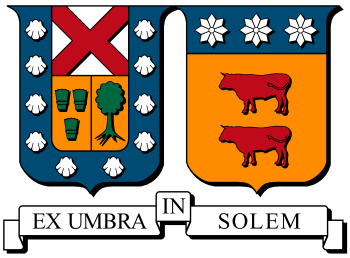
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO – CHILE



"ANALYSIS OF ROUTING PROBLEM INSTANCES FOR METAHEURISTIC PARAMETER PREDICTION USING MACHINE LEARNING"

TOMÁS WILLIAM BARROS EVERETT

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INFORMÁTICO

PROFESOR GUÍA: ELIZABETH MONTERO
PROFESOR CORREFERENTE: NICOLÁS ROJAS-MORALES

ENERO 2024

Acknowledgements

I thank my family for providing me with stillness, and my friends for providing me with joy. I also thank my tutor Elizabeth for guiding me during this work.

Resumen

El proceso de sintonización de parámetros de una metaheurística es un proceso computacionalmente costoso. En este trabajo de título se explora la aplicación de algoritmos de aprendizaje automático para la predicción de parámetros para metaheurísticas basada en características extraídas de las instancias de pruebas de un problema en particular a resolver. Al analizar los conjuntos de instancias de CVRPTW de la literatura, se estudia la relación entre sus características y sus configuraciones óptimas de parámetros con un enfoque basados en datos. A partir del estudio es posible concluir que la predicción de parámetros es más útil cuando hay una abundancia de instancias con características diversas con las cuales entrenar modelos predictivos.

Abstract

The process of tuning a metaheuristic's parameters is computationally costly. We explore the application of machine learning algorithms for optimal metaheuristic parameter prediction based on extracted instance features. Analyzing the literature's CVRPTW instance sets, we study the relationship between instance characteristics and optimal parameter configurations through data-driven approaches. We conclude that parameter prediction is most useful when there is an abundance of diverse instances with which to train predictive models.

Index

List of Tables

List of Figures

Introduction

Vehicle Routing Problems (VRPs) are about finding efficient delivery routes for a set of vehicles. The goal here is to minimize travel times while dealing with constraints, such as vehicle capacity and delivery time-windows. Metaheuristics are algorithms that approximate the optimal solution to hard problems such as VRPs. To solve an instance of a VRP using a metaheuristic algorithm, we must first define the parameters values with which the algorithm will run. These parameters define the behaviour of the metaheuristic, and can have an important effect on the quality of the solutions the algorithm finds. There is no a unique set of parameters values that work the best for all possible problem instances, and as such, we may attempt to determine the ideal set of parameters on a per-instance basis.

Machine learning can be applied to understand the characteristics of a problem instance in order to predict what parameters values will be the best for a metaheuristic to use. By developing a set of numerical features that are algorithmically extracted from each problem instance, and obtaining a set of quality parameters through the use of a tuning algorithm, we can learn the mapping of features to parameters with the use of machine learning models.

The objective of this work is to analyze a set of extracted features in order to train a machine learning model for quality parameter prediction.

Chapter 1

Problem Definition

In the following chapter we define the most relevant technical terms to understand the definition of the problem we intend to solve, that is, to apply machine learning techniques to features extracted from a combinatorial optimization problem's instances to identify quality parametrizations for an instance-solving metaheuristic.

1.1. Intractable Problems

A problem is considered intractable [4] when there is no **efficient** algorithm to solve reasonably sized problem instances. While computationally intractable problems are frequently thought of as problems belonging to the category of NP, this relationship is not strict. Problems belonging to P, meaning problems that can be solved by a deterministic algorithm in polynomial time [4] can become intractable if the coefficients of said polynomial are large enough. Similarly, a problem belonging to NP with time complexity $O(1.000001^n)$ can be solved in a reasonable amount of time for large values of n.

When the domain of a problem consists of discrete values, finding the globally optimal solution for a problem means finding the set of values that optimize a given metric. Thus, an intractable problem usually posses a search space that is both large and topologically

complex[1].

1.2. Metaheuristics

Combinatorial optimization is a field of mathematics that focuses on solving computationally intractable problems, such as finding the optimal route through a set of locations (Traveling Salesman Problem [6]) or finding a set of objects of maximal value subject to size or weight restrictions (Knapsack Problem [7]).

Given the intractable nature of these problems, optimal solutions cannot be found using exhaustive methods in a reasonable time frame. Thus, algorithms that search only a subset of the problem's search space have been developed [8]. These local search algorithms, or metaheuristics methods, may find a solution of sufficient quality in a more reasonable amount of time. Some examples include Hill-Climbing, Simulated Annealing and Genetic Algorithms. [9]

Some well known metaheuristics will be described in detail in Chapter 2

1.3. Metaheuristic Tuning

Depending on the implementation, metaheuristics require a set of parameters to regulate its behaviour. As an example, a few parameters of a Genetic Algorithm [9] are given:

- Population Size: The number of individuals that are exploring the search space during each iteration.
- Mutation Rate: The probability that an individual may undergo a mutation (modification) between iterations.
- Maximum Iterations: The number of iterations after which the algorithm stops.

[1] A problem with a large search space can be solved trivially if both its search space and evaluation function are convex [5]

The values set for a metaheuristic's parameters determine, in part, its capacity to solve instances. It is expected that a fixed set of parameter values will produce optimal results for only some layouts. The problem of finding the set of parameters that best solves a set of instances is known in the literature as *tuning*. [10]

Generally, parameter tuning runs into a few issues:

- Running a metaheuristic algorithm on all parameter configurations is enormously costly.
- Depending on the context of the problem and the metaheuristic used, the quality of solutions may be particularly sensitive to both configuration of parameters and the features of each problem instance being solved.
- Inter-parameter interactions tend to be complex, unpredictable and hard to quantify.
- It is difficult to identify the relationship between instance features and a quality parameter configuration, even after finding said configuration. Neither metaheuristics nor tuning algorithms tend to make use of the massive amount of data generated during their execution.

In Chapter 2, we describe in detail some well known tuning methods in literature.

1.4. Metaheuristic Tailoring

Aside from parameter tuning, metaheuristics performance depend significantly on the instance's underlying distribution. The No Free Lunch Theorem [11] asserts that no single algorithm can be expected to outperform all others for all problem instances. Thus, identifying which algorithms to apply to specific instances may prove useful to improve solution quality or reduce computation time.

Adjusting a metaheuristic to a specific problem is known as *tailoring* [1]. Manual metaheuristic tailoring is usually done by individuals with expert knowledge of the problem's context, and has been found to improve performance.

Manual tailoring does however face a few limitations:

- Running many versions of a metaheuristic on a set of instances and manually altering metaheuristic features is a time-consuming process.
- Manual tailoring is an intuitive process that depends on an expert's knowledge, making it error-prone.
- Manual tailoring may not be applied to scenarios where there are no human experts, or scenarios with rapidly changing features.

In Chapter 2 we describe in detail some manual tailoring approaches.

1.5. Automatic Metaheuristic Design

The field of automatic metaheuristic design proposes an alternative to manual tailoring [1]: Substituting expert judgements with algorithms that quantify the quality of design decisions in order to select the best option from a predetermined set of possibilities.

In Chapter 2, we present some papers describing automatic metaheuristic approaches in literature.

1.6. Machine Learning

Machine learning is a field of mathematics focused on the automatic construction of prediction and decision models from data. Generally, the field of machine learning can be separated into three distinct approaches [9]:

- Supervised Learning, where a model learns to predict an outcome, or *target variable* by learning the relationship between said outcome and a set of input data.

- Unsupervised Learning, where there is no predefined target variable and algorithms are utilized to exploratively extract information from a dataset.
- Reinforcement Learning, where an agent learns in a simulated environment that gives it feedback in order to guide future decisions.

In recent years, machine learning techniques have been utilized to assist the resolution of intractable problems with metaheuristics [12], leveraging the high volume of data that can be obtained from solved problem instances to approximate both algorithmic decisions and design decisions.

In Chapter 2 we describe some applications of machine learning approaches to the problem of automatic metaheuristic selection/generation.

1.7. Capacitated Vehicle Routing Problem with Time Windows

The Vehicle Routing Problem (VRP) is a classic combinatorial optimization problem. It involves finding an optimal route for a set of vehicles to fulfill the demands of a set of geographically distributed customers. The objective of the VRP is to minimize overall transportation costs, or in the simplest case, travel distance.

CVRPTW includes two additional restrictions: Vehicles possess a maximum carrying capacity (C) and clients may only be services during predefined time windows (TW).

The purpose of this project is to apply machine learning techniques to features extracted from CVRPTW instances in order to predict a quality parametrization for a metaheuristic that will best solve novel instances.

The details of our implementation will be expanded upon in Chapter 3

Chapter 2

State of the Art

We present an introductory summary of the state of the art with respect to the fields of metaheuristics, metaheuristic tuning, automatic metaheuristic design, and machine learning in general.

2.1. Metaheuristics

2.1.1. Hill-Climbing

Hill-Climbing attempts to find the best solution to a problem by iteratively modifying a known solution and selecting a nearby better option. [9]

From an initial solution, Hill-Climbing will generate a neighbourhood through the application of a movement[1] function. After evaluating every neighbour[2], the best among them will be selected as the new current solution, and will in turn be used to generate the next set of neighbouring solutions. The algorithm will terminate once every solution in the current solution's neighbourhood is of lower quality that the current solution, thus guaranteeing convergence to a local optima.

[1] Also known as a neighbourhood function
[2] In the case of Best Improvement Hill-Climbing

Algorithm 1 Hill-Climbing

```
1:  currentSolution ← initialSolution
2:  while True do
3:      neighbourhood ← movement(currentNode)
4:      bestEval ← INF
5:      bestSolution ← NULL
6:      for solution in neighbourhood do
7:          if EVAL(solution) > bestEval then
8:              bestSolution ← solution
9:              bestEval ← EVAL(solution)
10:     if bestEval ≤ EVAL(currentSolution) then
11:         return currentSolution
12:     currentNode ← bestSolution
```

While great at arriving at local optima, Hill-Climbing is incapable of further exploring the search space of a problem. This can be partially mended by wrapping the algorithm in a random-restart routine, though further modifications are required to truly increase the metaheuristic's exploratory capacity.

The general performance of Hill-Climbing was benchmarked against other metaheuristics in [13], and applications of the metaheuristic to solve chemical engineering problems can be found in [14]. Some modifications to the original Hill-Climbing algorithm include probabilistically moving to neighbouring solutions (Probabilistic Hill-Climbing [15]), moving strictly through one search-space dimension at a time (Coordinated Ascent [16]) and randomizing restart times and parameters (Random Restart Hill-Climbing [9]).

2.1.2. Simulated Annealing

Simulated Annealing, initially proposed by Kirkpatrick et al. [17], is a modification of First Improvement Hill-Climbing, where a solution of lesser quality may be selected with a probability proportional to the algorithm's current *temperature*. With a high temperature, the

algorithm will frequently accept worse solutions, which serves as a mechanism for search space exploration. As iterations pass, the temperature will asymptotically tend to zero, guaranteeing eventual convergence to a local optimum.

The name derives from the process of steel annealing, where a piece of steel is raised to a specific temperature and then is allowed to cool slowly. At the atomic level, taking a hard and brittle piece of steel and heating it above it's recrystallization temperature allows it's atoms to reorganize into their equillibrium state. By increasing the system's enthalphy, the piece of steel will manage to escape to a configuration of lower entropy. Similarly, simulated annealing can be seen as initially 'flattening' the search space, for high temperatures allow the algorithm to indiscriminately select lesser solutions. As temperature gradually lowers, the search space returns to it's real topology, where lesser solutions are ignored in favour of better performing neighbours, eventually converging to a stable optimum.

Simulated Annealing was applied to finding shortcuts between objects in [18], to plan large-scale European aircraft trajectories in [19], for example.

2.1.3. Tabu Search

Also a modification of Hill-Climbing, Tabu Search [20] constructs a neighbourhood but does not compare neighbours with the current solution. Instead, it simply chooses the best neighbour. To prevent falling into loops, a Tabu List is implemented, which stores recent solutions to prevent the algorithm from revisiting them.

Tabu Search has been widely applied to combinatorial optimization problems, such as scheduling, vehicle routing and expert systems. [21]

2.1.4. Evolutionary Algorithms

An Evolutionary Algorithm is a population-based metaheuristic inspired by Charles Darwin's Theory of Evolution. Solutions in the population are probabilistically subjected to mutation (slight alterations), crossover (mixing of two or more solutions) and selection, in

order to generate new solutions.

The main advantage of Evolutionary Algorithms over a Hill-Climbing derived metaheuristic is their exploratory capacity. For search spaces that possess many local optima, Hill-Climbing must continuously prevent early convergence to further explore the search space. While individual solutions in an Evolutionary Algorithm may similarly converge to local optima, the population as a whole tends to explore a larger subset of the search space.

Furthermore, many of the operations applied to candidate solutions are probabilistic, providing further differentiation between algorithm executions which can be taken advantage of with a restart routine.

Evolutionary algorithms have been successfully used to solve a vast array of problems, including automatic antenna design [22], automatic software verification [23] as well as many classic optimization problems such as scheduling and routing problems.

2.1.5. Particle Swarm Optimization

Particle Swarm Optimization is a population-based metaheuristic inspired by swarming behaviours in biology, first proposed in [24]. A set of solutions explores the search space by iteratively moving around according to predefined functions. Commonly, a particle's movement is influenced by it's best known previous position and the population's best known position, though alternative information-sharing routines have been developed to prevent premature convergence to local optima. [25]

PSO algorithms have been applied to health-care, environmental and industrial issues, as well as general optimization tasks. [26]

2.2. Metaheuristic Tuning

Among the most frequently utilized tuning methodologies we find:

- Meta-Optimization: Applying local search metaheuristics to search the parameter space of a metaheuristic. For example, ParamILS [27] implements Iterative Local Search akin to Hill-Climbing, while Evoca [28] applies an Evolutionary Algorithm to search the parameter space of a target problem.
- Racing Algorithms: Brute force based algorithms that test a set of parameter configurations on a set of instances, simulating a race, and disqualifying parameter configurations whenever they prove to be of lesser quality. This reduces the amount of time spent on lesser configurations so it can be invested in further testing of more promising options. Example: IRace [29].
- Model-Based Optimization: A regression model is constructed to predict the performance of untested parameters by interpolating between tested parameters. Further parameter testing is then focused on sections of the parameter space with highest statistical uncertainty in order to accelerate the discovery of promising options.

2.3. Automatic Metaheuristic Design

The authors of [1] formulate the problem of automatic metaheuristic design generically as an optimization problem in which the expected value of an algorithms evaluation, given a problem domain, is to be maximized. 2.1 describes the process of automatic metaheuristic design.

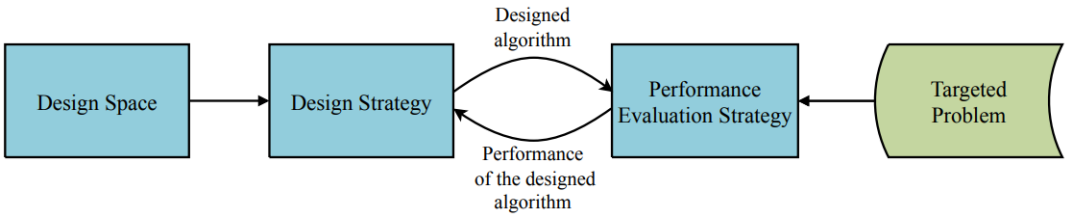


Figure 2.1: Automatic metaheuristic design process, Zhao et al. [1]

The Design Space is constructed from every possible parameter and operator value initially

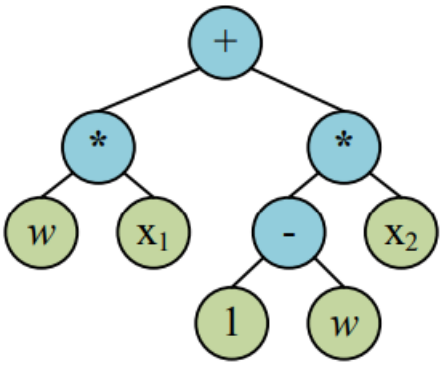


Figure 2.2: Genetic Programming Tree, Zhao et al. [1]

defined. From this set, algorithms are generated according to a Design Strategy. Such algorithms are evaluated over their performance on the target problem and their evaluations promote future changes in the iterative automatic design process.

One approach to automatic metaheuristic design applies binary trees similar to the ones utilized for genetic programming [30] (see Figure 2.2). An algorithmic operator can be dynamically constructed through the permutation and modification of non-terminal nodes. [31] serves as an example, where genetic programming was utilized to improve the travel function of particles in a *Particle Swarm Optimization* (PSO) algorithm.

2.4. Machine Learning

2.4.1. Supervised Learning

A general schema of Supervised Learning is as follows: A model M is chosen to be *trained*. In this context, *training* is understood as the learning of a model's parameters (a vector β) from a set of training data.

The optimal set of parameters for a set of data, $\hat{\beta}$, corresponds to the set that minimizes the prediction error on said data. In the case of Classification problems, error metrics such as

the Mis-classification Rate are used. In the case of Regression Problems, error metrics such as Mean Squared Error are used.

As simply minimizing a model's error metric on a set of data is prone to *over-fitting*[3], it is standard to separate the available data into three disjoint subsets:

- Training Set: Used to find a model's optimal set of parameters, $\hat{\beta}$.
- Validation Set: Used to fine-tune a model's *hyperparameters*[4] in order to maximize it's capacity to *generalize*. A model's training is typically stopped once it's prediction accuracy for the validation set is near it's accuracy for the training set, as further training would only allow the model to over-fit the training data.
- Testing Set: Also known as the hold-out set. This data is utilized exclusively to evaluate a finished model in order to confirm it's effectiveness. It is not rare for the testing set to be non-existent during a model's training, instead being the set of future real data that the model will attempt to predict once actually implemented.

The following model descriptions correspond exclusively to models used for classification problems.

Naive Bayes

A simple model for classification problems, Naive Bayes assumes (very optimistically) that all features of the training data are linearly independent from one another given a target value and that all features are equally important for the prediction of the target. The model then attempts to compute the conditional probability of the target given a set of features. Naive Bayes classifies an input according to whichever target possesses the highest probability given said input.

Infamously, despite its assumptions being generally incorrect for any real-world application, Naive Bayes performs surprisingly well, especially on small datasets, where more complex

[3]When a model performs well on the training data but cannot predict future data correctly
[4]The set of parameters that dictate a model's training

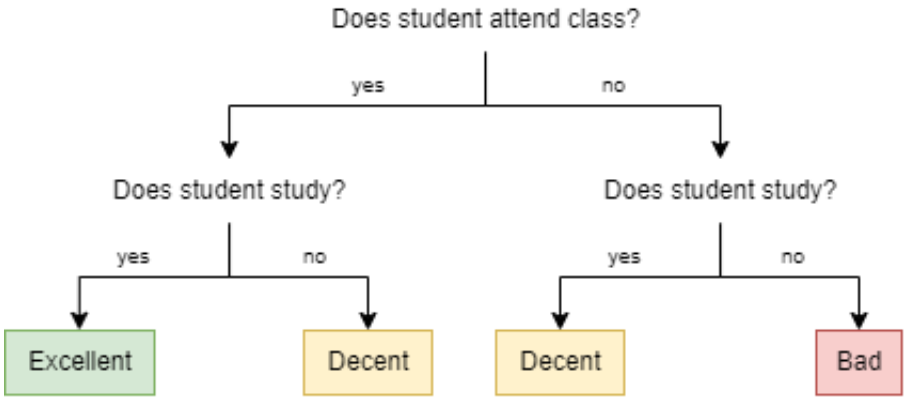


Figure 2.3: A decision tree to predict student grades

models are incapable of finding a quality set of parameters [32]. Thus, Naive Bayes is frequently used as a baseline model, providing ease of training and acceptable (sometimes exceptional) results.

Decision Trees

A decision tree is a supervised learning model that greedily and iteratively divides a dataset into subsets according to a *purity* or *entropy* metric. It attempts to model the relationship between a problem's features and it's target by defining a sequence of 'decisions'. [9]

While a single decision tree is limited by the greedy nature of it's construction, one can construct a set of distinct decision trees and utilize this set to democratically predict an input's target class. This is the approach taken by *Random Forest* models, an ensemble model that tends to outperform single decision trees by 'averaging out' the bias of a single tree's construction. [33]

K-Nearest Neighbours

K-Nearest Neighbours is a supervised learning algorithm proposed by Evelyn Fix and J.L. Hodges, Jr in [34]. It attempts to predict an input's class by comparing it's position in the feature-space with previously labeled inputs.

An input's class is determined by vote, where each data point's vote is weighed according to it's proximity to the new input.

Here, K is a parameter that determines the number of neighbours to query. While large values of K reduce the effect of noise on the classification, they also blur the boundaries between clusters.

K-Nearest Neighbours has been applied to a variety of domains, such as Recommender Systems, Image Similarity and missing data imputation.

SVMs: Support Vector Machines

Support Vector Machines are supervised learning models that construct a binary decision boundary from known data [35]. New inputs are classified according to whichever side of the boundary line that they are on.

While the decision boundary is necessarily linear, a *kernel trick* allows the model to perform non-linear classification by expanding the dimensionality of the inputs through kernel transformations.

SVMs have enjoyed widespread use in many domains [36], including the supporting of medical decisions, time series predictions and face authentication, as well as being one of the most popular and well studied classification models.

ANNs: Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by the interconnected neurons of the human brain [37]. Just as electrical signals propagate throughout the brain to transmit sensory and cognitive information, ANNs define propagation algorithms that process inputs and learn from expected outputs.

A neuron is defined as a mathematical function that takes a numerical input and produces a numerical output. Usually, a neuron's input will be a linear combination of many other

neuron's outputs. An ANN can be defined as a sequence of interconnected neuron layers.

For an input vector of length I, an ANN's first layer must be of matching length, as each feature of the input vector is to be fed to it's corresponding neuron. In the case of a two-layered ANN, the remaining layer corresponds to the output layer. For binary classification problems, this output layer requires only a single neuron that will receive as input a linear combination of the input layer's outputs, and will output a value close to 0 for one class, and a value close to 1 for the other.

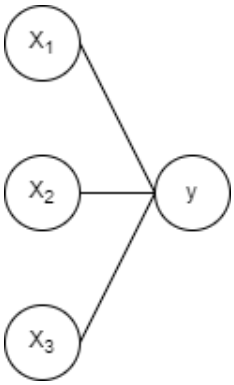


Figure 2.4: A 2-layered neural network that predicts a binary class for a 3 dimensional input vector

While two-layered ANNs may prove sufficient for very simple classification problems, their modelling capacity is not unlike that of a simple linear combination of inputs. To increase the model's complexity, additional *hidden-layers* can be added between the input and output layer.

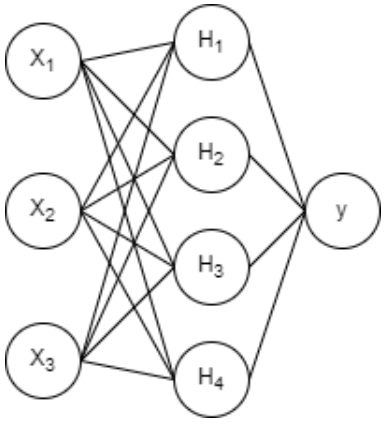


Figure 2.5: A 3-layered neural network for binary classification.

Through the addition of *hidden layers*, *dropout units* and a variety of problem-oriented pre-processing layers, neural networks develop the capacity to learn datasets of arbitrary complexity. Neural networks have been tremendously effective at solving many of AIs most difficult problems, such as language modeling [38], image recognition and classification [39], and automatic decision making in the context of high branching-factor games such as Go [40].

2.4.2. Unsupervised Learning

The biggest limitation facing supervised learning is the necessity for quality labelled data. For complex prediction problems a large amount of training data is required, which can usually only be labelled manually by humans.

Unsupervised Learning attempts to bypass this limitation by extracting information from a data set without prior knowledge of a target feature. Most unsupervised learning algorithms fall into the categories of *Clustering*, *Dimensionality Reduction* and *Anomaly Detection*. An example of each is therefore presented.

K-Means Clustering

K-Means Clustering is an unsupervised learning algorithm that attempts to find patterns in a data set. [41]

By starting with an initial amount of randomly selected centroids, the algorithm iteratively assigns data points to said centroids, forming clusters. After every assignment, the centroids are updated.

As the appropriate amount of clusters is initially unknown, K-Means Clustering is usually tested on a set of reasonable K values. The value of K selected will be whichever value produces the minimum average point-to-cluster distance.

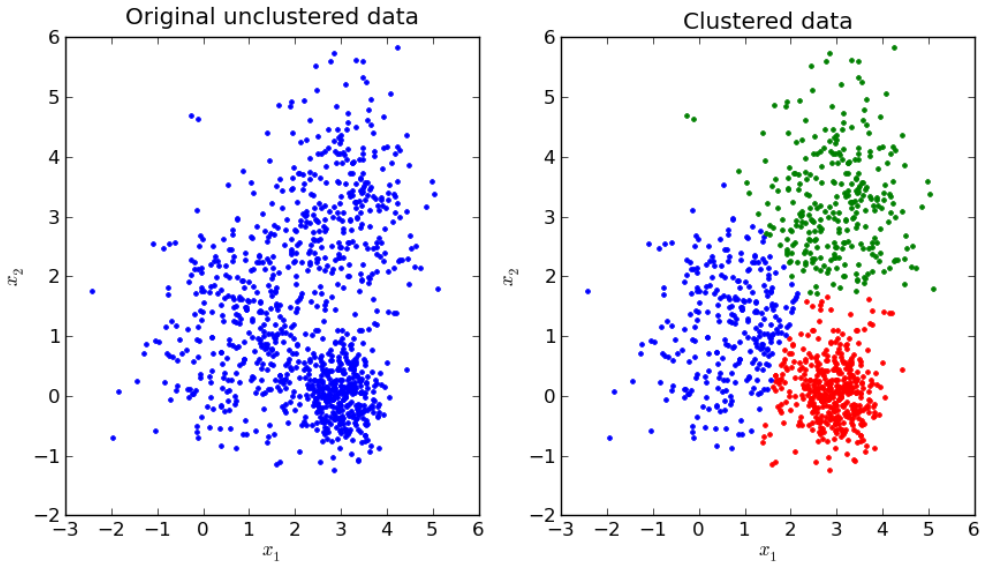


Figure 2.6: The input and output of a clustering algorithm [2]

Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction algorithm that attempts to map a set of features to a dimensionally reduced feature space while losing the least amount of information possible. For example, a set of three-dimensional data points may occupy a shared two-dimensional plane. Each data point can then be represented by a two-dimensional coordinate corresponding to said plane.

While real data is never perfectly linearly dependent, PCA can be applied to eliminate whichever dimensions provide the least amount of variance. This can prove useful for the pre-processing of supervised learning data, as eliminating features that are roughly linearly dependent will improve the performance of certain models, as is the case with Naive Bayes.

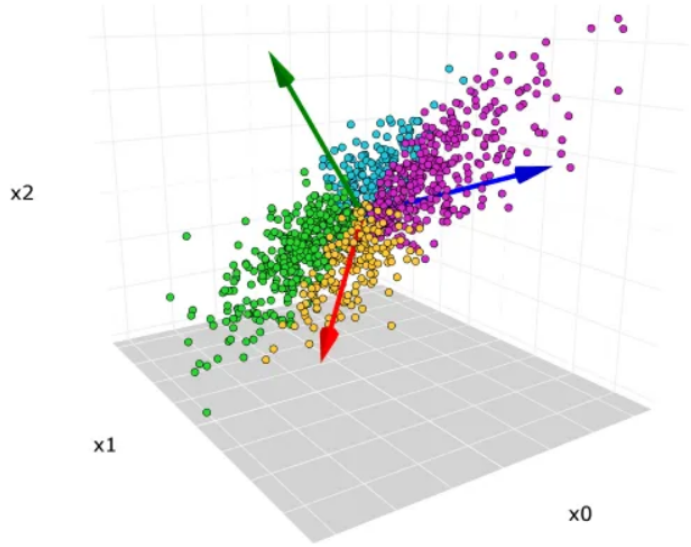


Figure 2.7: PCA applied to three dimensional dataset. The green axis provides little variance [3]

2.5. Machine Learning applied to Automatic Metaheuristic Design

The first applications of machine learning to the field of combinatorial optimization were focused on the substitution of both design decisions taken by experts and algorithmic decisions taken be metaheuristics. Generally, supervised learning has been used to approximate expert decisions, minimizing the distance between the action taken by the supervised learning model and the expert decision taken by the human.

As an example we provide the case of [42], where supervised learning was applied to determine whether linearizing a quadratic programming problem would reduce solve time. With a similar approach a metaheuristic's computational load can be reduced by approximating costly calculations with a regression model. Supervised learning can also be used to evaluate parametrizations on novel instances, potentially producing a parametrization better adapted to the particular instance.

Unsupervised learning has not been as frequently utilized in the field of combinatorial optimization. Some examples include the use of clustering in evolutionary algorithms to reduce

the cost of evaluating populations (Fitness Imitation) and the use of clustering algorithms to group problem instances to obtain a set of quality metaheuristic parameters per group (known as automatic algorithm configuration, see [43]).

As future work, [44] propose investigating the application of machine learning techniques to predict the evolution of dynamic instances to efficiently update a metaheuristic's parameters. They also reflect on the potential of utilizing unsupervised learning to group problem instances to discriminate their underlying distributions in order to optimize the search of parameters.

Chapter 3

Solution Proposal

The purpose of this project is to apply machine learning techniques to features extracted from CVRPTW instances to predict high quality parameter vectors for novel instances.

Rasku et al. [45] propose the use of a clustering algorithm to configure the parameters for each group of similar problem instances separately, such that when a new problem instance needs to be solved, the automatically configured parameters of the most similar instance group is used. Here, instances were clustered according to the similarity of their extracted features.

We intend to perform a thorough analysis of our set of extracted features, and to apply a variety of machine learning algorithms to determine which features and algorithms produce the best results.

3.0.1. A parameter prediction pipeline

The process for automatic metaheuristic configuration to be used in this project has been split into three steps:

1. Per-instance parameter tuning

2. Feature extraction

3. Parameter prediction

Figures 3.1 through 3.3 present the step by step process.

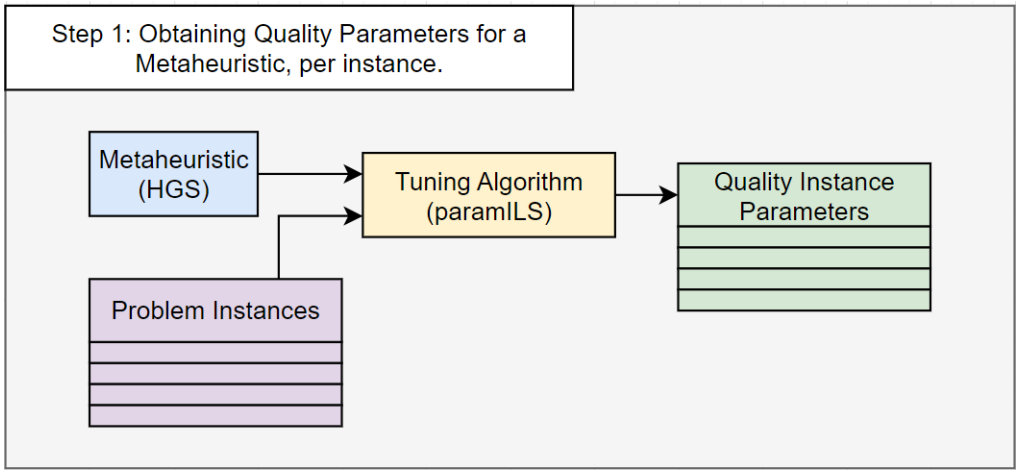


Figure 3.1: Per-instance parameter tuning

Given a set of instances and a metaheuristic, we run a tuning algorithm individually on each problem instance, thus obtaining a set of quality parameters where each member of the set is the best parameter vector found for an instance (figure 3.1). While the best possible results may be obtained by running an exhaustive search in the parameter space, using a tuning algorithm significantly reduces the computation effort invested.

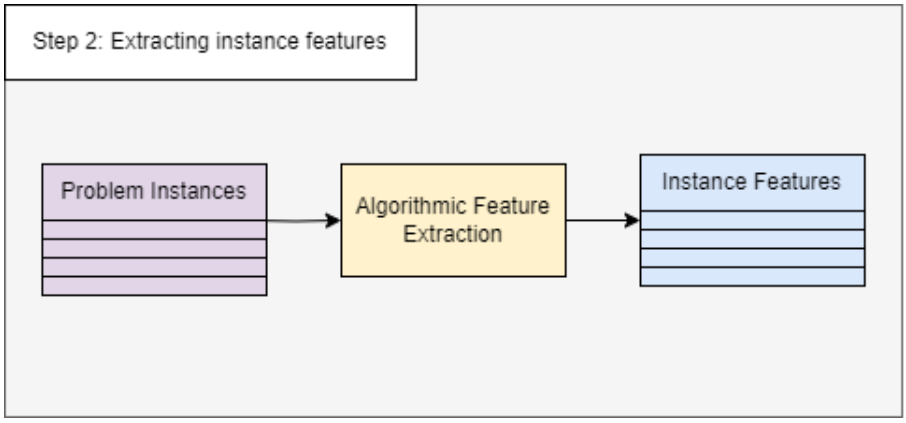


Figure 3.2: Per-instance feature extraction

Figure 3.2 shows how instances are submitted to a process known as *Feature Extraction*. The idea here is to quantify the characteristics of each instance by computing numerical features on a per-instance basis. Rasku et al. [45] presents a detailed summary of many of CVRP's most frequently utilized features. The features used in this project are presented and analyzed in Section 4.1.

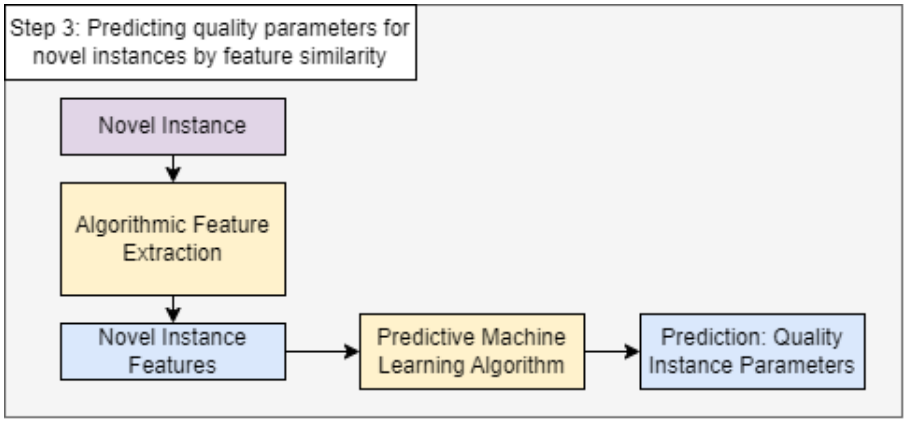


Figure 3.3: Parameter prediction for a novel instance

Figure 3.3 provides a sequence to predict quality parameters for a novel instance. By training a machine learning model on the extracted features, we learn a mapping of feature vectors to parameter vectors. We can then use this mapping to transform a novel instance's feature vector into a corresponding parameter vector.

3.0.2. Summary

A metaheuristic's performance on an instance depends on it's parameters. By extracting numerical features that characterize an instance and training a machine learning model to learn the mapping of said features to quality parameters obtained through a tuning algorithm, we can predict high quality parameters for instances that have not yet been solved.

Chapter 4

Implementation

This chapter presents the implementation details for a parameter-predicting machine learning pipeline. We explain which features are to be extracted from the CVRPTW instances, as well as measuring the quality of said features through an exploratory data analysis and validation through clustering results.

4.1. CVRPTW Instance Features

Three sets of features will be used to numerically describe CVRPTW problem instances:

Spatial Features

Mainly taken from Rasku et al. [45], this set of features describe the client distribution and the level of 'vehicle burden' present in each problem instance. We define vehicle burden as the amount of work that each vehicle must undertake for all clients to be served. Naturally, vehicle burden is inversely proportional to the of vehicles and their capacity, while increasing with client demands and the number of clients. Table 4.1 lists the spatial features to be used.

Feature	Objective
Number of clients	Measures the instance size
Distance from centroid to depot	Measures client distribution bias with regards to the depot
Avg. distance between depot and clients	Measures degree of clustering around the depot.
Avg. distance between centroid and clients	Measures client distribution density
CV of distance between centroid and clients	Measures client distribution heterogeneity
Ratio of mean client demand to capacity	Measures vehicle burden
Ratio of standard deviation of client demands to capacity	Measures vehicle burden heterogeneity
Ratio of clients to vehicles	Measures vehicle burden
Avg. of distances to nearest neighbour	Measures client distribution density
CV of distances to nearest neighbour	Measures uniformity of client distribution density

Table 4.1: Spatial features

Distance related features have been normalized through division by the longest possible distance present in each instance, which corresponds to the diagonal of the instance's bounding box. This prevents the instance size from influencing the values of other features.

For nearest neighbour features a value of $k = 2$ was chosen as presented in Rasku et al. [45].

While Rasku et al. initially decide to use mean, standard deviation, skewness, kurtosis and coefficient of variation for all spatial features, they end up applying PCA to reduce their 386 features to 7 components containing 71% of the data set's variance. This work uses only mean and coefficient of variation[1] metrics to reduce the *curse of dimensionality*. We also avoid using PCA in order to maintain the interpretability of our results.

Clustering Features

All clustering features are computed from the results of applying OPTICS clustering [46]. While OPTICS possesses a single parameter *min_samples* that is generally set at 4 for 2D space, this does not produce ideal results for all instances. The procedure used to determine the ideal *min_samples* value is shown in algorithm 2. As the runtime of each OPTICS execution is relatively low, we simply perform a linear search over the relevant parameter space,

[1] Coefficient of variation was used over standard deviation as the latter tended to be correlated with the mean

storing the result of highest quality according to the *clusteringQuality* function.

Algorithm 2 dynamicOPTICS

```
1: best_quality ← −INF
2: for min_samples in [2, 50] do
3:     clustering ← OPTICS(min_samples).fit(instance)
4:     quality ← clusteringQuality(clients, clustering.labels)
5:     if (quality > best_quality) then
6:         best_quality ← quality
7:         best_clustering[inst] ← clustering
```

We calculate *clusteringQuality* as:

$$Q = \alpha * D_{out} - \beta * D_{in} - \gamma * OR$$

where:

- $D_{out} = \frac{\sum_{c}^{|Clusters|} dist(centroid_C, centroid_{C_{neighbour}})}{|Clusters|}$ is the average inter cluster distance
- $D_{in} = \frac{\sum_{i=0}^{N_C}(dist(ClusterClients[i], centroid_C))}{N_C}$ is the average intra cluster distance.
- $OR = \frac{|Outliers|}{|Clients|}$ is the Outlier Ratio
- $\alpha, \beta, \gamma \in \mathbb{R}^+$
- N_C is the number of clients in a cluster C
- $C_{neighbour}$ is the cluster closest to C.
- $centroid_C = \frac{\sum_{i=0}^{N_C}(ClusterClients[i].x, ClusterClients[i].y)}{N_C}$ is the centroid of a cluster C.

Thus, maximizing the clustering quality means minimizing the number of outliers and the distance between clients of the same cluster, while maximizing the distance between clusters. This is similar to the Silhouette Coefficient [47], however it has the benefit of having

adjustable parameters ($\alpha, \beta, \gamma \in \mathbb{R}^+$) that can be tuned to modify the importance of each metric, while also including a penalization proportional to the number of outliers. The frequently used 'Elbow Metric' was not chosen as it is inherently subjective and requires either manual inspection or imprecise heuristics for its automatic implementation.

The range of *min_samples* $\in [2, 50]$ is more than enough for the set of instances, as values over 30 mostly produce a single cluster containg all clients. For α, β and γ, the values were determined from preliminary experiments and set as 1, 2 and 1, respectively. For a larger set of instances with tagged clusters it would be possible to use supervised learning to determine the optimal values of these parameters.

Figures 4.1 and 4.2 show two sets of clustering results. The first, with *min_samples* set to 4, and the second, with *min_samples* being automatically determined for each instance. For each case we display the best result as determined by the OPTICS algorithm. The of clusters is automatically determined by OPTICS.

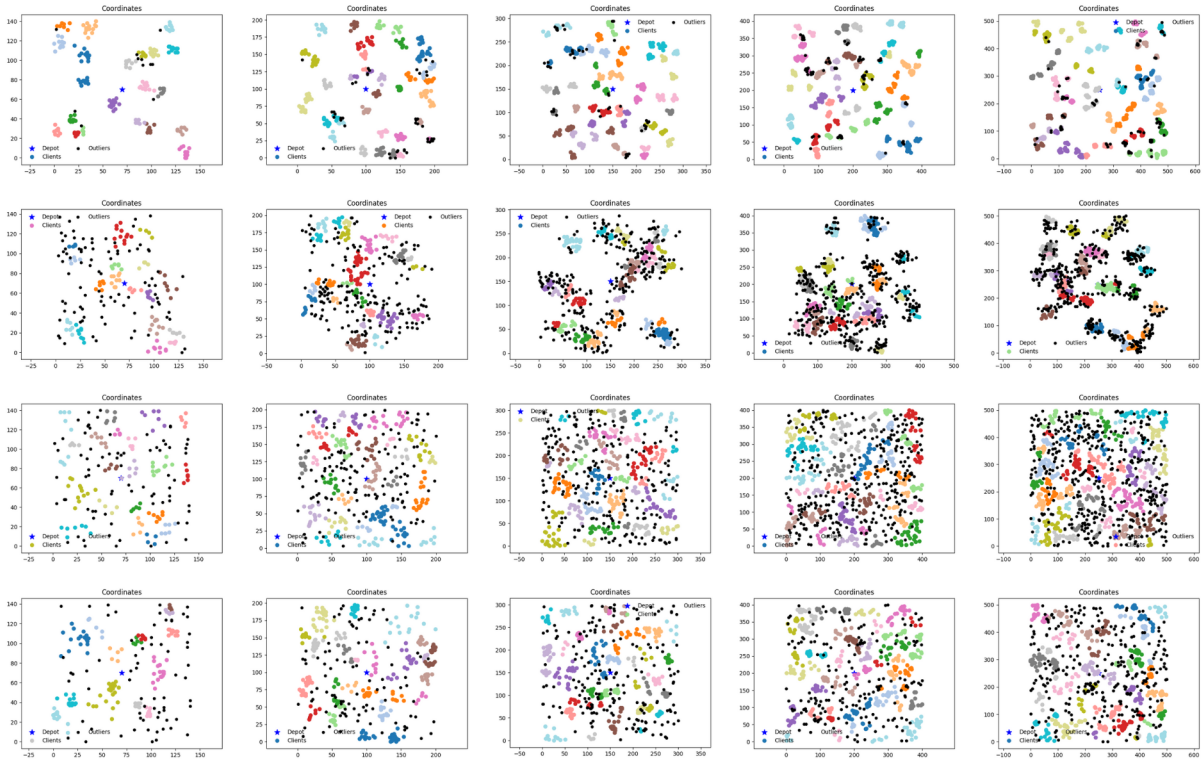


Figure 4.1: Clustering of Homberger instances with *min_samples* = 4. Black points are outliers not belonging to any cluster

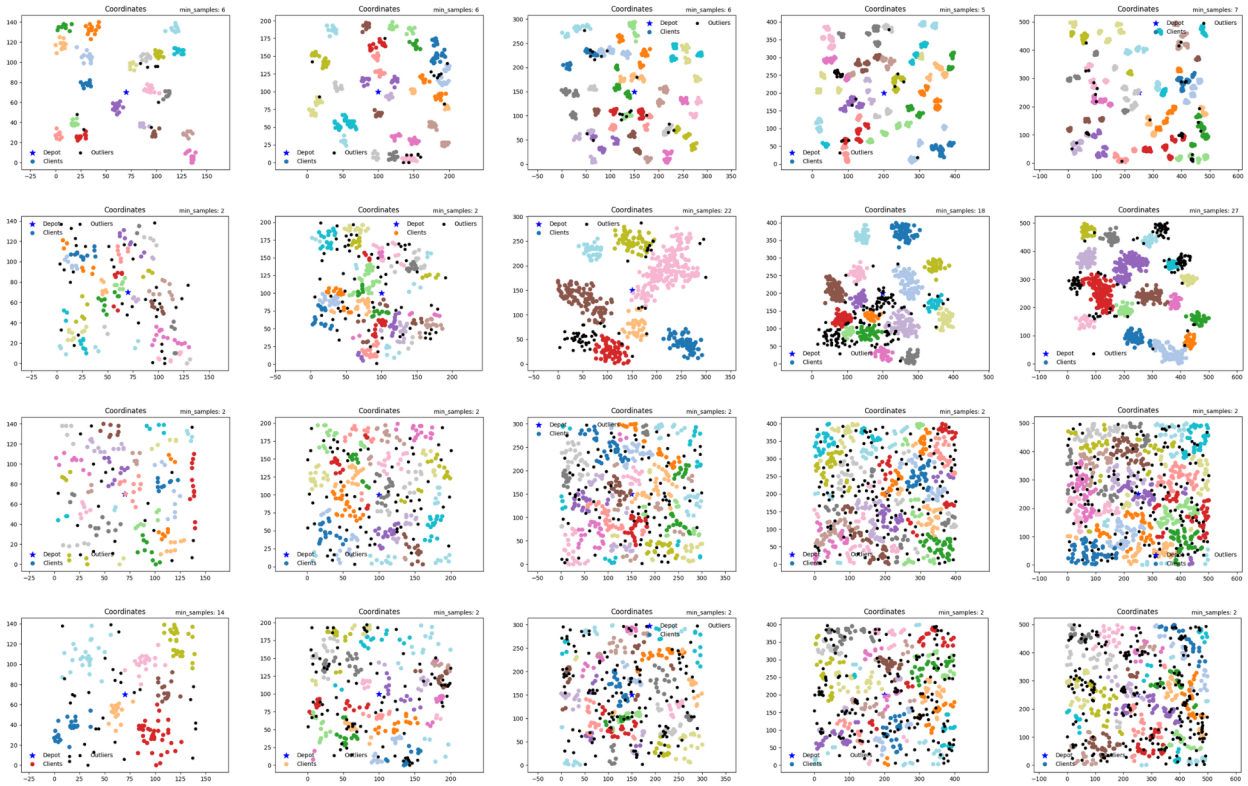


Figure 4.2: dynamicOPTICS clustering of Homberger instances. Black points are outliers not belonging to any cluster. Corresponding min_samples values are at the top of each instance

dynamicOPTICS produces significantly better results for sparsely clustered instances (second row) and noisy clusters (fourth row), however it also finds more clusters in cases with random noise (third row). Despite the latter, the metrics obtained from these instances is enough to differentiate easily clustered instances from noisy client distributions. Table 4.2 shows our interpretation of the relationship between the clusterability of an instance's client distribution with regards to its optimal *min_samples* value. Thus, we include the optimal *min_samples* value as a clustering feature to measure clusterability.

min_samples **value**	**Interpretation**
2	Instance does not possess clear clustering of clients.
3-7	Instance contains small, dense clusters
>7	Instance contains large or sparse clusters.

Table 4.2: Interpretation of instance clusterability with regards to optimal *min_samples* values

Finally, the clustering features considered for this work are listed in Table 4.3.

Feature	**Objective**
Optimal min_samples value	Measures clusterability
Cluster Ratio	Measures the number of clusters
Outlier Ratio	Measures the number of non-clustered clients
Avg. of clients per cluster	Measures cluster size (number of clients)
CV of clients per cluster	Measures cluster size heterogeneity
Avg. of intra cluster distances	Measures cluster density
Avg. of inter cluster distances	Measures cluster spread

Table 4.3: Clustering features

Time Window Features

Additionally, we propose four features related to the time windows of the CVRPTW instances. Table 4.4 lists the Time Window features considered.

Feature	**Objective**
Ratio of highest number of overlapping windows to total.	Measures tightest time interval
Ratio of average number of overlapping windows to total.	Measures average time tightness
Ratio of average window length to longest window.	Measures normalized time window length
Ratio of standard deviation of window length to longest window.	Measures normalized time window heterogeneity

Table 4.4: Time Window features

Discarded Features

When training a machine learning model, one must decide on a training sample vector of the smallest of features that carry a maximal amount of information. Table 4.5 presents the set of features that were discarded from the final working set. Most features were discarded for being highly correlated to other features while providing no novel information, while some features were discarded for being non-comparable between instances.

Feature	**Objective**	**Reason for Removal**
Average Client Demand	Measures vehicle burden	Does not provide information in a vacuum. Must be interpreted with regards to vehicle capacity. Replaced by 'Ratio between mean client demand and capacity'
Standard Deviation of Client Demands	Measures vehicle burden heterogeneity	Does not provide information in a vacuum. Must be interpreted with regards to vehicle capacity. Replaced by 'Ratio between standard deviation of client demands and capacity'
Ratio of Total Demand to Capacity	Measures vehicle burden	Redundant with respect to 'Ratio between mean client demand and capacity'
Ratio of Largest Demand to Capacity	Measures vehicle burden	Redundant with respect to 'Ratio between mean client demand and capacity'
Ratio of Median Demand to Capacity	Measures vehicle burden	Redundant with respect to 'Ratio between mean client demand and capacity'
Area Enclosing Rectangle	Measures client spread	Incomparable between differently sized instances. Strictly dependent on the of clients.

Table 4.5: Discarded Features

4.2. Exploratory data analysis for the proposed instance features

The following analysis considers two sets of CVRPTW instance sets: A set of 56 instances proposed by Solomon in [48] and a set of 300 instances proposed by Gehring and Homberger in [49]. These sets are described below.

4.2.1. Solomon Instances

The 56 instances proposed in [48] are characterized by having a fixed number of clients (100) and vehicles (25) with varying distributions of client positions, vehicle capacities and time windows.

The instance set is split into six subsets. These sets are listed in Table 4.6.

Set Name	Client Distribution	Capacity
C1	Densely Clustered	200
C2	Sparsely Clustered	700
R1	Random	200
R2	Random	1000
RC1	Semi Clustered	200
RC2	Semi Clustered	1000

Table 4.6: Solomon subsets

The main difference between the instances suffixed 1 and 2 is their *Scheduling Horizon* [48]. Lower vehicle capacity in C1, R1 and RC1 enforces a short scheduling horizon, reducing the number of clients that can be serviced by a single vehicle.

Highly relevant to the purposes of this thesis work is the variance in client distributions. From all 56 of Solomon's instances there are only four distinct client distributions (See Figure 4.3).

For the purposes of parameter prediction, we would require our instances to be as different

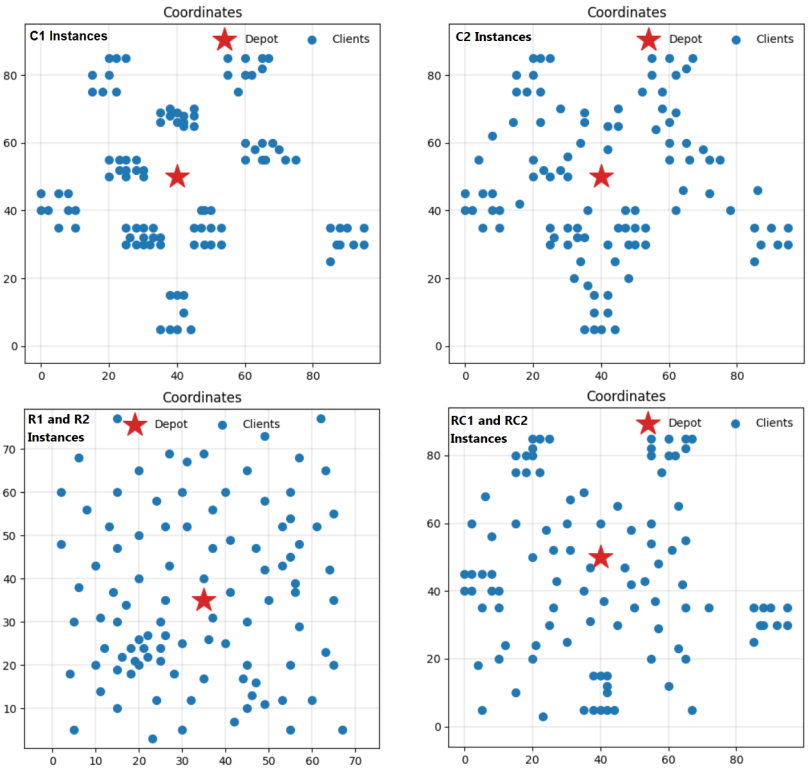


Figure 4.3: Client and depot distribution for Solomon instances

from each other as possible, such that each may require a specific parameter configuration that can be learned by a machine learning algorithm. The lack of variance in client distributions will therefore negatively affect a machine learning algorithm trained on this set of instances, as the data set of extracted instance features will only span a narrow subset of the feature space.

This does, however, prove useful to validate the effectiveness of instance clustering algorithms. That is to say, a quality clustering of Solomon instances with regards to spatial features can be easily validated by attempting to replicate the *a priori* grouping of the instance set.

Figure 4.4 shows a correlation matrix plot of the features extracted from Solomon instances. From these results we can observe a few insights:

- All the centroid and depot related metrics are highly correlated with each other, indicating that depot placement was likely determined from the clients' centroid.
- The *mean of client demands* is strongly correlated with their *coefficient of variation*, such that when an instance possesses large client demands they will also be of high variance.
- The time window related features are almost completely uncorrelated with the spatial features. This is to be expected as they were independently set [48].
- Curiously, the *cluster_ratio* is strongly correlated to the *average_distance_to_depot* (and centroid), as well as the *coefficient of variation of nearest neighbour distances*.
- Also of interest is the high correlation between the *average_number_of_clients_per_cluster*, *optimal_min_samples* and *average_inter_cluster_distance*.

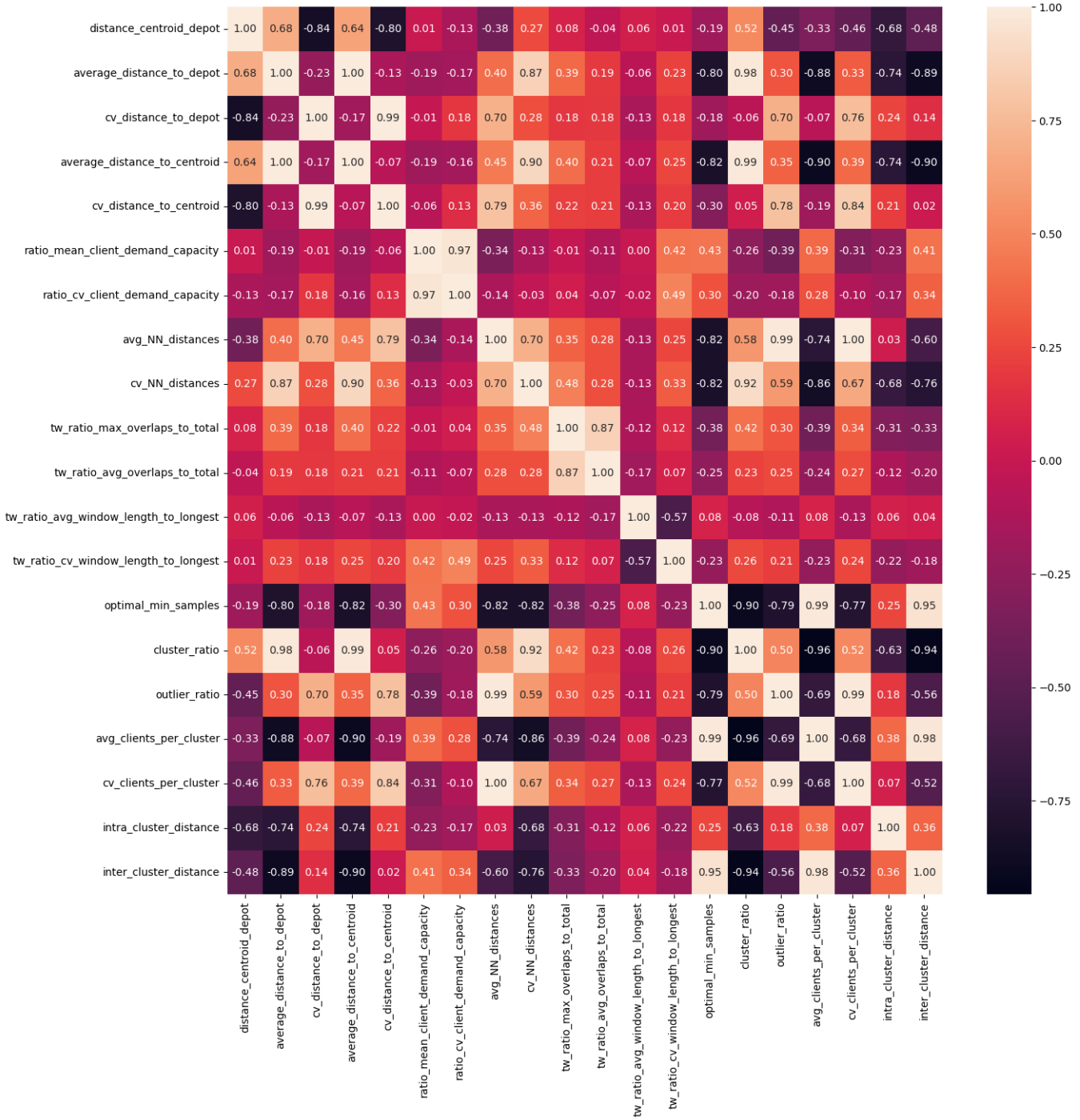


Figure 4.4: Correlation matrix of Solomon features. Constant-valued features have been omitted.

In Figure 4.5 we plot pairwise relationships between the most interesting correlated features.

While the data points do seem to follow a pattern of linear dependence, there are too few of them to draw proper conclusions. This is due to the lack of variety in client distributions. Figure 4.6 further reinforces this idea.

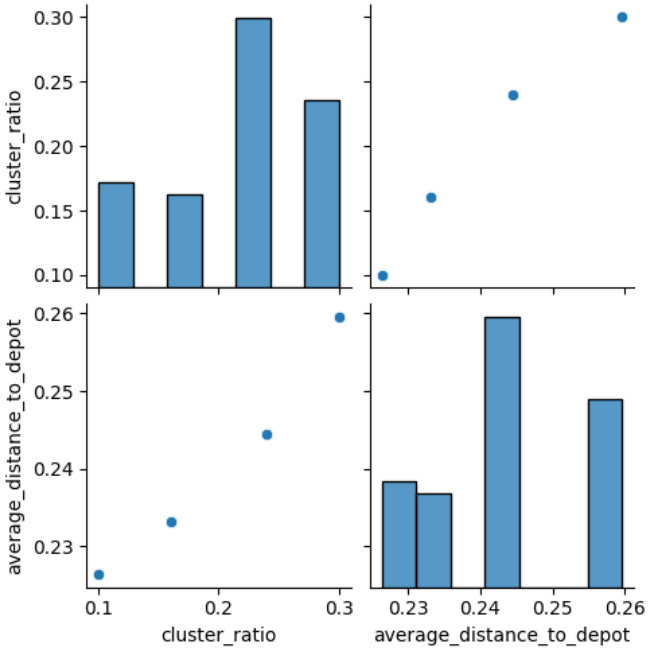


Figure 4.5: Pairwise plot between *cluster_ratio* and *average_distance_to_depot*.

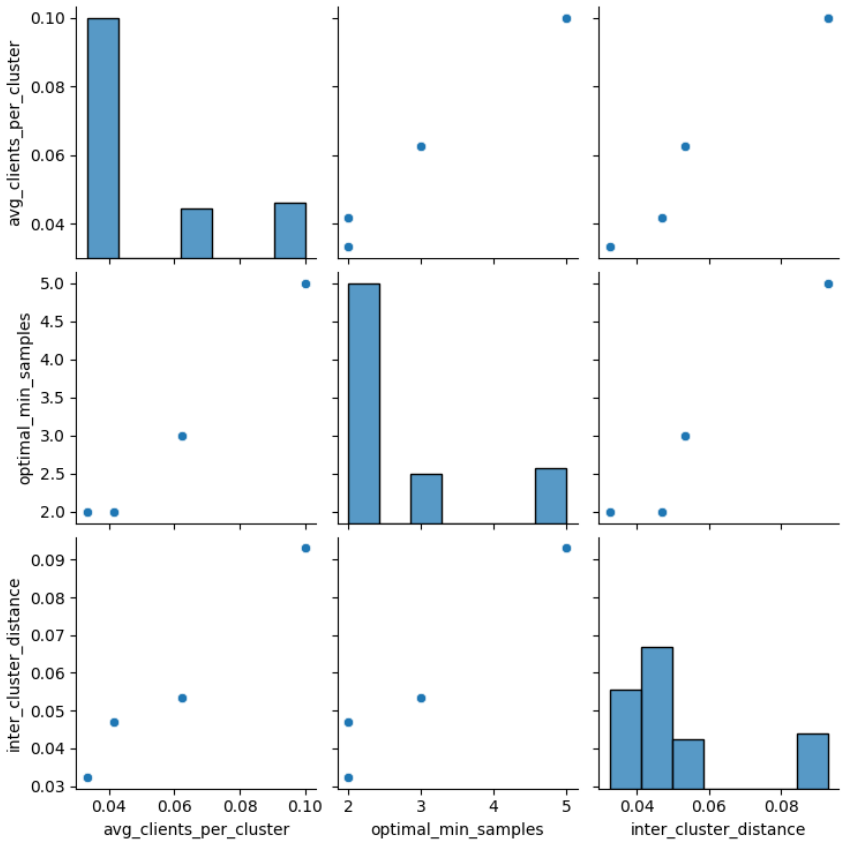


Figure 4.6: Pairwise plot between *average number of clients per cluster*, *optimal_min_samples value* and *inter_cluster_distance*.

Therefore, in order to identify dependence issues in our extracted features we must apply this analysis to a larger and more varied problem instance set.

4.2.2. Homberger Instances

The 300 Homberger instances are characterized by having varying number of clients (from 200 to 1000), a number of vehicles proportional to the corresponding number of clients and varying distributions of client locations, vehicle capacities and time windows.

The instances are once again split into six subsets shown in table 4.7.

Each subset is further split into five groups of instances with different number of clients and

Set Name	Client Distribution	Capacity
C1	Densely Clustered	200
C2	Sparsely Clustered	700
R1	Random	200
R2	Random	1000
RC1	Semi Clustered	200
RC2	Semi Clustered	1000

Table 4.7: Homberger subsets

vehicles.

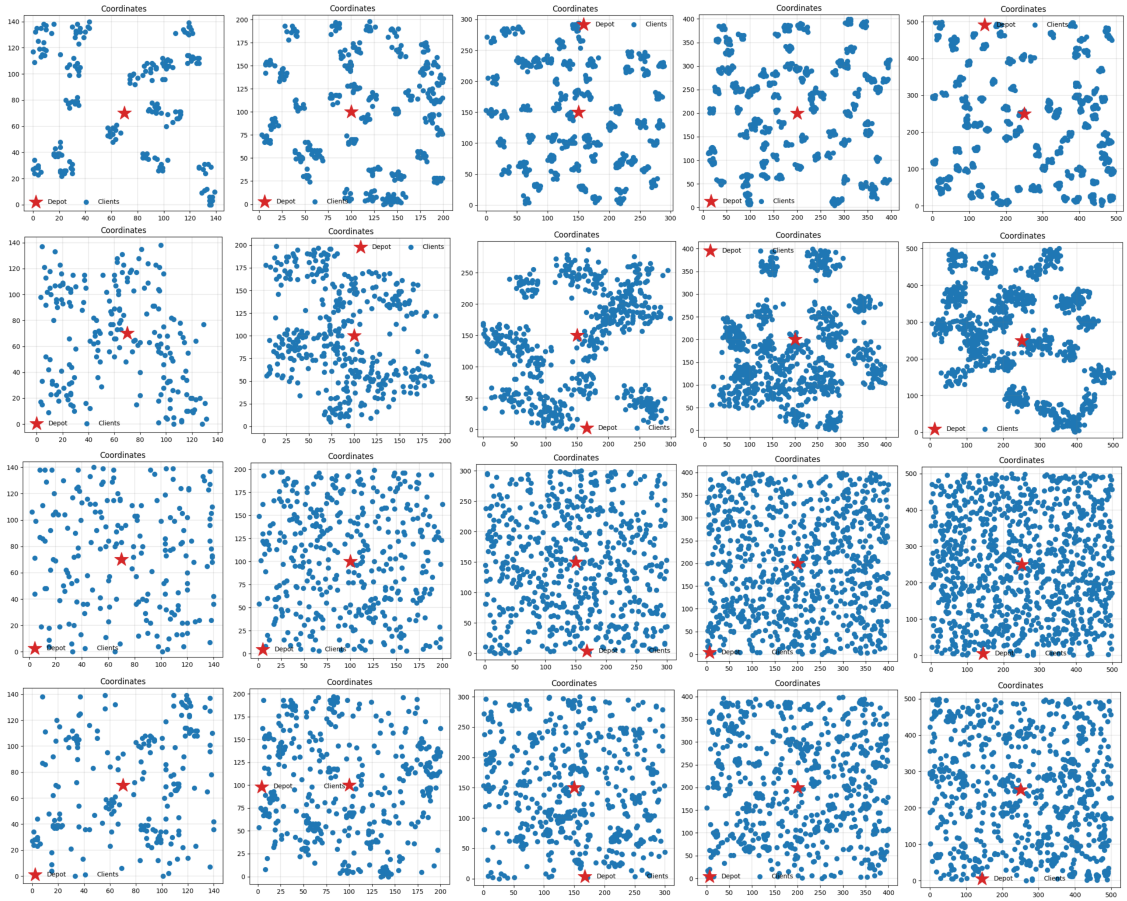


Figure 4.7: Client and depot distributions for all Homberger instances

Figure 4.7 presents all Homberger instance client distributions. Here each row corresponds to a particular group (C1, C2, R and RC respectively), and each column presents instances with a given number of clients (50, 100, 150, 200 and 250, respectively).

Figure 4.8: Correlation matrix of Homberger features. Constant-valued features have been omitted.

From the Homberger instances' correlation matrix we notice:

- Centroid and depot placement are once again strongly correlated.
- *Mean client demand* is again strongly correlated to the *coefficient of variation of client demand*. This means that even when adjusting for the difference in values in the data, their variance still grows with their mean.
- The *average clients per cluster*, *intra cluster distance* and *inter cluster distance* are highly correlated. This means that an instance that contains clusters with a large number of clients will, generally, have sparse clusters that are widely spaced out. On the other side, an instance that has clusters with few clients (such as randomly generated noise) will have dense clusters that are close to the others clusters.

In Figure 4.9 we plot the pairwise relationships between the *average clients per cluster*, *intra cluster distance* and *inter cluster distance* metrics. In the case of Homberger's instances there are enough distinct client distributions as to validate the positive correlation between these three metrics.

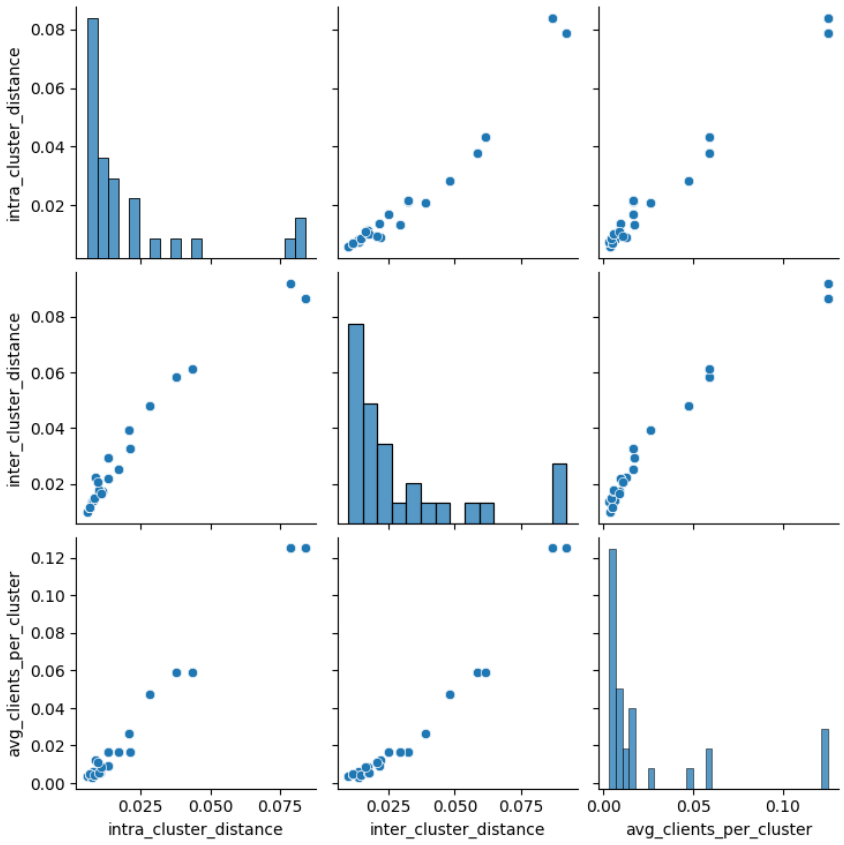


Figure 4.9: Pairwise plot of the *average clients per cluster*, *intra cluster distance* and *inter cluster distance* metrics.

In Figure 4.10 we plot the relationship between the mean client demand and the coefficient of variation of the client demand, both with respect to capacity. Here there is a clear clustering of instances, as there are only two distinct values for *capacity* in the set of instances. A clustering of all Homberger instances with respect to these capacity metrics would likely recreate the *a priori* grouping of these instances.

4.3. Feature validation through instance clustering

The objective of our extracted features is to holistically describe an instance through numerical values. A quality set of features must be enough to distinguish dissimilar instances from

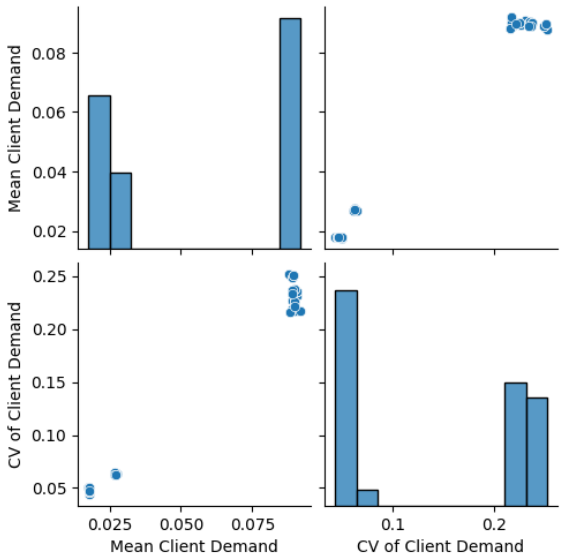


Figure 4.10: Pairwise plot between the *mean client demand w.r.r vehicle capacity* and *coefficient of variation of client demand w.r.r vehicle capacity*

each other, as well as to group similar instances together. As both Solomon and Homberger instance sets are grouped into subsets with different characteristics, we can validate the quality of our proposed feature set by clustering the instances with respect to different feature subsets and checking whether we obtain a grouping similar to the *a priori* grouping defined at the instance sets' creation.

For this purpose, KMeans[2] was chosen as we are searching for quality clustering results for a known number of clusters on each execution. In preliminary experiments, spectral clustering [50] was briefly tested, producing noisier results. For each execution, KMeans was run with 1000 random centroid initializations and the result of minimal inertia was chosen. All feature values were normalized to a $[0, 1]$ range before clustering as to equalize the weight of each feature. Despite this, spatial and clustering features will weigh more than time window and demand features when clustering with respect to the complete set, as the former two subsets contain more features than the latter.

[2]Lloyd's Algorithm

4.3.1. Validation on Solomon Instances

We first test our feature set on the set of 56 Solomon instances. We use two values of K, $K = 3$ and $K = 6$, to attempt to recreate the *a priori* grouping of the set.

Using all extracted features and the KMeans algorithm with $k = 3$ we can plot the results shown in Figure 4.11. We notice the groupings fit the three instance subsets: C, R and RC.

By using $k = 6$. Figure 4.12 shows the groupings perfectly matching the six predefined subsets: $C1$, $C2$, $R1$, $R2$, $RC1$ and $RC2$. The perfect clustering of Solomon instances, however, is not sufficient to prove the quality of the extracted features, as the Solomon set is small and contains only a few, very distinct client distributions.

Figure 4.11: Clustering of Solomon instances with 3 clusters. All features were used.

Figure 4.12: Clustering of Solomon instances with 6 clusters. All features were used.

4.3.2. Validation on Homberger instances

We now validate the feature set on the Homberger set. Whereas all features were used for the case of Solomon instances, the set of Homberger instances proves harder to cluster due to the similarity between instance types (mainly, *R* and *RC*) and the amount of instances. We first validate each feature subset independently (spatial, demand and clustering features). We conclude by presenting the clustering results using all extracted features.

Figure 4.13 shows the clustering results of all 300 Homberger instances using only spatial features, with $k = 3$. Both the *average_distance_to_centroid* and *cv_distance_to_centroid* features were discarded as they are perfectly correlated to depot-related features on Homberger instances. The *ratio_of_clients_per_vehicle* was also discarded as it is constant in the instance set. Each column corresponds to a certain number of clients, while each row contains 5 subsets of 10 instances each. The instances in these subsets differ only in their time window allocation. We notice that by clustering with respect to spatial features we successfully differentiate between small, medium and large problem instances. Larger values of k do not produce more granular distinctions between instance sizes. This is likely due to the lack of

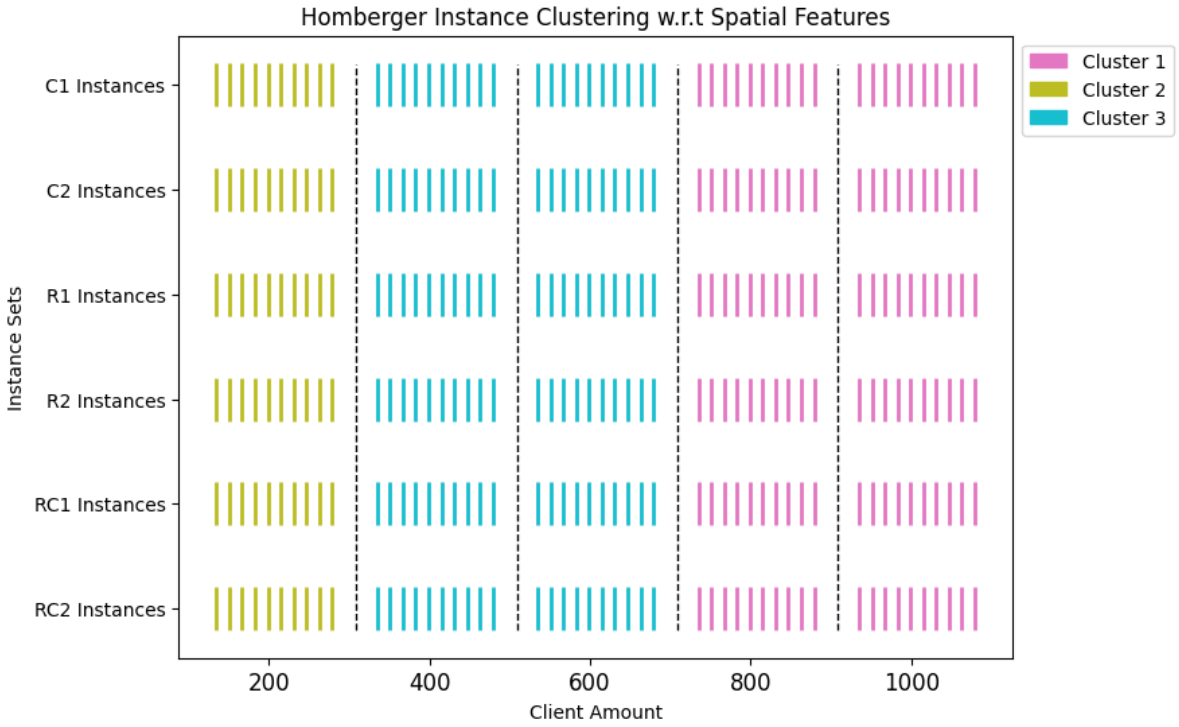


Figure 4.13: Clustering of Homberger instances with 3 clusters, using only spatial features.

inter-instance variance in depot- and centroid-related features.

Clustering with respect to the two demand-related features returns a perfect separation between the three values of vehicle capacity present in the subsets (see Figure 4.14). We notice how the $Capacity = 200$ subsets $C1$, $R1$ and $RC1$ are grouped together, the $Capacity = 700$ subset $C2$ is in it's own group, and the $Capacity = 1000$ subsets $R2$ and $RC2$ are also clustered together.

Figure 4.15 shows how clustering with respect to client-clustering features produces a one-to-one match with our instance clusterability matrix (see Figure 4.2). Returning to Table 4.2 (section *CVRPTW*, subsection *Clustering features*) we notice how all instances with an *optimal_min_samples* value of 2 were grouped into cluster 1. Optimal values in the range of $[3, 7]$ were grouped into cluster 2, and optimal values larger than 7 were grouped into cluster 3. Most importantly, this clustering result is robust to the exclusion of the *optimal_min_samples* feature, indicating that the rest of the clustering features adequately measure the clusterability of the instance's client distribution.

Figure 4.14: Clustering of Homberger instances with 3 clusters, using only demand-related features.

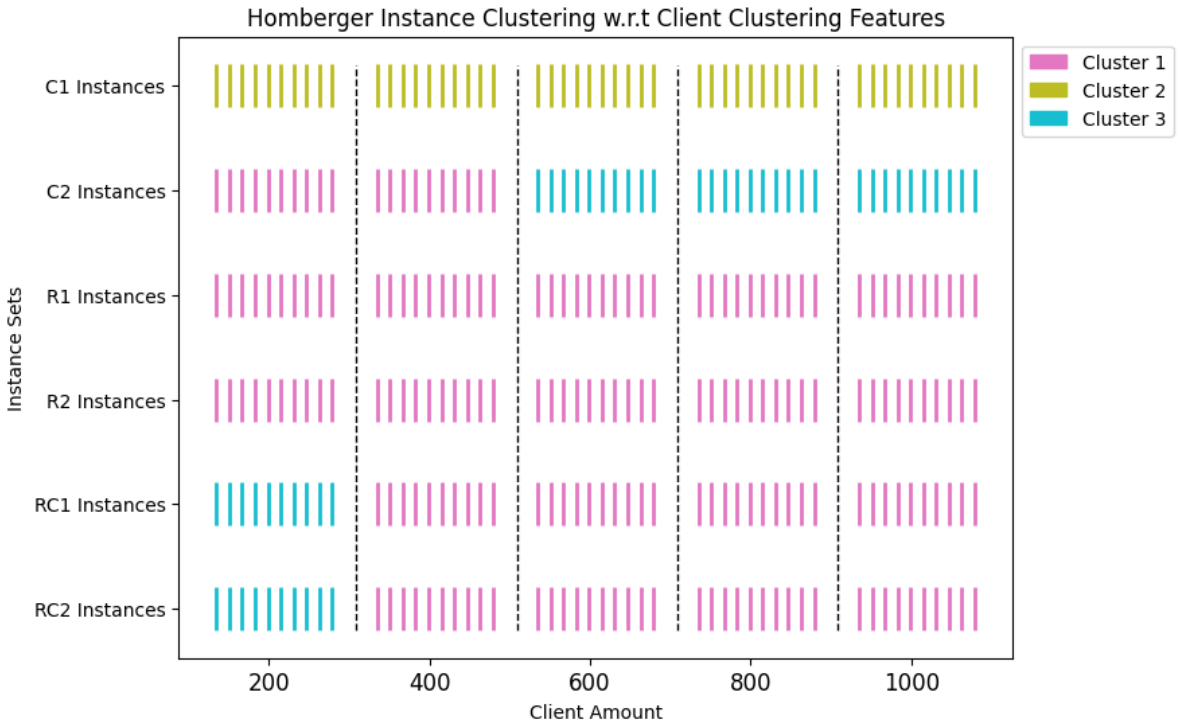


Figure 4.15: Clustering of Homberger instances with 3 clusters, using only client-clustering features.

We present the clustering results of Homberger instances with all extracted features and $k = 4$ in Figure 4.16. Results are noisier than on Solomon instances as there are five times more client distributions, with some of them, namely *R* and *RC*, being almost indistinguishable from a client-distribution perspective.

Most notable is the correct clustering of *C*1 instances, as these are most noticeable different from all others due to their distinctive client distributions. In order to understand the differences between clusters 2, 3 and 4, we present the Table 4.8 listing the most decisive features. Only features that present high inter-instance differences have been included. 'Low' and 'High' values are strictly relative. With all values being normalized to the $[0, 1]$ range prior to clustering, a value marked as 'High' will be 3 to 10 times larger than a value of the same feature marked as 'Low'. Cluster 4 correctly groups together instances that contain either large or sparse client-clusters, these instances also present high vehicle capacities. Finally, clusters 2 and 3 present very similar values for all features except *Capacity*[3]. This correctly distinguishes between type 1 and 2 instances as they are only different with respect to their maximum capacity in the case of *R* and *RC* instances. Cluster 3 does however incorrectly include the smaller *C*2 instances, indicating that the feature set was insufficient to distinguish high-capacity, small, densely clustered instances from high-capacity, large, non-clustered instances. This may be solved by including more client-clustering features in the feature set.

Cluster	Capacity*	NN Distance	Cluster Ratio	Outlier Ratio	Clients per Cluster	CV of Clients per Cluster	Intra Cluster Distance	Inter Cluster Distance
1	Low	Low	Low	Low	Low	Low	Low	Low
2	Low	High	High	High	Low	High	Low	Low
3	High	High	High	High	Low	High	Low	Low
4	High	High	Low	Low	High	Low	High	High

Table 4.8: Comparison of cluster centroids. $k = 4$, using all extracted features.

4.3.3. Exploratory data analysis conclusions

The current sets of CVRPTW instances are designed with metaheuristic experimentation in mind [48]. By setting a static distribution of clients and varying the number of vehicles, capacity and time window allocation, one may evaluate the efficacy of a metaheuristic as a function of said variables.

[3]Capacity* is measured as $\frac{1}{ratio\_mean\_client\_demand\_capacity}$ and was included to facilitate interpretation

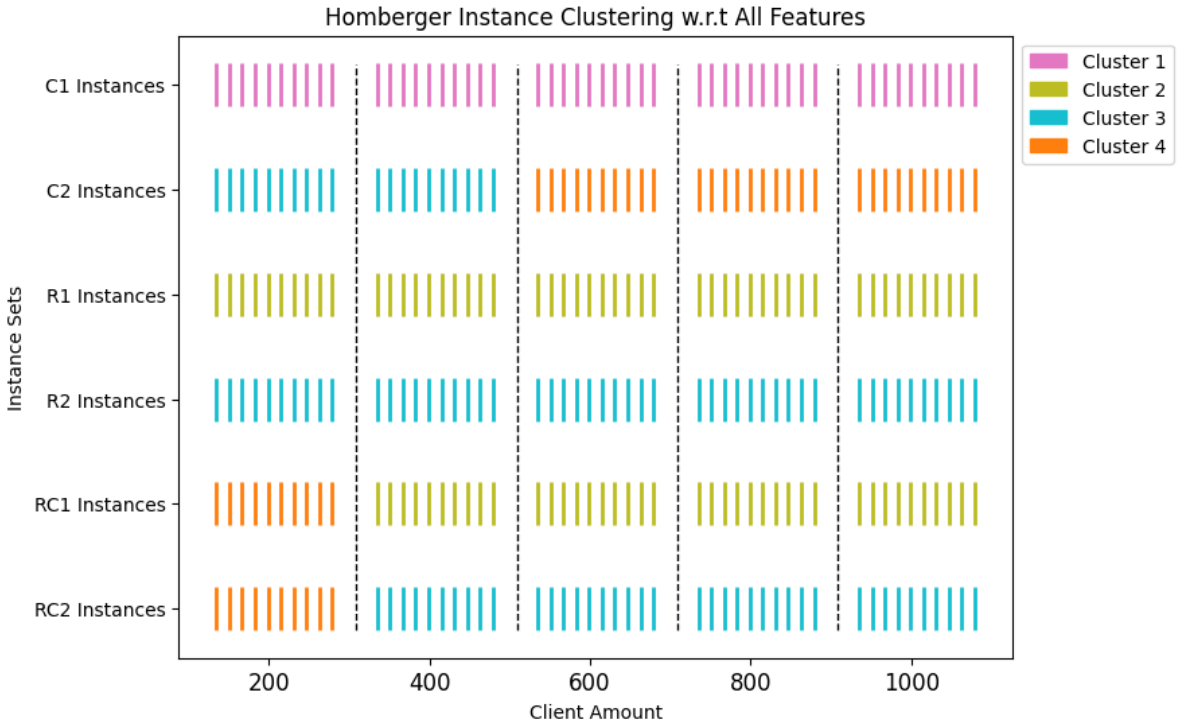


Figure 4.16: Clustering of Homberger instances with 4 clusters, using all features.

The position of the depot was selected to be close to the client's centroid in Solomon instances, while it was set to be the center of the client bounding box in Homberger instances. While this placement is of approximately good quality for the reduction of total travel time, it is not realistic with regards to real client-depot distributions, and will also fare worse than algorithmically determined placements [51].

Random client distributions in both Solomon and Homberger instances were obtained by sampling a uniform distribution in the plane. Thus, skewed client distributions are missing from both data sets.

In the case of Homberger instances, the number of vehicles is determined as a function of the number of clients. While this guarantees that there are enough vehicles to satisfy client demands, it prevents us from analyzing instances where vehicles are either 'overburdened' or 'underburdened'.

The lack of distribution variety in published CVRPTW instances will likely reduce the effectiveness of machine learning models, especially with regards to their ability to generalize to arbitrary instances.

According to Smith-Miles et al. [52], an approach to training machine learning models for metaheuristic selection is to randomly generate instances based on statistically varying features known to relate to instance difficulty (such as vehicle , capacity, and client distribution). This could be done with regards to the previously presented extracted features, but is beyond the scope of this project.

4.3.4. Summary

The set of features we used to characterize CVRPTW instances was validated by successfully recreating the *a priori* groupings of the instance sets with the use of clustering algorithms. We conclude a novel instance that is similar to a known instance will be identified as such by the similarity of their extracted feature vectors.

Chapter 5

Experiments and Results

What follows is a description of the results obtained by applying machine learning algorithms to learn the mapping of feature vectors to parameter vectors.

Computation was performed on a 12 core, Intel(R) Xeon(R) CPU E5-2680 v3 at 2.50GHz. ParamILS was run with a 1,000 evaluation budget limit per problem instance. HGS was run for 10 seconds per instance. Total tuning time for all 356 instances was roughly 100 hours of compute time. To compare the quality of the predicted parameter sets, we ran HGS on each instance-vector pair and computed the mean of the resulting objective function values. To reduce the effect of HGS's stochasticity on our results, we ran HGS 5 times on each instance-vector pair and averaged the results.

5.1. Target metaheuristic

To evaluate the previously proposed framework, we selected the Hybrid-Genetic Search algorithm [53] as implemented in [54]. HGS was first proposed as an algorithmic framework to address three VRP variants: multi-depot VRP, periodic VRP and multi-depot periodic VRP with capacitated vehicles and constrained route duration. In [55] this framework was extended to incorporate a large class of time-constrained VRPs including multi-depot VRP with time windows, multi-depot VRP with time windows and vehicle-site dependencies VRP

with time windows.

PyVRP is an open-source, state-of-the-art VRP solver that implements HGS in C++ through a Python interface. The pseudocode in algorithm 4 shows the Hybrid-Genetic Search algorithm structure as implemented in *PyVRP* [54].

HGS is an evolutionary algorithm that manages both a feasible and infeasible population. Every iteration, two solutions are selected from the whole population for *crossover* to produce one novel solution. This novel solution is then improved using a local-search procedure and added to the corresponding population. HGS attempts to maintain a static proportion of feasible to infeasible solutions. Thus, infeasible solutions may be repaired to become feasible. Importantly, parent solutions are selected not only for their quality, but also for the contribution they provide to the *diversity* of the population as a whole. Whenever the size of the population exceeds a threshold, the remaining solutions are removed from the pool until it is back to its minimum allowed size.

Algorithm 4 Hybrid-Genetic Search (HGS)

```
Initialize population
while number of iterations without improvement < It_NI and time < T_max do
    Select parent solutions P_1 and P_2
    Generate offspring C from P_1 and P_2 (crossover)
    Educate offspring C (local search procedure)
    if C is infeasible: then
        Insert C into infeasible subpopulation; repair with probability P_rep
    if C is feasible: then
        Insert C into feasible subpopulation
    if maximum subpopulation size is reached: then
        Select survivors
    Adjust penalty parameters for violating feasibility conditions
    if best solution has not been improved during It_div iterations: then
        diversify population
Return best feasible solution
```

5.1.1. HGS's parameter search space

The six parameters defined for HGS in [54] are as follows:

- μ: Minimum population size; lower bound of the number of solutions that are allowed to exist in either the feasible or unfeasible subpopulation.
- λ: Generation size; the number of solutions over μ allowed to exist. λ solutions are eliminated through a survivors selection phase once the maximum number is reached.
- $n_{\texttt{Elite}}$: Number of elite solutions considered in the fitness calculation.
- $n_{\texttt{Closest}}$: Number of close solutions considered in the diversity-contribution measure of the fitness solution.
- Γ: Granular search parameter; determines the number of nodes considered to be in the neighborhood of any given node. It is used for solution neighbourhood construction.
- ξ^{REF}: Target proportion of feasible individuals to total population size.

Whereas *pyVRP*'s HGS implementation features a few additional parameters for modulating the *Penalty Manager*, only the previous six will be taken into account in order to streamline the prediction process.

The original paper defined $n_{\texttt{Elite}}$, $n_{\texttt{Closest}}$ and Γ as proportions of the population size μ, both [54]'s and *pyVRP's* implementations define them as *static amounts*. This means that running the metaheuristic with parameters $\mu = 5$ and $n_{\texttt{Elite}} = 10$, for example, should result in an error. While *pyVRP* has implemented fixes for these issues (such as always choosing the *final* value of $n_{\texttt{Elite}}$ as $min(\mu, n_{\texttt{Elite}})$), this means that some parameter combinations will not be testable. To remedy the issue, the interface developed for the purpose of this project will revert to proportional values for $n_{\texttt{Elite}}$, $n_{\texttt{Closest}}$ and Γ.

Thus, the parameter search space that ParamILS will search through is summarized in table 5.1.

	Values				
λ	10	20	**40**	70	100
$n_{\texttt{Closest}}$	0.1	**0.2**	0.4	0.6	0.8
$n_{\texttt{Elite}}$	0.1	0.2	**0.4**	0.6	0.8
μ	5	15	**25**	35	45
ξ^{REF}	0.1	**0.2**	0.4	0.6	0.8

Table 5.1: Parameter space

Bold values show the optimal values defined in [54]. The ranges of values for each parameter where obtained from the algorithm's original paper's *Computational Experiments* section (see [53]). Values for ξ^{REF} correspond to the median value of the proportion of feasible individuals. For the purposes of this project, the proportion is allowed to oscillate in the range of $[\xi^{REF} - 0.1, \xi^{REF} + 0.1]$

5.2. Tuning algorithm

ParamILS, as described in [27], was selected to explore the parameter space in search of high-quality configurations for HGS. It operates by iteratively adjusting a parameter vector and assessing the metaheuristic's performance across a set of instances for each newly configured setup. Whenever ParamILS converges to a local optimum, it disturbs the parameter vector to continue the exploration of the search space.

In this work, the tuning algorithm was applied on a per-instance basis resulting in a vector of quality parameters for each one. While the resulting parameter configuration is likely to be better than a random setting, it may not be the globally optimum parameter configuration, as ParamILS only explores a subset of the parameter search space.

Once having obtained a vector of quality parameters per instance, we can then learn a mapping of instance features to these vectors.

5.3. Parameter prediction using machine learning

The main issue concerning the prediction of a metaheuristic's optimal parameter configuration for a given instance is the stochastic nature of both the tuning algorithm and of the metaheuristic itself. When constructing a model to map an instance's features to it's optimal parameter vector, we must consider that an instance deterministically produces a feature vector, but may produce different parameter vectors depending on the execution of both the tuning method and the metaheuristic algorithm.

5.3.1. Predicting parameters using K Nearest Neighbours

Using the K Nearest Neighbours (KNN) algorithm [34], we can assign parameter vectors to novel instances by finding which of our known instances is most similar with respect to their extracted features. The advantage of this machine learning algorithm is that it can produce good results even with low amounts of data, as long the novel inputs don't stray too far from what the model has seen. For this case we joined both Solomon and Homberger instance sets, as we would like to populate the parameter vector space that we will search through as densely as possible.

A KNN model is first trained on all instances. We then find the nearest neighbour of each instance and use the parameter vector obtained through our tuning algorithm as the predicted parameter vector for the problem instance.

From these results, 41% of Homberger instances had their nearest neighbour belonging to the same instance type (C, R or RC) while 68% of Solomon instances do. This reinforces our notion that Homberger instance types are not as distinguishable as Solomon instance types are.

Benchmarking results will be compared with other prediction methods in subsection 5.3.3.

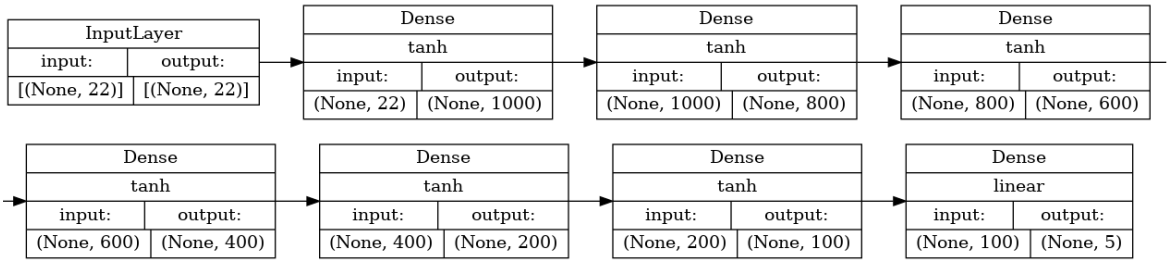


Figure 5.1: Neural network used to fit the feature data.

5.3.2. Predicting parameters using a neural network

In our approach, the neural network algorithm will take a vector of features as an input, and will produce a vector of parameters as an output. Given our training set of 356 samples (56 Solomon instances and 300 Homberger instances) we used the neural network showcased in figure 5.1. The network consists of a 22-neuron input layer, 6 hidden layers and a 5-neuron output layer. The amount neurons per hidden layer was manually tuned. All hidden layers use a hyperbolic tangent (*tanh*) as their activation function.

We trained the network with a 80/20 train/validation split using 5-fold cross validation, manually tuning the hyperparameters to minimize the average validation loss. The neural network's hyperparameters are as follows:

- Loss function: *Mean Squared Error*
- Optimizer: *AdamW*, $\alpha = 0.0002$
- Epochs: 1000
- Batch size: 100
- Patience: 200
- Activation function for all layers: *tanh*
- Activation function for output layer: *linear*

5.3.3. Predicted parameter benchmarking

To determine whether the parameter prediction has produced quality results, we compare nine sets of parameter vectors:

A. Base parameter values for the HGS metaheuristic (as defined in [54]).

B. Parameter values predicted by KNN with $K \in 1, 3, 5, 7, 9, 11$

C. Parameter values predicted by the neural network.

D. Parameter values samples from a random distribution.

The set of base parameter values (A) is constant for all instances as specified in the literature [54]. A set of random parameter values (D) is included to test the metaheuristic's sensitivity to parameter selection. Parameter values were randomly sampled, for each instance, using a uniform distribution covering the discrete parameter space used for parameter prediction (see figure 5.1). KNN was run on six different K values. $K = 1$ simply borrows the parameters from the instance's most similar neighbour, while $K = 3$ through $K = 9$ take the nearest neighbours and averages their parameter vectors. It should be noted that depending on the metaheuristic used, averaging of parameters may not make sense, as the metaheuristic's performance with respect to it's parametrization's location in parameter-space will, in all likelihood, be non-linear. These cases are nonetheless included in order to provide a more robust parameter prediction that is less influenced by noise in the instance set.

For each instance-vector pair, we ran five executions to reduce the effect of HGS's stochasticity on the results. For each execution, the metaheuristic was run with a 10 seconds time limit. In the case of the random parameter set (D), the parameters for each instance were randomized for every execution in order to minimize the effect of sampling bias.

The results obtained for Solomon instances were almost independent of the parameters used (see Tables 5.4 and 5.5). The instances are likely too small to provide a challenge to HGS, rendering it's parameters values irrelevant.

Parameter Set	Type	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Total Average
KNN1	C1	18431	18217	18204	18179	18211	**18248**	**16399**
	C2	8411	8606	8514	8513	8338	**8476**	
	R1	25623	25155	24432	24447	24415	**24814**	
	R2	14316	14160	14147	14128	14093	**14169**	
	RC1	20658	20925	20942	20907	20784	**20843**	
	RC2	11890	11855	11850	11833	11786	**11843**	
KNN3	C1	18051	18232	18159	18258	18015	**18143**	**16470**
	C2	8433	8593	8366	8626	8296	**8463**	
	R1	25425	25336	25264	25393	25419	**25367**	
	R2	13717	14198	14022	14222	13606	**13953**	
	RC1	21066	20935	20856	20987	21009	**20971**	
	RC2	12019	11953	11771	11929	11929	**11920**	
KNN5	C1	18488	18273	18178	18267	18418	**18325**	**16065**
	C2	8464	8557	8314	8610	8267	**8442**	
	R1	22474	22755	22661	22797	22406	**22619**	
	R2	14104	14203	14055	14282	13942	**14117**	
	RC1	21175	20910	20866	20962	21143	**21011**	
	RC2	11921	11953	11749	11923	11821	**11873**	
KNN7	C1	18251	18229	18217	18143	18159	**18200**	**16385**
	C2	8862	8687	8604	8370	8373	**8579**	
	R1	25357	25319	24023	23920	23940	**24512**	
	R2	14315	14333	14233	14058	14074	**14203**	
	RC1	21037	20987	20988	20859	20868	**20948**	
	RC2	11988	11936	11906	11745	11771	**11869**	
KNN9	C1	18291	18258	18459	18390	18372	**18354**	**16486**
	C2	8707	8707	9039	9046	8898	**8879**	
	R1	24130	24123	25406	22977	23118	**23951**	
	R2	14312	14252	14712	14671	14495	**14488**	
	RC1	20974	20935	21205	21131	21110	**21071**	
	RC2	11874	11924	12467	12302	12287	**12171**	
KNN11	C1	18290	18265	18217	18222	18121	**18223**	**16561**
	C2	8702	8823	8450	8560	8385	**8584**	
	R1	24067	24077	26456	26473	26016	**25418**	
	R2	14312	14351	14223	14290	14292	**14294**	
	RC1	20979	21001	20949	20976	20667	**20914**	
	RC2	11991	12004	11943	11973	11749	**11932**	

Table 5.2: Objective function evaluations for KNN parameter sets on Homberger instances. 5 runs were averaged to reduce the effect of noise on the results.

Parameter Set	Type	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Total Average
Base	C1	18514	18493	18508	18366	18373	**18451**	**16780**
	C2	9228	9243	9227	9020	8994	**9142**	
	R1	25355	24462	25340	25132	25126	**25083**	
	R2	14718	14721	14692	14481	14493	**14621**	
	RC1	21250	21242	21249	21117	21116	**21195**	
	RC2	12249	12265	12271	12072	12087	**12189**	
Random	C1	18531	18449	18238	18224	18158	**18320**	**16571**
	C2	9008	9177	8709	8672	8372	**8788**	
	R1	24133	24114	25323	25274	25197	**24808**	
	R2	14515	14748	14294	14255	14049	**14372**	
	RC1	21232	21225	20964	20966	20858	**21049**	
	RC2	12468	12390	11948	11894	11759	**12092**	
Neural Net	C1	18222	18242	18258	18200	17711	**18127**	**16347**
	C2	8510	8631	8635	8556	8385	**8543**	
	R1	25114	26455	24121	24081	22665	**24487**	
	R2	14371	14313	14217	14220	14052	**14235**	
	RC1	20717	20954	20978	20907	20685	**20848**	
	RC2	11859	11954	11933	11835	11615	**11839**	

Table 5.3: Objective function evaluations for base, random and Neural Net parameter sets on Homberger instances. 5 runs were averaged to reduce the effect of noise on the results.

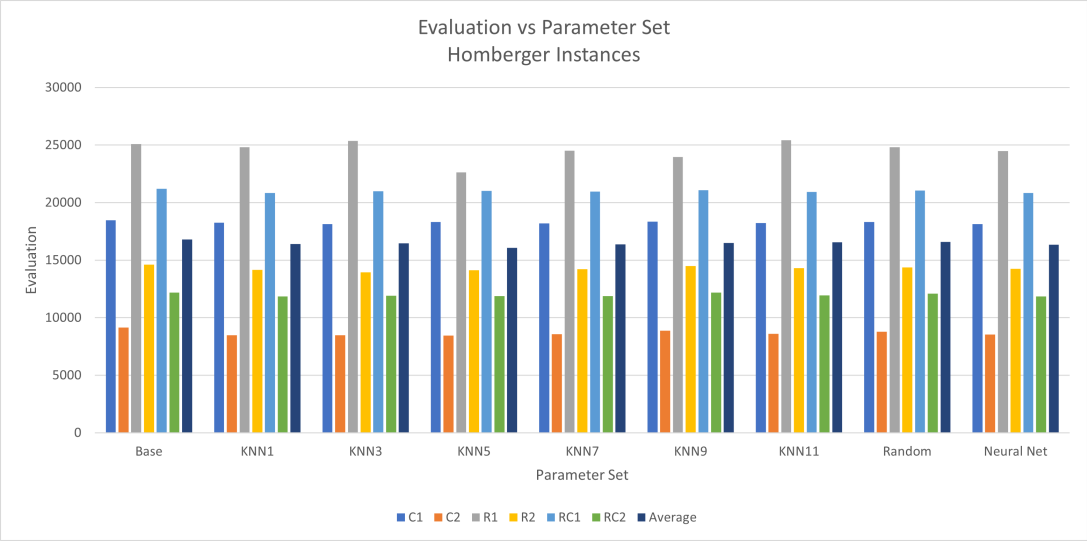


Figure 5.2: Objective function evaluation per parameter set for Homberger instances. KNN with $K = 5$ performed best with the largest difference in $R1$ instances.

ParameterSet	Type	Run1	Run2	Run3	Run4	Run5	Average	TotalAverage
KNN1	C1	828	828	828	828	828	**828**	**960,7**
	C2	588	589	589	589	588	**589**	
	R1	1157	1158	1157	1157	1157	**1157**	
	R2	861	865	865	865	864	**864**	
	RC1	1321	1331	1330	1330	1327	**1328**	
	RC2	1003	998	997	997	999	**999**	
KNN3	C1	828	828	828	828	828	**828**	**961,4**
	C2	589	589	589	589	589	**589**	
	R1	1157	1158	1157	1158	1157	**1157**	
	R2	867	866	865	866	865	**866**	
	RC1	1331	1331	1330	1330	1330	**1330**	
	RC2	998	998	998	998	998	**998**	
KNN5	C1	828	828	828	828	828	**828**	**961,2**
	C2	589	589	589	589	589	**589**	
	R1	1158	1158	1158	1158	1158	**1158**	
	R2	865	865	864	865	864	**865**	
	RC1	1330	1330	1329	1330	1329	**1330**	
	RC2	998	998	998	998	998	**998**	
KNN7	C1	828	828	828	828	828	**828**	**961,2**
	C2	589	589	589	589	589	**589**	
	R1	1157	1157	1157	1157	1157	**1157**	
	R2	865	865	864	865	864	**865**	
	RC1	1330	1331	1331	1331	1330	**1331**	
	RC2	998	998	998	998	998	**998**	
KNN9	C1	828	828	828	828	828	**828**	**961,3**
	C2	589	589	589	589	589	**589**	
	R1	1159	1158	1158	1158	1158	**1158**	
	R2	864	865	865	865	864	**865**	
	RC1	1330	1330	1330	1330	1330	**1330**	
	RC2	998	998	998	998	998	**998**	
KNN11	C1	828	828	828	828	828	**828**	**961,0**
	C2	589	589	589	589	589	**589**	
	R1	1157	1157	1157	1157	1157	**1157**	
	R2	865	865	865	865	865	**865**	
	RC1	1330	1330	1330	1330	1330	**1330**	
	RC2	997	997	997	997	997	**997**	

Table 5.4: Objective function evaluations for KNN parameter sets on Solomon instances. 5 runs were averaged to reduce the effect of noise on the results.

ParameterSet	Type	Run1	Run2	Run3	Run4	Run5	Average	TotalAverage
Base	C1	828	828	828	828	828	**828**	**962,0**
	C2	589	589	589	589	589	**589**	
	R1	1159	1158	1158	1158	1158	**1158**	
	R2	866	866	865	865	865	**865**	
	RC1	1331	1331	1331	1331	1331	**1331**	
	RC2	1001	1000	1000	1000	1000	**1000**	
Random	C1	828	828	828	828	828	**828**	**962,0**
	C2	589	589	589	589	589	**589**	
	R1	1159	1158	1159	1157	1158	**1158**	
	R2	866	866	866	868	866	**866**	
	RC1	1333	1334	1330	1330	1331	**1332**	
	RC2	999	999	999	999	998	**999**	
Neural Net	C1	828	828	828	828	828	**828**	**960,7**
	C2	588	589	589	589	588	**589**	
	R1	1156	1158	1158	1158	1156	**1157**	
	R2	860	865	865	865	860	**863**	
	RC1	1323	1331	1331	1331	1322	**1328**	
	RC2	1003	998	997	997	1003	**1000**	

Table 5.5: Objective function evaluations for base, random and Neural Net parameter sets on Solomon instances. 5 runs were averaged to reduce the effect of noise on the results.

Parameter Set	Instance set	Run 1	Run 2	Run 3	Run 4	Run 5	Average
Base	Solomon	968	969	968	968	968	**968**
	Homberger	16698	16886	16738	16881	16698	**16780**
KNN, K=1	Solomon	967	968	968	967	967	**967**
	Homberger	16289	16482	16486	16353	16373	**16397**
KNN, K=3	Solomon	968	968	968	968	968	**968**
	Homberger	16416	16490	16541	16407	16569	**16485**
KNN, K=5	Solomon	967	968	968	967	968	**968**
	Homberger	15977	16086	16108	15970	16140	**16056**
KNN, K=7	Solomon	967	968	968	968	967	**968**
	Homberger	16398	16579	16547	16582	16635	**16548**
KNN, K=9	Solomon	967	968	968	968	968	**968**
	Homberger	15967	16357	16300	16367	16381	**16274**
KNN, K=11	Solomon	967	967	967	967	967	**967**
	Homberger	16197	16329	16182	16420	16390	**16304**
Neural Net	Solomon	968	968	968	968	968	**968**
	Homberger	16617	16500	16758	16706	16749	**16666**
Random	Solomon	968	969	969	968	968	**968**
	Homberger	16380	16648	16684	16881	16420	**16603**

Table 5.6: Summary of objective function evaluations per parameter set.

With respect to Homberger instances, we obtain more interesting results. Surprisingly, the base parameters presented the worst average performance (see Figure 5.2) even when compared to random parameters. This may be explained by the fact that Vidal et al. [54] ran the HGS algorithm for far longer time intervals (between 4 and 40 minutes per instance), while we ran the metaheuristic with a 10 seconds time limit. It is likely that larger values of gs and ps allow for better exploration of the search space when runtimes are extended, while smaller values for these population size parameters work better when there is not enough computation time to thoroughly explore the search space.

A big disadvantage of this work's approach to parameter prediction is that it requires an extended period of time for parameter tuning to build the training data. As a tuning algorithm must run a metaheuristic thousands of times to obtain a result, an increase in the metaheuristic's execution time produces a multiplicative growth in parameter tuning time.

The best model was KNN with $K = 5$, having returned a 4% improvement over the base parameter set (see Table 5.6 for a summary of all results). While the differences between parameter sets are relatively small, this improvement demonstrates that, despite the stochasticity of HGS and ParamILS, and the relative insensitivity of HGS to its parametrization, improvements can be attained by learning a mapping between an instance's features and it's optimal parameter vector.

Given the size of the improvement, we consider that computation time would likely be better spent increasing HGS's run time or ParamILS's amount of allowed evaluations when looking for better solutions.

Machine learning models are only truly efficient when the amount of training data available is overwhelmingly large. In the theoretical case of possessing a widely spanning set of instances we would expect that parameter prediction through machine learning models would likely be of great advantage to the resolution of intractable problems, especially if one requires frequent, quick solutions of novel instances.

Conclusions

A thorough analysis of our set of 22 instance features was performed including validation through instance clustering. This set of features is robust enough to numerically describe CVRPTW instances in a holistic manner. The features are also computationally cheap, providing faster computation times when compared to the larger set of instances proposed in [45].

Parameter prediction was successful towards reducing the average travel time for short run times in Solomon and Homberger instances when compared to the base parameters proposed in [54]. The difference is however too small to justify the computation time spent on training a machine learning model.

Future work that could be performed based on this thesis work open questions includes:

(A) Constructing a wider spanning set of CVRPTW instances. This instance set may be generated with an evolutionary algorithm as mentioned in Smith-Miles et al. [52] with the objective of maximizing the variance of extracted features in the instance population.

(B) Testing this parameter prediction pipeline on a metaheuristic that is more parameter-sensitive.

(C) Applying this parameter prediction pipeline to tune the parameters of a metaheuristic for dynamically changing instances.

Bibliography

[1] Q. Zhao, Q. Duan, B. Yan, S. Cheng, and Y. Shi, "A survey on automated design of metaheuristic algorithms," 2023.

[2] U. Köse, "Techniques for adversarial examples threatening the safety of artificial intelligence based systems," 2019.

[3] C. Vision and I. Group, "3d morphable models," *Medium.com*, 2022.

[4] M. R. Garey and D. S. Johnson, *A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[5] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, New York, 2004.

[6] J. Robinson, "On the hamiltonian game (a traveling salesman problem)," 1949.

[7] Martello and Toth, *Knapsack Problems*. John Wiley Sons Ltd., 1990.

[8] T. G. Stutzle, *Local search algorithms for combinatorial problems - analysis, improvements, and new applications*. PhD thesis, Technischen Universitat¨ Darmstadt, 1998.

[9] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*. 1995.

[10] E. Montero, M.-C. Riff, and B. Neveu, "A beginner's guide to tuning methods," *Applied Soft Computing*, pp. 39–51, 2013.

[11] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, 1997.

[12] M. Gasse *et al.*, "The machine learning for combinatorial optimization competition (ml4co): Results and insights," pp. 1–12, 2022.

[13] S. Jacobson and E. Yücesan, "Analyzing the performance of generalized hill climbing algorithms," *J. Heuristics*, vol. 10, pp. 387–405, 07 2004.

[14] "Applications of a hill climbing method of optimization," *Chemical Engineering Science*, vol. 17, no. 1, pp. 45–52, 1962.

[15] "Palo: a probabilistic hill-climbing algorithm," *Artificial Intelligence*, vol. 84, no. 1, pp. 177–208, 1996.

[16] S. J. Wright, "Coordinate descent algorithms," 2015.

[17] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[18] P. Grabusts, J. Musatovs, and V. Golenkov, "The application of simulated annealing method for optimal route detection between objects," *Procedia Computer Science*, vol. 149, pp. 95–101, 2019. ICTE in Transportation and Logistics 2018 (ICTE 2018).

[19] D. Delahaye, S. Chaimatanan, and M. Mongeau, "Simulated annealing: From basics to applications," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), vol. 272 of *International Series in Operations Research & Management Science (ISOR)*, pp. 1–35.ISBN 978–3–319–91085–7, Springer, 2019.

[20] F. Glover, "Tabu search—part i," *ORSA Journal on Computing*, 1989.

[21] F. Glover, E. Taillard, and D. Werra, "Tabu search," *Annals of Operations Research*, pp. 3–28, 01 1993.

[22] G. Hornby, A. Globus, D. Linden, and J. Lohn, "Automated antenna design with evolutionary algorithms," *Collection of Technical Papers - Space 2006 Conference*, vol. 1, 09 2006.

[23] A. Tonda, E. Sanchez, and G. Squillero, *Industrial Applications of Evolutionary Algorithms*, vol. 34. 01 2012.

[24] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948 vol.4, 1995.

[25] J. Kennedy, "Population structure and particle swarm performance," *Proceedings of the Congress on Evolutionary Computation*, 2002.

[26] A. Gad, *Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review.* 2022.

[27] F. Hutter, H. H. Hoos, and T. Stützle", "Automatic algorithm configuration based on local search," *Proceedings of the twenty-second conference on artificial intelligence*, 2007.

[28] M.-C. Riff and E. Montero, "A new algorithm for reducing metaheuristic design effort," in *2013 IEEE Congress on Evolutionary Computation*.

[29] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[30] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and Computing*, (1994).

[31] R. Poli, C. D. Chio, and W. B. Langdon, "Exploring extended particle swarms: A genetic programming approach," *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, 2005.

[32] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. 2001.

[33] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, pp. 278–282 vol.1, 1995.

[34] E. Fix and J. L. Hodges, "Discriminatory analysis - nonparametric discrimination: Consistency properties," *International Statistical Review*, vol. 57, p. 238, 1989.

[35] Cortes and Vapnik, "Support-vector networks,"

[36] T. Evgeniou and M. Pontil, "Support vector machines: Theory and applications," 2001.

[37] W. McCullough and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulleting of mathematical biophysics*, 1943.

[38] M. U. Hadi, Q. Al-Tashi, R. Qureshi, A. Shah, A. Muneer, M. Irfan, A. Zafar, M. Shaikh, N. Akhtar, J. Wu, and S. Mirjalili, "Large language models: A comprehensive survey of its applications, challenges, limitations, and future prospects," 2023.

[39] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[40] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, 2017.

[41] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu, "An efficient k-means clustering algorithm: analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[42] P. Bonami, A. Lodi, and G. Zarpellon, "Learning a classification of mixed-integer quadratic programming problems," *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 595–604, 2018.

[43] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney, "Isac - instance-specific algorithm configuration," vol. 215, pp. 751–756, 01 2010.

[44] Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E.-G. Talbi, "Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art.," *European Journal of Operational Research.*, pp. 1–10, 2021.

[45] J. Rasku, T. J. Kärkkäinen, and N. Musliu, "Feature extractors for describing vehicle routing problem instances," in *Student Conference on Operational Research*, 2016.

[46] M. Ankerst, M. Breunig, P. Kröger, and J. Sander, "Optics: Ordering points to identify the clustering structure," vol. 28, pp. 49–60, 06 1999.

[47] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[48] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations Research*, 1987.

[49] H. Gehring, "A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows," 1999.

[50] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, 2000.

[51] A. Salawudeen, E. E. Akut, I. Momoh, A. Ibrahim, M. Zion, and S. Yusuf, "Depot location analysis for capacitated vehicle routing problem: A case study of solid waste management," *IJEEC - International Journaly of Electrical Engineering and Computing*, vol. 4, p. 132, 10 2020.

[52] K. Smith-Miles and J. van Hemert, "Discovering the suitability of optimisation algorithms by learning from evolved instances," *Annals of Mathematics and Artificial Intelligence*, 04 2011.

[53] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei, "A hybrid genetic algorithm for multidepot and periodic vehicle routing problems," *Operations Research*, vol. 60, pp. 611–624, 06 2012.

[54] T. Vidal, "Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood," *Computers Operations Research*, vol. 140, p. 105643, 2022.

[55] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins, "A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows," *Computers Operations Research*, 2013.