per-cluster optimized configuration, resulting in a bad algorithm prediction quality of the entire "AAS first, AAC second" system.

The two bottom lines in table 3 show that the integrated approach proposed in this thesis clearly outperforms the single best solver as well as the three other benchmarks of using AAS and AAC either individually or sequentially and is able to close the SBS-VBS gap by more than 75% with the base model and with the final model even by more than 77%. This clearly states, that given the already underlined importance of an algorithm's configuration, using the performance of an algorithm using the configuration optimized for an instance's cluster instead of the default configuration when collecting performance data and thus the class label, yields a significant increase in final algorithm prediction quality and therefore resulting performance.

However, it has to be noted that, while there is a performance increase from the intermediate results of using the entire features set with principal components analysis to the final results of using the reduced feature set after feature selection, the performance increase is not significant. Future research could explore whether this performance gain could be increased by experimenting with alternative feature selection methods.

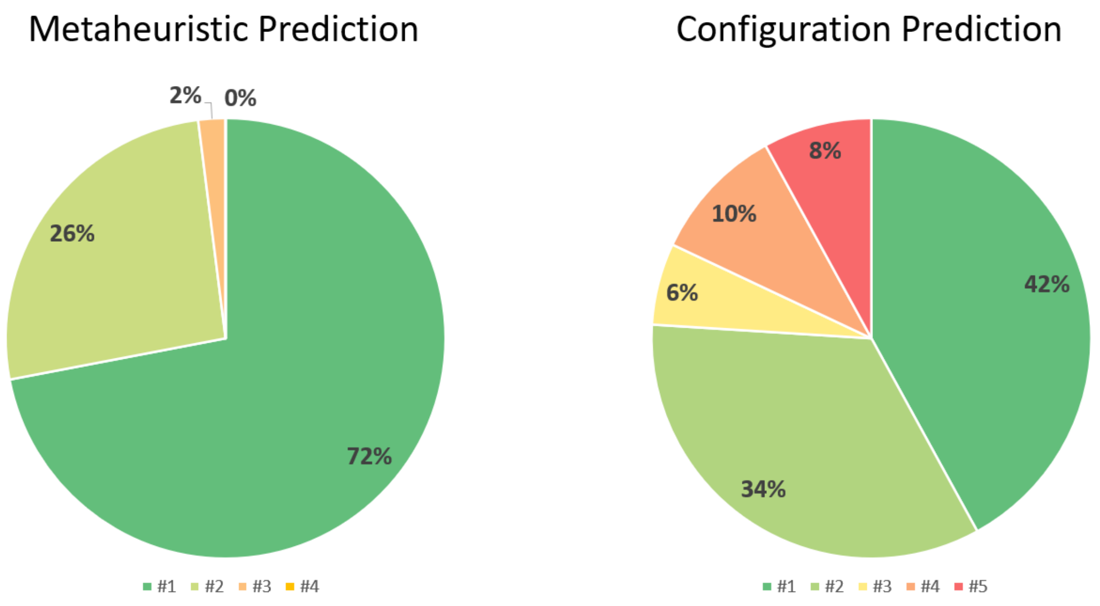
## 6.2.Prediction Quality

As stated in the previous section, the performance gain of the proposed approach over the "AAS first, AAC second" approach can be attributed to the increased metaheuristic prediction quality of the entire system by using the per-cluster optimized configuration when collecting performance data and class labels. This section analyzes the actual performance of the classification and clustering algorithm used in determining the algorithm and its configuration.

In order to determine the metaheuristic prediction quality every instance was solved with every metaheuristic, with the respective metaheuristic using the configuration optimized for the cluster the instance is assigned to. In measuring the configuration prediction, on the other side, the metaheuristic chosen by the classification algorithm is taken and this metaheuristic is used to solve the instance using the configurations of each cluster. This provides 4 values for measuring the metaheuristic prediction, one for each metaheuristic, and 5 values for measuring the configuration prediction, one for each cluster. These values are ranked before they are compared to the values chosen by the proposed approach. The results are shown in figure 16.

**Figure 16**

*Analysis of Predictive Performance*



The best metaheuristic out of the 4 possible options is chosen in 72% of the cases, which equals 36 out of the 50 test instances. In 26% of the instances the second-best metaheuristic is chosen, which means that there is only a single instance in the test set were only the third best option was selected and in fact not a single case where the worst one was chosen. As mentioned before, this good metaheuristic prediction quality is the key driver in the good performance of the proposed method and that more than 77% of the SBS-VBS gap could be closed.

At first glance the configuration prediction quality does not look as good as the metaheuristic prediction quality, with the best available configuration out of the 5 options for a given instance and chosen metaheuristic selected in 42% of the cases, the second best configuration being selected in 34% of the instances and even the third, fourth and fifth options being selected in between 6% and 10% of the cases respectively. However, on the one side the fact that either the best or second best configuration is chosen in 76% of the cases is an acceptable result, and on the other side, when comparing the selected configurations with the default configurations, which are obtained by the per-set algorithm configuration of the training set, the chosen configuration outperforms the default in 86% of the cases. That means that in most cases where the best possible configuration was missed, performance gains over a default scenario, like only using automated algorithm selection with default configuration, could still be realized.

## 6.3.Runtime Analysis

With 2 of the 3 research questions, namely the possibility of predicting the best metaheuristic and configuration for an instance as well as the performance gains that can be realized with the integrated approach presented in this thesis, answered, this section will focus on the third questions of whether it is possible to achieve the two points above within a reasonable time frame.

As defined in the problem formulation, the time limit for the solver to actually solve the instance was set to 30 seconds. The term reasonable time frame was defined in a way that the overhead, consisting of calculating instance features, normalizing them and selecting a metaheuristic as well as a configuration, cannot take longer than the solver itself, meaning the overhead runtime has to be smaller or equal to 30 seconds on each instance. As mentioned before, this is necessary because if the overhead would take longer than the solver itself, one could simply run the solver multiple times and take the best results, practically rendering the presented approach irrelevant.

However, table 4 shows that this time constraint could easily be satisfied across all 50 test instances. The average overhead computation time over all instances is only around 4 seconds, which is only 13.3% of the allowed time. Even the maximum overhead computation time, which comes from an instance containing 960 customers is less than half of the allowed time.

**Table 4**

*Runtime Analysis*

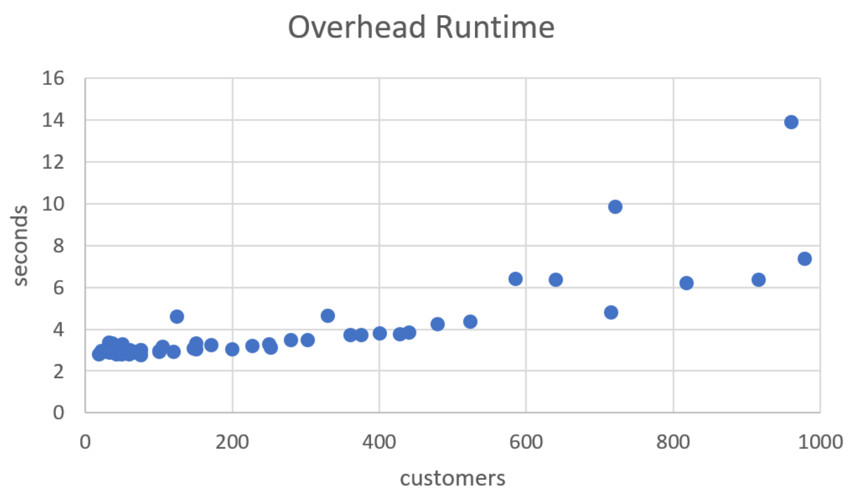
| | **Minimum** | **Mean** | **Maximum** |
|---|---|---|---|
| Overhead | 2.78s | 3.99s | 13.91s |
| Solver | 28.51s | 29.26s | 30.14s |
| **Total** | **31.29s** | **33.25s** | **44.05s** |

On the one hand these results prove that an integrated approach of combining instance specific algorithm selection and instance specific algorithm configuration can definitely perform well within a very reasonable time frame, however on the other hand, these results could also open up the possibility to extend the feature set with additional features in order to try to further increase the prediction quality. Other approaches from the literature use branch and cut probing and local search probing features, where some iterations of these algorithms are run on the instance and the results are recorded. A decision was made against including these features in this approach, as for larger instances of up to 1000 customers the feature extraction algorithm would likely exceed the 30 second time limit. Therefore, a decision would have to be made for either relaxing the time constraint, which was not an option given the research direction set for this thesis or imposing a hard cutoff at 30 seconds and if necessary, returning incomplete values for these instances. As this might reduce the explanatory power of these values, this option was discarded as well. Future research however could experiment with these options and thereby try to increase performance.

Figure 17 shows the overhead runtime in dependence of the number of customers in that respective problem instance. If the two highest points lying at around 10 seconds and 14 seconds respectively were to be considered outliers, one could argue that the runtime follows almost a linear development in respect to the number of customers. But even if these 2 points are included in the observation the runtime development still follows only a relatively moderate incline.

**Figure 17**

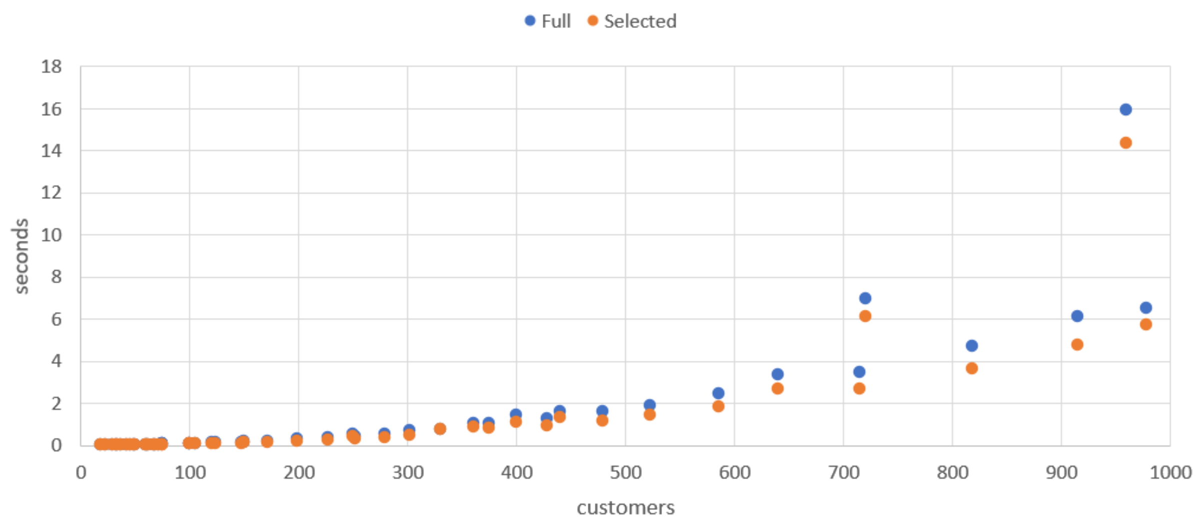
*Overhead Runtime in Seconds by Number of Customers*



It is also worth noting that out of the 4 steps included in the overhead, namely feature extraction, feature normalization, classification and clustering, only the feature extraction step has a variable runtime and is dependent on the size of the problem instance. Once the features are extracted, the following steps work with a feature vector that always has the same dimensionality regardless of the size of the problem instance, resulting in constant runtime.
Another aspect to analyze in regard to the runtime are the time savings that could be realized by performing feature selection. As mentioned in previous sections, increasing the predictive performance of the model was not the only reason to perform feature selection, but the fact that a reduced set of instance features also requires less computational effort was a very welcome side effect. While only 24 out of the 148 initial instance features were selected, it is obvious that the reduction of runtime would not happen to the same degree. This is due to the fact that at least one feature out of all 6 major groups of features was selected, therefore the basic, time-consuming tasks, like clustering the nodes, constructing the MST, the 2-nearest-neighbor-graph or the convex hull, still had the performed, only the calculations performed on these constructs were reduced. As a result, an average runtime saving of 27.5% could be achieved over the 50 test instances, as can be seen in figure 18. While there are some outliers in both directions, the percentage of time savings remains relatively constant across all instances, regardless of the number of customers in the problem instance.

**Figure 18**

*Runtime of Feature Computations for Full and Selected Feature Set by Number of Customers*



# 7. Conclusion

Every year new solvers for many optimization problems are proposed within the scientific literature, with many of them achieving state-of-the-art results. However, for most types of problems, there exists no single solver outperforming all others on all instances, which presents the user with a wealth of solver options to choose from. The choice of an algorithm from this portfolio as well as an ideal setting of its parameters, which also requires a lot of knowledge regarding the algorithm, the problem type and the problem instance, are very hard and time-consuming tasks. (Stützle & López-Ibáñez, 2019)

Therefore, this thesis presented the automated algorithm selection problem as well as the automated algorithm configuration problem and described several approaches within those domains. However, the scope of this thesis went beyond that. First, the goal was to find a set of representative instance features describing a vehicle routing problem instance, which can be used in the two problems mentioned above. Second, an integrated approach was implemented that not only runs automated algorithm selection and automated algorithm configuration sequentially but combines them into one approach. Lastly, with a real-world application in mind, the goal was to achieve those two points above within a reasonable time frame.

The results of the first goal were measured by the prediction quality of the model, meaning the percentage in which the best, second best, and so forth, option was selected, both from the portfolio of algorithms as well as from the portfolio of algorithm configurations. While the algorithm prediction turned out to be very successful, the prediction of the configurations was not just as good, even though in most cases the performance could still be improved over the use of a default configuration. For measuring the performance of the proposed approach, various baselines were introduced. The performance was compared to the single best solver, which is the single best metaheuristic with its single best configuration over the test set, as well as to using only automated algorithm selection with default configurations, only automated algorithm configuration with the single best performing metaheuristic and even running automated algorithm selection and automated algorithm configuration in sequence. The proposed integrated approach was in fact able to significantly outperform all those baseline scenarios, which suggests that algorithm selection and algorithm configuration should in fact be regarded together, as the outcome of the algorithm configuration problem changes the labels, which represent the best option for each instance, used in the algorithm selection problem. Lastly, the term "within a reasonable time frame" was defined so that the overhead consisting of calculating and normalizing the features, performing algorithm selection and algorithm configuration must not take longer than the solver itself. By carefully selecting the features in the feature set, the average overhead runtime could be kept at just over 13% of the allowed runtime, with even the maximum runtime, coming from an instance with 960 customers, recording a runtime of just 46% of the allowed runtime. This clearly shows that reasonable prediction qualities and significant performance gains can be achieved even with a very small overhead runtime.

While the implementation, the computations and the analysis conducted in this thesis have shown the promise of this approach and its ability to further improve average solution qualities by combining automated algorithm selection and automated algorithm configuration into one approach, there are some parts of the model that were intentionally kept quite simple in order to allow for an easier first evaluation of the model and its potential. Future research could focus on expanding on these simplified parts with the aim of further testing this approach as well as potentially improving its performance.

The most important simplification was the fact that the metaheuristics were self-implemented, instead of using state-of-the-art solvers from the literature. This decision was made, as the metaheuristics themselves were not supposed to be at the forefront of the evaluation, as they were only used to test the model, and by keeping their implementation simplistic it was easier to keep them competitive among each other, meaning that none of them outperforms the others on a majority of instances. Future research could therefore test the model on a portfolio of state-of-the-art solvers to see if it can yield the same performance gains.

A second focus for future research could be on the inclusion of local search probing and branch and cut probing features in the feature set, as proposed for example by Pihera and Musliu (2014). As described in the previous section, these features were omitted from the feature set used in this approach, as the runtime optimization was one of the main focus areas and, as these features would have required significantly larger amounts of computational effort, a decision would have had to be made, whether the runtime constraint would be relaxed or whether a hard calculation cutoff would be enforced once the runtime limit is reached, potentially reducing the explanatory power of these incomplete feature values. However, as the actual runtimes stayed well below the predefined runtime limit, it might be worth testing if including them could improve the predictive performance of the model.

Lastly, some of the applied machine learning algorithms were rather simple, so substituting them with more sophisticated approaches could possibly further improve the model’s performance. In the feature selection process, a simple filtering approach based on the ANOVA f-value for single feature evaluation was employed, this could be improved by using for example a wrapper method for feature subset evaluation. (Rasku et al., 2019) In order to improve the configuration prediction quality of the model, it could be worth looking at different clustering techniques instead of the k-Means clustering applied in this approach. While other techniques were briefly tested, this was not done in detail and a more thorough analysis could reveal more suitable techniques. Lastly, the basic concept of ISAC by Kadioglu et al. (2010) was used for instance specific algorithm configuration. While the basic approach is still regarded as state-of-the-art, newer versions, like ISAC++ by Ansótegui et al. (2016) for example, have been proposed over the years. Experimenting with these or alternatively the newest approach by Ansótegui et al. (2021) might have the potential to improve the model's performance as well.

However, as the techniques and approaches employed in the model presented in this thesis already managed to clearly outline the advantage of using a combined model of automated algorithm selection and automated algorithm configuration over using its parts individually, these possible extensions are left for future research.

## References

Ansótegui, C., Sellmann, M., & Tierney, K. (2009, September). A gender-based genetic algorithm for the automatic configuration of algorithms. In *International Conference on Principles and Practice of Constraint Programming* (pp. 142-157). Springer, Berlin, Heidelberg.

Ansótegui, C., Gabas, J., Malitsky, Y., & Sellmann, M. (2016). MaxSAT by improved instance-specific algorithm configuration. *Artificial Intelligence*, *235*, 26-39.

Ansótegui, C., Pon, J., & Sellmann, M. (2021). Boosting evolutionary algorithm configuration. *Annals of Mathematics and Artificial Intelligence*, 1-20.

Augerat, P. (1995). *Approche polyèdrale du problème de tournées de véhicules* (Doctoral dissertation, Institut National Polytechnique de Grenoble-INPG).

Bai, R., Chen, X., Chen, Z. L., Cui, T., Gong, S., He, W., ... & Zhang, H. (2021). Analytics and Machine Learning in Vehicle Routing Research. *arXiv preprint arXiv:2102.10012*.

Balaprakash, P., Birattari, M., & Stützle, T. (2007, October). Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In *International workshop on hybrid metaheuristics* (pp. 108-122). Springer, Berlin, Heidelberg.

Barbosa, E. B. D. M., & Senne, E. L. F. (2017). Improving the fine-tuning of metaheuristics: an approach combining design of experiments and racing algorithms. *Journal of Optimization*, *2017*.

Bartz-Beielstein, T., Lasarczyk, C. W., & Preuß, M. (2005, September). Sequential parameter optimization. In *2005 IEEE congress on evolutionary computation* (Vol. 1, pp. 773-780). IEEE.

Baxter, J. (1981). Local optima avoidance in depot location. *Journal of the Operational Research Society*, *32*(9), 815-819.

Berthold, M. R., Borgelt, C., Höppner, F., Klawonn, F., & Silipo, R. (2020). *Guide to intelligent data science: how to intelligently make sense of real data*. Springer International Publishing.

Birattari, M., & Kacprzyk, J. (2009). *Tuning metaheuristics: a machine learning perspective* (Vol. 197). Berlin: Springer.

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., ... & Vanschoren, J. (2016). Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, *237*, 41-58.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, *35*(3), 268-308.

Bonaccorso, G. (2017). *Machine learning algorithms*. Packt Publishing Ltd.

Christofides, N., & Eilon, S. (1969). An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society*, *20*(3), 309-318.

Christofides, N., Mingozzi, A., & Toth, P. (1979). The vehicle routing problem. In: Christofides, N., Mingozzi, A., & Toth, P. (eds), *Combinatorial Optimization*, pages 315–338. Wiley, Chichester.

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.

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

Dorigo, M., & Di Caro, G. (1999, July). Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)* (Vol. 2, pp. 1470-1477). IEEE.

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., Horvath, M., Kowalczyk, W., & Schut, M. C. (2006). Reinforcement learning for online control of evolutionary algorithms. In *International Workshop on Engineering Self-Organising Applications* (pp. 151-160). Springer, Berlin, Heidelberg.

Fisher, M. L. (1994). Optimal solution of vehicle routing problems using minimum k-trees. *Operations research*, *42*(4), 626-642.

Gendreau, M., & Potvin, J. Y. (Eds.). (2010). *Handbook of metaheuristics* (Vol. 2, p. 9). New York: Springer.

Gillett, B. E., & Miller, L. R. (1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations research, 22*(2), 340-349.

Golden, B. L., Wasil, E. A., Kelly, J. P., & 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* (pp. 33-56). Springer, Boston, MA.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on systems, man, and cybernetics*, *16*(1), 122-128.

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. (2003). An introduction to variable and feature selection. *Journal of machine learning research*, *3*(Mar), 1157-1182.

Guyon, I., & Elisseeff, A. (2006). An introduction to feature extraction. In *Feature extraction* (pp. 1-25). Springer, Berlin, Heidelberg.

Hoos, H., Lindauer, M., & Schaub, T. (2014). claspfolio 2: Advances in algorithm selection for answer set programming. *Theory and Practice of Logic Programming*, *14*(4-5), 569-585.

Huang, C., Li, Y., & Yao, X. (2019). A survey of automatic parameter tuning methods for metaheuristics. *IEEE transactions on evolutionary computation*, *24*(2), 201-216.

Hutter, F., & Hamadi, Y. (2005). Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. *Microsoft Research, Tech. Rep. MSR-TR-2005-125*.

Hutter, F., Hamadi, Y., Hoos, H. H., & Leyton-Brown, K. (2006, September). Performance prediction and automated tuning of randomized and parametric algorithms. In *International Conference on Principles and Practice of Constraint Programming* (pp. 213-228). Springer, Berlin, Heidelberg.

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. H., & Leyton-Brown, K. (2011, January). Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization* (pp. 507-523). Springer, Berlin, Heidelberg.

Hutter, F., & Ramage, S. (2015). Manual for SMAC version v2. 10.03-master. *Available online at: http://www. cs. ubc. ca/labs/beta/Projects/SMAC/v2*, *10*.

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

Jolliffe, I. T., & Cadima, J. (2016). Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *374*(2065), 20150202.

Kadioglu, S., Malitsky, Y., Sellmann, M., & Tierney, K. (2010). ISAC–instance-specific algorithm configuration. In *ECAI 2010* (pp. 751-756). IOS Press.

Kanda, J., Carvalho, A., Hruschka, E., & Soares, C. (2011). Selection of algorithms to solve traveling salesman problems using meta-learning 1. *International Journal of Hybrid Intelligent Systems*, *8*(3), 117-128.

Kerschke, P., Kotthoff, L., Bossek, J., Hoos, H. H., & Trautmann, H. (2018). Leveraging TSP solver complementarity through machine learning. *Evolutionary computation*, *26*(4), 597-620.

Kerschke, P., Hoos, H. H., Neumann, F., & Trautmann, H. (2019). Automated algorithm selection: Survey and perspectives. *Evolutionary computation*, *27*(1), 3-45.

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

Laporte, G., & Semet, F. (2002). Classical heuristics for the capacitated VRP. In *The vehicle routing problem* (pp. 109-128). Society for Industrial and Applied Mathematics.

Li, F., Golden, B., & Wasil, E. (2005). Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research*, *32*(5), 1165-1179.

Lindauer, M., Hoos, H., & Hutter, F. (2015a). From sequential algorithm selection to parallel portfolio selection. In *International Conference on Learning and Intelligent Optimization* (pp. 1-16). Springer, Cham.

Lindauer, M., Hoos, H. H., Hutter, F., & Schaub, T. (2015b). Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, *53*, 745-778.

Lindauer, M., van Rijn, J. N., & Kotthoff, L. (2019). The algorithm selection competitions 2015 and 2017. *Artificial Intelligence*, *272*, 86-100.

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* (pp. 320-353). Springer, Boston, MA.

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

Malitsky, Y., Mehta, D., & O'Sullivan, B. (2014). Evolving instance-specific algorithm configuration. In *Instance-Specific Algorithm Configuration* (pp. 93-105). Springer, Cham.

Mercer, R. E., & Sampson, J. R. (1978). Adaptive search using a reproductive meta-plan. *Kybernetes*.

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.

Mohammed, M., Khan, M. B., & Bashier, E. B. M. (2016). *Machine learning: algorithms and applications*. Crc Press.

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). *Foundations of machine learning*. MIT press.

Nannen, V., & Eiben, A. E. (2007, September). Efficient relevance estimation and value calibration of evolutionary algorithm parameters. In *2007 IEEE congress on evolutionary computation* (pp. 103-110). IEEE.

Pihera, J., & Musliu, N. (2014, November). Application of machine learning to algorithm selection for TSP. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence* (pp. 47-54). IEEE.

Rasku, J., Kärkkäinen, T., & Musliu, N. (2016). Feature extractors for describing vehicle routing problem instances. *OASICS; 50*.

Rasku, J., Musliu, N., & Kärkkäinen, T. (2019). Feature and algorithm selection for capacitated vehicle routing problems. *Toward Automatic Customization of Vehicle Routing Systems*, 129.

Rice, J. R. (1976). The algorithm selection problem. In *Advances in computers* (Vol. 15, pp. 65-118). Elsevier.

Rochat, Y., & Taillard, É. D. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of heuristics*, *1*(1), 147-167.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, *20*, 53-65.

Scheffe, H. (1999). *The analysis of variance* (Vol. 72). John Wiley & Sons.

Smit, S. K., & Eiben, A. E. (2009, May). Comparing parameter tuning methods for evolutionary algorithms. In *2009 IEEE congress on evolutionary computation* (pp. 399-406). IEEE.

Smith-Miles, K., van Hemert, J., & Lim, X. Y. (2010, January). Understanding TSP difficulty by learning from evolved instances. In *International Conference on Learning and Intelligent Optimization* (pp. 266-280). Springer, Berlin, Heidelberg.

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.

Steinhaus, M. (2015). *The application of the self organizing map to the vehicle routing problem*. University of Rhode Island.

Stützle, T., & López-Ibáñez, M. (2019). Automated design of metaheuristic algorithms. In *Handbook of metaheuristics* (pp. 541-579). Springer, Cham.

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.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and computing*, *4*(2), 65-85.

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

Xavier, I. (2014). *CVRPLIB: Capcitated Vehicle Routing Problem Library*. Retrieved July 14, 2021, from: http://vrp.atd-lab.inf.puc-rio.br/index.php/en/

Xu, L., Hoos, H., & Leyton-Brown, K. (2010, July). Hydra: Automatically configuring algorithms for portfolio-based selection. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2012, June). Evaluating component solver contributions to portfolio-based algorithm selectors. In *International Conference on Theory and Applications of Satisfiability Testing* (pp. 228-241). Springer, Berlin, Heidelberg.

Yuan, Z., De Oca, M. A. M., Birattari, M., & Stützle, T. (2012). Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. *Swarm Intelligence*, *6*(1), 49-75.

# Appendix

## Appendix A: Detailed List of the Metaheuristic's Parameters

**Table 5**

*Parameters of Iterated Local Search*

| Iterated Local Search | | | |
|---|---|---|---|
| **Name** | **Description** | **Type** | **Range** |
| const | construction heuristic | categorical | {random, nn, savings, cfrs} |
| neighborhood | neighborhood used for perturbation | categorical | {move, swap, two_opt} |
| num_steps | number of perturbation steps | integer | [1, 1000] |
| ls | local search heuristic | categorical | {move_first, move_best, swap_first, swap_best, two_opt_first, two_opt_best} |

**Table 6**

*Parameters of Simulated Annealing*

| SA | | | |
|---|---|---|---|
| **Name** | **Description** | **Type** | **Range** |
| const | construction heuristic | categorical | {random, nn, savings, cfrs} |
| neighborhood | neighborhood used for perturbation | categorical | {move, swap, two_opt} |
| temp | starting temperature | real | [1, 10000] |
| cool_rate | cooling rate | real | [0.001, 0.999] |

**Table 7**

*Parameters of the Genetic Algorithm*

| GA | | | |
|---|---|---|---|
| **Name** | **Description** | **Type** | **Range** |
| const | construction heuristic | categorical | {random, nn, savings, cfrs} |
| neighborhood | neighborhood used for perturbation | categorical | {move, swap, two_opt} |
| ls | local search heuristic | categorical | {move_first, move_best, swap_first, swap_best, two_opt_first, two_opt_best} |
| perform_ls_init | whether to perform local search on initial solutions | boolean | {true, false} |
| perform_ls | whether to perform local search on new solutions | boolean | {true, false} |
| p_size | population size | integer | [2, 100] |
| num_elite | number of elite indiviuals | integer | [0, 99] |
| select | selection operator | categorical | {roulette, binary, tournament} |
| cross | crossover operator | categorical | {one_point, partially_mapped} |
| mut_rate | mutation rate | real | [0, 1] |
| num_steps | number of perturbation steps | integer | [1, 1000] |

**Table 8**

*Parameters of the Ant Colony Optimization*

| ACO | | | |
|---|---|---|---|
| **Name** | **Description** | **Type** | **Range** |
| p_size | population size | integer | [2, 100] |
| ls | local search heuristic | categorical | {move_first, move_best, swap_first, swap_best, two_opt_first, two_opt_best} |
| perform_ls | whether to perform local search on solutions | boolean | {true, false} |
| init_pher | initial pheromone value | real | [0.001, 1000] |
| rho | evaporation rate | real | [0.001, 1] |
| alpha | potency of pheromone value | integer | [-20, 20] |
| beta | potency of distance value | integer | [-20, 20] |
| q | Importance of solution quality in pheromone update | integer | [0, 1000] |

## Appendix B: Detailed List of Feature Set after Feature Selection

- VRP-domain specific features:
  - *minimum* number of vehicles
- node distribution features:
  - *minimum* value in the distance matrix
- cluster features:
  - *absolute number* of clusters
  - relative cluster sizes (*skewness, kurtosis, coefficient of variation*)
  - cluster reach (*mean, skewness*)
- geometric features:
  - *ratio* of points on the convex hull
  - *coefficient of variation* of the convex hull edge lengths
- MST features:
  - distances of MST edges (*mean, minimum*)
  - *mean* node degrees of the MST
- nearest neighbor features:
  - distance to 1st nearest neighbor (*mean, minimum*)
  - *maximum* node degree in 2-nearest-neighbor graph
  - *absolute number* of 2-nearest-neighbor graph components
  - relative sizes of 2-nearest-neighbor graph components (*mean, minimum, maximum, skewness, kurtosis*)
  - *mean* reach of components in 2-nearest-neighbor graph