Erasmus University Rotterdam
Erasmus School of Economics
Master Thesis Econometrics and Management Science

Meta-Learning using Neural Networks in Large-Scale Traveling Salesman Problems

M.B. Bossenbroek (668865)

Supervisor:	dr. O. Kuryatnikova
Second assessor:	dr. MH Akyuz
Date final version:	7th August 2023

Abstract

This thesis explores the application of meta-learning to the traveling salesman problem (TSP) for large-scale instances that require low complexity methods. Traditional meta-learning models for TSP often come with increased computational time, making them unsuitable for situations where numerous instances need to be solved quickly. In this study, different meta-learning models are developed to predict the performance of fast and greedy heuristics based on instance features. The study employs three distinct techniques to address the research problem, namely a neural network, a decision tree, and k-nearest neighbors. The research investigates the models performance, influential features for heuristic selection, and the efficiency improvement compared to always selecting the overall best heuristic. Furthermore, the study introduces novel meta-features and investigates the viability of using artificial data as a replacement for real data during the models training processes. The findings demonstrate the models' ability to capture patterns in TSP instances and outperform simple selection techniques, providing a promising alternative for efficient TSP solutions on large-scale instances. The best performance is achieved by employing the complex neural network model. However, the relatively small performance differences compared to less complex models make them viable candidates, especially when considering computational efficiency.

List of Frequently Used Abbreviations and Symbols

Abbreviation	Definition	More Information
CO	Combinatorial Optimization	See Section 1
DBSCAN	Density-Based Spatial Clustering of Applications with Noise	See Section 4.5
DFJ	Dantzig–Fulkerson–Johnson formulation	See Section 3
DT	Decision Tree	See Section 4.4
FI	Farthest Insertion heuristic	See Section 4.1
GR	the Greedy Algorithm	See Section 4.1
KNN	K-Nearest Neighbours	See Section 4.4
ML	Machine-Learning	See Section 2.3
MST	Minimum Spanning Tree	See Section 2
MLP	MultiLayer Perceptron	See Section 2.4
NI	Nearest Insertion heuristic	See Section 4.1
NN	Nearest Neighbour heuristic	See Section 4.1
NNet	Neural Network	See Section 4.3
PCB	Printed Circuit Board	See Section 1
ρ	Spearman's correlation coefficient	See Equation 6
TSP	Traveling Salesman Problem	See Section 1

Contents

1 Introduction **5**

2 Literature review **8**
2.1 The Traveling Salesman Problem 8
2.2 Heuristics 9
2.3 Meta-learning 11
2.4 Neural Networks 13

3 Problem statement **15**

4 Methodology **17**
4.1 The Heuristics 17
4.2 Performance evaluation 24
4.3 Neural Network 24
4.3.1 Neural Network Architecture 25
4.3.2 Neural Network Training and validation 26
4.4 Benchmark models 28
4.5 Meta-features 29

5 Data **35**
5.1 Randomly generated instances 35
5.2 Modified instances 37
5.3 Test dataset 40

6 Results **42**
6.1 Data properties 42
6.2 Model validation 43
6.3 Model comparison 44
6.4 Effect of artificial data 49
6.5 Features importance 51

7 Conclusion **52**
7.1 Model performance 52
7.2 Choice of machine-learning technique 53
7.3 Artificial data and Feature importance 54
7.4 Model limitations and future research. 55

A Decision Trees **57**

B Overview meta-features **59**

C k-fold validation NNet **60**

D SET TSP **61**

E Online appendix **62**

1 Introduction

In Combinatorial Optimization (CO) problems, the goal is to maximize or minimize some objective function subject to one or more constraints over a finite number of outcomes. These problems often represent simplified versions of real-life scenarios. One of the most widely studied problems is the traveling salesman problem (TSP). The TSP requires a salesman to visit a number of cities and find the optimal route in which each city is visited exactly once. More formally, this problem can be described as the following graph problem [14]. Consider a graph $G = (V, E)$, where V represents a set of vertices and E represents the set of edges. Each edge $e \in E$ is assigned a cost c_e. The objective is to find a Hamiltonian cycle on the graph that minimizes the sum of all costs on the cycle. The problem has been proven to be NP-complete [13], indicating that it belongs to the most challenging problems within the class of CO problems and no known polynomial-time algorithms exist to solve it.

The TSP has a broad range of practical applications in various domains. Primarily, it is applied in the domain of logistics, where it finds use in scenarios such as planning of sightseeing tours by tour operators or transportation of students by a school bus [1] , delivery of packages using drones [34], and order-picking scenarios [40]. However, TSP has also been successfully implemented in other fields. Another, less apparent application of TSP is found in the domain of DNA sequencing. Here, it is utilized to determine the shortest sequence of DNA fragments that can assemble a complete set of genes [4]. While the aforementioned applications of the TSP typically involve a limited number of cities or locations, there are certain scenarios that require solving much larger instances of TSP.

In this thesis, our focus is on applications that necessitate the rapid solution of relatively large instances of the TSP. We will examine two examples of such applications, with the first one being network design [11]. In network design, the primary objective is to determine the most efficient route for data transmission. The vertices in this context can represent various entities such as data centers, network hubs, or devices connected to the network. For certain networks, the task involves connecting thousands of network elements, and any delays caused by longer routes can adversely affect data transmission. Given the continuous and time-sensitive nature of data transmission, the route calculations must be executed swiftly to prevent data loss or degradation in its value.

A second example of such an application can be found in the field of circuit board design [28, 45]. A printed circuit board (PCB) is a flat surface used for mounting electrical components in various electronic devices such as computers, smartphones, televisions, and industrial systems. Efficiently connecting these components on the PCB is crucial for ensuring optimal performance of the device. In this context, the PCB application can be formulated as a TSP instance, where the vertices represent the different components, and the edges represent the interconnections between them. The objective is to minimize the trace length, which refers to the distance traveled by the electrical signal between components. A shorter trace length leads to faster signal transmission and improved device speed. In practice, there are often additional constraints imposed to avoid signal crossovers, which can cause malfunctions in the device [28]. However, in this thesis, we relax these constraints to focus on the broader application of solving large instances of the TSP quickly. With advancements in technology, the number of components on PCBs has significantly increased, with powerful computers and high-end equipment requiring thousands of components [45]. Furthermore, there is a growing trend towards product customization, where users can personalize the technical details of their purchases. This shift from mass production to customized production has resulted in a large number of unique TSP instances that need to be sequentially solved before the production process suffers from significant delays. In this thesis, our goal is to select methods that can efficiently solve these large TSP instances within seconds to prevent production delays. We prioritize methods that can handle even the most complex PCB layouts, accommodating the increasing number of components and the need for fast solution times.

Many heuristics are developed for TSP over the years. Though some heuristics outperform others [21], this often comes with the cost of a higher computational time. In some earlier mentioned applications, a large number of instances needs to be solved within a short period. Thus, it may not be feasible to use exact methods or complex heuristics, as they may take too long to provide a good feasible solution. In such situations, a faster greedy algorithm may be used to obtain a solution. While there exist a number of fast heuristics, their performance may depend on characteristics of instance such as the level of symmetry and density of the graph. In this thesis, a meta-learning model is created using neural networks. Meta-learning is a subfield of machine-learning that focuses on developing algorithms and techniques that enable models to learn to adapt to new tasks or environments more efficiently. In the context of CO problems, meta-learning is commonly used to determine the most suitable methods to apply in different scenarios of the problem. In this thesis, the model predicts the performance of different fast heuristics of

TSP based on features of the instance. Although previous successful studies are conducted using more complex or local search heuristics [24, 22], meta-learning has not been applied to simple heuristics for large instances. Therefore, we study the possibilities for large-scale applications where methods which require large computational time are not suitable. For other CO problems, meta-learning models have shown promising results even for simple heuristics [31], making the usage of meta-learning for heuristic selection a interesting subject to apply to TSP.

This thesis addresses the following research questions:

- How does a meta-learning model using a feed-forward neural network perform when applied to the fast and greedy heuristics for the traveling salesman problem?
- Is the specific choice of machine-learning technique relevant for constructing the right model?
- What are the crucial features that influence heuristic selection in the context of the traveling salesman problem?
- Can artificial data serve as a valid substitute for real data in constructing the meta-learning model?
- What level of tour length reduction can be achieved by utilizing different meta-learning models instead of always selecting the overall best heuristic?

These questions will be investigated through a comprehensive analysis of existing literature, data generation techniques, and numerical experiments. By answering these questions, this research aims to provide valuable insights and contribute to the advancement of knowledge in the field of CO.

The presented thesis is structured in the following manner. Firstly, we offer a brief overview of the significant findings on the traveling salesman problem and the methodologies employed in the literature review. In Chapter 3, we formally introduce TSP through a problem statement. The methodology section (Chapter 4) provides a comprehensive description of the meta-learning model, including details on heuristics, meta-features, and the neural network. In the data section (Chapter 5), we delve into the creation of an artificial dataset. Lastly, we report the findings in Chapter 6 and draw meaningful conclusions in Chapter 7.

2 Literature review

2.1 The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a well-known problem that has a long history dating back to the 19th century. The first known instance of the problem was described in a German manual in 1832, which discussed an optimal tour to visit 45 German cities but did not contain any mathematical formulations [44]. The first mathematical formulation of the problem appeared in the 1930s, when Karl Menger discussed the problem of obtaining a Hamiltonian path for given points on a graph [32]. In the following years, this problem evolved into the formulations that we know today. During this time, the concept of a minimum spanning tree (MST) was also defined. The tree length denotes the minimum length necessary for linking all nodes, thereby serving as a valid lower bound, as a complete tour requires the connection of every node. Robert Clay Prim improved upon earlier work and in 1957 defined an algorithm to obtain the MST in polynomial time [39], while Joseph Kruskal developed a comparable algorithm for the MST a year earlier, which has the same time-complexity $\mathcal{O}(n^2)$ [26]. Other important development include the work of Dantzig, Fulkerson, and Johnson (DFJ) in 1954, where they presented the first widely adopted integer linear programming formulation [10]. This formulation enabled the optimal solution of TSP instances using the Branch and Bound method. Subsequently, in 1960, Miller, Tucker, and Zemlin (MTZ) introduced an alternative formulation [33] that also achieves the optimal solution. The DFJ and MTZ formulations differ in their construction approach. The DFJ formulation employs binary variables for each edge and incorporates specific constraints that eliminate the presence of subtours. In contrast, the MTZ formulation enforces subtour elimination constraints implicitly by employing variables that represent the order of vertex visits. These constraints are incorporated into the formulation through the use of inequalities that ensure the validity of the tour structure. While both formulations yield optimal solutions, they vary in the number of variables and constraints involved. The DFJ formulation generally has fewer variables but often requires more constraints compared to MTZ. As a practical consideration, DFJ is often preferred for smaller instances, while MTZ becomes more suitable for slightly larger ones. However, due to their exact nature, neither formulation is well-suited for applications that demand computational efficient solutions for large-scale instances.

In 1970, Held and Karp introduced a lower bound for the optimal solution of TSP known as the 1-tree bound. This bound is derived by constructing a minimum spanning tree (MST) and adding an efficient edge to create a tour [16]. However, it was discovered

that this method did not always provide the 1-tree that is closest to the optimal solution. To address this limitation, Held and Karp developed a branch-and-bound procedure the following year, which obtained the maximum 1-tree bound. This improved bound is often regarded as one of the most accurate bounds for large instances of TSP. Consequently, when the optimal value is unknown, heuristics for TSP instances are frequently compared to the Held-Karp bound [17] as an alternative.

2.2 Heuristics

While the exact formulations discussed earlier guarantee an optimal solution, they often suffer from high computational time, making them impractical for large-scale problems where the number of cities is significant. Consequently, these formulations are well-suited for obtaining optimal tours in instances involving only a few cities. However, as the problem size increases, their computational demands become restraining. To tackle larger instances, a wide range of heuristics have been developed, offering different levels of complexity and performance. These heuristics provide approximate solutions TSP, meaning that their solution might be suboptimal. They are striking a balance between solution quality and computational efficiency. The simplest heuristic for TSP is the nearest neighbor heuristic (NN). In 1974, Rosenkrantz et al. proved that NN has an approximation ratio of $\Theta(\log(|V|))$[43]. Subsequently, due to technological advances, the heuristic was tested on instances with up to a million cities by Johnson and McGeoch in 1995 [21]. They found that, despite the increase in size, the performance of the heuristic is, on average, 25% higher than the Held-Karp lower bound. Another heuristic, the Greedy heuristic (GR), was also tested in the same paper and showed somewhat better results while only having a minor increase in polynomial complexity compared with NN [21]. GR produces a solution that is, on average, between 15 and 20% higher than the Held-Karp lower bound. Moreover, its performance relative to NN appears to improve with an increase in the size of the instance [21]. Hence, both heuristics serve as viable alternatives when solving large instances of TSP. While they may not guarantee the optimal solution, they provide efficient solutions that are reasonably close to the optimal solution.

In 1964 Clarke and Wright proposed a fast heuristic for the Vehicle Routing Problem (VRP). This problem is a more complex version of TSP as a simple polynomial reduction from VRP to TSP is possible [6]. Clarke and Wright constructed an algorithm that iteratively adds edges to the tour, aiming to minimize the routing cost in each

iteration. This method known as the saving algorithm [6] can easily be adapted to be implemented in TSP. These adaptations were later defined as an insertion heuristic for TSP. Insertion heuristics are a of class greedy algorithms that involve adding edges to a small sub-tour iteratively until a complete tour is obtained. There exist several variants, each with unique conditions for edge addition and different complexities [36]. Examples of these variants include Nearest Insertion, Farthest Insertion, Convex hull Insertion, and Cheapest Insertion. These methods offer an alternative to other greedy heuristics commonly employed in practice. In many cases, multiple greedy heuristics are applied to a given instance, and subsequently, the best tour among them is selected. In 1976, Nicos Christofides developed a greedy algorithm for TSP that outperforms other greedy heuristics [5]. His algorithm, now known as Christofides algorithm, uses a combination of a minimum spanning tree, a minimum weight perfect matching, and short-cutting to obtain a solution. Its worst-case performance is only 1.5 times the optimal solution [5]. In practice, it provides a solution within 10% of the Held-Karp lower bound on average [21]. Although, it is still a polynomial time algorithm ($\mathcal{O}(n^3)$), it has a higher complexity than the previously mentioned greedy heuristics [5]. Hence, this method is commonly applied in situations where obtaining the best possible solution is crucial, but the size of the problem prohibits the use of exact methods. Furthermore, while it can find solutions for instances with up to 1000 cities within minutes, the computational time significantly increases for larger instances, reaching hours or more when the number of cities approaches 10,000 [21]. Therefore, this method serves as a viable alternative when aiming for a good solution to a single instance. To better illustrate this, consider the following scenario in the context of PCBs mentioned earlier. Suppose a factory produces a large quantity of identical PCBs, it is advantageous to compute a solution that is close to optimal. However, this method is not suitable when each PCB is unique, as it would result in significant delays in the factory operations.

Another area of heuristics focuses on improving suboptimal tours. These heuristics, called local search heuristics for TSP, were introduced by G.A. Croes in 1958 [8]. The approach involves switching two vertices in an existing tour to obtain a better solution, and continuing the search until no further improvements are possible [8]. This method, known as 2-opt, was later extended to a larger neighborhood in 1986 by Papadimitriou and Yannakakis, [38], e.g. by swapping three or four vertices at the same time. This resulted in a method which is called k-opt, where k can be any integer larger than 1. In further research, Lin and Kernighan found that varying the value of k at each iteration further enhances the algorithm's performance [29]. The application of the Lin-Kernighan

algorithm, on average, yields a solution that is within 2% of the Held-Karp lower bound [21]. Although, the algorithm has a relative low complexity $\mathcal{O}(n^{2.2})$ [29], large-scale tests by Johnson and McGeoch indicate that the computational time becomes substantial for large instances, as the algorithm requires two sequential computations, deriving an initial path and optimizing it [21]. One advantage of these local search heuristics is their ability to operate within time constraints. Taking our factory example into account, let's consider a scenario where the calculation time per chip is limited to five minutes. In such cases, the heuristic can provide the best solution found within that time frame.

The final category of heuristics that will be briefly discussed are meta-heuristics. Some examples of these methods include Tabu-search, Simulated Annealing, Genetic Algorithms and Ant Colony Optimization [36]. Meta-heuristics are distinct from traditional greedy or local search heuristics in that they employ a different search strategy. Rather than solely focusing on finding local optima, meta-heuristics utilize intelligent techniques to explore a much wider range of solutions. However, as a consequence, this often results high complexity. Therefore, these methods are often not practical for applications which require a solution within seconds even for large scale instances. Although they do not play a direct role in this thesis, it is worth mentioning that the methods used in this study have already been applied to these meta-heuristics for TSP. Furthermore, the subsequent section provides an elaboration on the obtained results of these studies.

2.3 Meta-learning

Meta-learning in CO is a technique that utilizes machine-learning (ML) algorithms to identify patterns and characteristics of problem instances and learn which algorithms are best suited to solve them. In 2008, Smith-Miles was the first to introduce meta-learning in optimization problems, offering a broad description of how machine-learning methods could be used to determine which algorithm is best suited to tackle the instance of the problem, without restricting herself to a single problem or ML method [47]. This framework can be applied to a wide range of specific problems, including TSP [47].
Kanda et al. applied meta-learning to the TSP [23] and tested four ML methods: K-Nearest Neighbors (KNN), Decision Tree (DT), Support Vector Machine (SVM), and Naive Bayes (NB). They used these methods to select meta-heuristics, including Tabu Search (TS), Greedy Randomized Adaptive Search Procedure (GRASP), Simulated Annealing (GA), and Genetic Algorithms (GA). However, the team emphasizes that the used methods are not restricted to these complex meta-heuristics [23]. Using the frame-

work introduced by Smith-Miles [47], they tested their methods for instances up to 100 vertices and included twelve simple features to identify the instance. The results of their study indicated that the selection of a particular machine-learning method did not have a substantial impact on the performance of the meta-learning model. Specifically, when testing the model on real data, the accuracy levels achieved by KNN, DT, and SVM were found to be comparable, with only NB exhibiting slightly lower performance. Meaning that all machine-learning methods have comparable predictive power in predicting the best heuristic. However, the performance of the models was not impressive, leading researchers to believe that applying different techniques and incorporating better features could potentially improve their performance [23]. The improved models were published in 2016 [24]. Here, Kanda et al. improved upon their earlier work by including Ant Colony Optimization (ACO), using a technique called label ranking to train the model to provide a performance ranking of the heuristics and creating more meta-features [24]. This time KNN, DT and a multilayer perceptor neural network (MLP) were used. Besides the traditional TSP case, they trained and tested their model on instances of the TSP that were asymmetric, weakly connected, or both. They also expanded the list of features to include more characteristics of the graph structure, such as clustering and network vulnerability [24]. The experiments were still limited to instances up to 100 vertices, but the results showed that the choice of ML model is not the bottleneck, as all models seemed successful. Notably, the estimated correlation coefficients for all models were equal to 0.93, signifying a large improvement compared to earlier work [24]. The observed improvement, in comparison to the previous study conducted by Kanda et al. [23] which relied on heuristic classification, can be partly attributed to the adoption of a new model configuration. In this study, regression on label ranking was employed to predict the performance of the heuristics. As a result, a more comprehensive understanding is attained as it obtains not only the identification of the top-performing heuristic but also the ranking of subsequent alternatives. Finally, Smith-Miles et al. applied a meta-learning model to TSP for variants of the Lin-Kernighan heuristic [46]. However, their goal was not to obtain the best model but rather identify characteristics which makes an instance difficult. According to their results, important characteristics include: the variance of the edge's length, the fraction of unique distances and the number of clusters and outliers [46]. In conclusion, previous research has obtained important findings to the development of a meta-learning model for TSP. Valuable insights from earlier studies concerning machine-learning techniques, feature selection, and label ranking are implemented into this thesis.

Mao et al. conducted a heuristic selection study on the variable sized bin packing problem using a neural network and simple heuristics [31]. They used only simple statistical features of the items and boxes and were able to obtain promising results. Notably, their models achieved over 70% accuracy, providing evidence that meta-learning can be effective for such simple heuristics [31]. To the best of my knowledge, meta-learning has not been applied to TSP using less complex heuristics and on very large instances. This thesis contributes to closing this gap.

2.4 Neural Networks

The origin of the Neural Network (NNet) as a machine-learning model dates back to the 1950's when Frank Rosenblatt introduced the perceptron [42]. An electronic device which has the ability to learn and recognize simple patterns. Years later, in the 80's neural networks really started to develop as the models we know it today with the introduction of the Backpropagation algorithm by Paul Werbos [51]. This algorithm made it possible to compute more complex gradients and therefore to incorporate multiple layers in a network. Another important development followed by Hornik et. al. in 1990 with the proof of the Universal Approximation Theorem [20]. The theorem states that a single layer feedforward neural network with a large enough finite number of neurons can approximate any continuous function on a compact set to any desired degree of accuracy [20]. This means that in principle, neural networks have the potential to handle various levels of complexity in the functions they approximate. However, it's important to note that as the complexity of the function increases, the complexity of the network itself also grows. Therefore, expecting a neural network to effortlessly solve extremely challenging problems would result in an excessively complex network. Instead, we can focus on finding neural network architectures that are suitable for specific problem classes or achieve satisfactory performance, which may surpass certain heuristic approaches. While neural networks are not a magical solution for all hard problems, they offer the potential for effective problem-solving within certain problem domains or achieving outcomes that are "good enough" for practical purposes. The objective of this thesis is to develop a neural network with an appropriate level of complexity that enables the identification of key characteristics in various instances of the TSP, such that it's able to recommend the best heuristics to employ.

Regarding CO, neural networks can be used in various ways. Besides in meta-learning [24], neural networks are often used as a solving technique. For instance, Mansor et al. [30] used neural networks to tackle the Satisfiability problem (SAT), which involves finding a truth assignment for variables in a formula to satisfy the total expression evaluates to true. Their study incorporates a neural network to improve the computational efficiency of complex heuristics, resulting in promising improvements to the methods. This suggests that neural networks can serve as valuable additions to existing heuristics. Furthermore, Hopfield and Tank [19] introduce the Hopfield network, an unsupervised learning neural network capable of solving various CO problems, including TSP. The network uses different neuron states and an energy function and iterative dynamics between these. Hopfield networks can perform tasks such as pattern recognition and optimization. These networks are known for their ability to rapidly compute solutions to optimization problems based on input information. However, the scalability of Hopfield networks poses limitations, as they encounter memory and computational issues when handling larger problem instances, rendering them impractical for the specific applications addressed in this thesis. Nonetheless, these examples demonstrate that neural networks can be effectively applied in different learning techniques, including supervised learning through meta-learning and unsupervised learning through Hopfield networks.

3 Problem statement

The classical Euclidean Traveling Salesman Problem (TSP) is formulated as follows. We are given a graph $G = (V, E)$, where V represents a set of vertices, E represents the set of edges, and each edge $\{i, j\} \in E$ has a cost $c_{i,j}$. We need to find the cheapest Hamiltonian cycle. Here $c_{i,j}$ represents the distance of moving from v_i to v_j ($\{i, j\} \in e$) and can be calculated using the Euclidean formula: $c_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. This formulation ensures that the distances are symmetric, i.e., $c_{i,j} = c_{j,i}$. Moreover, Euclidean distances ensure that the triangle inequality holds, which means that it is always faster to move directly from one city to another than via a third city i.e. $c_{x,z} < c_{x,y} + c_{y,z}$. Finally, G is complete, which means that there exist an edge between every pair of vertices $\{i, j\} \in e, \forall i, j \in V$.

While there exist versions of TSP for which the graph is incomplete, the distances are non-Euclidean or asymmetric, and the triangle inequality may be relaxed, this thesis is limited to the classical case. There exist a number of exact ILP formulations to solve this version of TSP. The Dantzig–Fulkerson–Johnson (DFJ) formulation is as follows [9].

$$x_{i,j} = \begin{cases} 1 & : \text{if edge } e_{i,j} \text{ is used in the tour} \\ 0 & : \text{otherwise} \end{cases}$$

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{i,j} x_{i,j} \\
\text{subject to} \quad & \sum_{i=1, i\neq j}^{n} x_{i,j} = 1, && \forall j \in \{1, .., n\} && (1) \\
& \sum_{j=1, j\neq i}^{n} x_{i,j} = 1, && \forall i \in \{1, .., n\} && (2) \\
& \sum_{i \in S} \sum_{j \in S, j\neq i} x_{i,j} \leq |S| - 1, && \forall S \subset \{1, .., n\},\ |S| > 1 && (3) \\
& x_{i,j} \in \{0, 1\}, && \forall i, j \in \{1, .., n\} && (4)
\end{aligned}$$

The proposed model operates in the following manner. First, a binary choice variable is established for each edge present in the graph, and set to 1 if the corresponding edge is included in the optimal solution of the problem. The objective function aims to minimize the total length of the tour by disregarding the distances of edges that are not part of

the tour. The first and second constraints guarantee that each vertex has precisely one incoming and outgoing edge, respectively. The third constraint eliminates the existence of subtours by ensuring that the number of edges in any given subtour is never greater than the number of vertices in that same subtour minus one. As such, only the complete tour is permitted. Finally, the binary nature of the choice variables is enforced to ensure the feasibility of the model.

Given the aforementioned assumptions, it has been demonstrated that the Euclidian Traveling Salesman Problem belongs to the class of problems that are considered NP-Complete, as stated in [38]. This indicates that there does not exist a known algorithm that can solve the TSP in polynomial time. The factorial of the number of cities indicates the number of feasible tours, thereby rendering the computation of every tour impossible for large instances of the problem. For example, while an instance with four cities has only 24 possible tours, an instance with ten cities results in more than 3.5 million potential solutions. This highlights the infeasibility of a computer performing exhaustive search of all tours for larger instances of the TSP.

4 Methodology

For constructing the meta-learning model, the general framework of Smith-Miles is used, which consists of two phases [47]. In the first phase, problem instances of TSP are obtained from the problem space. This means that we apply different heuristics to a lot of instances of TSP. In our case the heuristics are: Nearest Neighbor (NN), Greedy Algorithm (GR), Nearest Insertion (NI) and Farthest Insertion (FI), their performance on these instances is what is considered the target values. The target values represent the underlying concept that the model seeks to explain. Afterwards, different characteristics of the instance are extracted into what are called the meta-features. Together with the target values, these are considered the meta-data. The second phase exist of constructing the meta-model. In this phase, a machine-learning technique is used to model the obtained meta-data to a predictive model as a method of supervised learning. The choice of ML technique seems to be irrelevant as different techniques have all been proven very successful [22, 24, 31]. In this thesis, a multilayer perceptor neural network (MLP) is chosen as its main method, since this technique has been proven successful with both more difficult heuristics [24] and for simple heuristics in other problems [31]. Next, we will provide a more detailed presentation of the heuristics used in this study. Section 4.3 will describe the neural network, followed by the presentation of two alternative machine-learning techniques in Section 4.4.

4.1 The Heuristics

For TSP a broad choice of different heuristics is available, from simple and fast heuristics to complex meta-heuristics all with there own performance. The meta-learning model described in this thesis is designed for applications which require large number of big instances to be solved within seconds. Such speed requirements makes it not suitable to check multiple fast heuristics and pick the best one. As a consequence, local search algorithm also seems not useful because they require an initial solution as a starting point, which leaves them no time to optimize in these applications. Practically, we included heuristics with a time complexity of at most $\mathcal{O}(n^2 \log_2(n))$. Therefore, also excluding strong greedy algorithms such as Christofides [5], which outperforms some of the heuristics used in the model [21]. Because of it higher complexity, Christofides is not able to solve instances in seconds when the number of cities becomes larger than a few hundred [21].

Nearest Neighbor (NN)

The Nearest Neighbor heuristic is the most straightforward heuristic for TSP. Starting with a random node, the heuristic adds the nearest node of that node to the tour. In the next steps iterative it adds the nearest unvisited node of the node which was added last until all nodes are in the tour. Finally, the tour is finished by returning to the first node [36]. The heuristic has a time complexity of $\mathcal{O}(n^2)$, meaning that even for large instances the heuristic provides a solution relatively fast. This has also been proven in experimental settings [21]. The provided solution is on average within, 25% of the Held-Karp lower bound [21]. Note that the provided solution depends on the starting node, hence when it is chosen randomly the tour has a stochastic element. However, in practice often the node with the lowest index is chosen as the starting node, which is also used in this thesis. A logical improvement of this algorithm is to pick the shortest tour out of all n starting nodes. However, this would result in a time complexity of $\mathcal{O}(n^3)$, as it needs to run NN for all n starting nodes. This makes it less suitable for the speed-demanding applications where this meta-learning model is applicable. Furthermore, it would be unfair to compare it with NN as the meta-learning model consistently performs at least as good as NN. The pseudo-code of NN is as follows.

Algorithm 1 Nearest Neighbor

Input: n nodes, m edges, All edges containing distance from one node to another

1. Add lowest index node as starting city to tour
2. Add unvisited node with smallest distance to last node to tour

if Any unvisted node left **then**

3. Return to step 2.

else

4. Finish the tour by adding the starting city to the tour

end if

Greedy Algorithm (GR)

The Greedy Algorithm is the second algorithm which is considered in this thesis. This heuristic focuses on the edges instead of the nodes. First all edges are sorted and the shortest edge is picked as a starting point. Afterwards, edges are added to the tour as long as adding the edge does not create a node with a degree higher than two or a subtour [36]. Compared with NN, the Greedy Algorithm has a little higher polynomial complexity $\mathcal{O}(n^2 \log_2(n))$. However, in practice for instances with up a few thousand nodes it provides a solution in seconds [21], making it suitable for this model. It has

been proven that this heuristic obtains a tour that has at most $0.5(\log_2(n)+1)$ times the length of the optimal tour [37]. Empirically, the performance of this algorithm is slightly better than NN with on average between 15 and 20 % higher than the Held-Karp lower bound [21]. However, it is not consistently dominant, meaning that there are instances in which NN obtains shorter routes than GR. GR has the following pseudo-code.

Algorithm 2 Greedy algorithm

Input: n nodes, m edges, All edges containing distance from one node to another

1. **SORT** all edges in non-decreasing order of distance
2. Add shortest edge as starting edge to tour

for `All remaining ordered edges` **do**
  if Number of edges in tour $\leq n$ **then**

    if Adding edge does not create a node with a degree higher than 2 **AND** Adding edge does not create a subtour **then**
      3. Add edge to tour.
    end if
  else
    4. **end for**

Nearest Insertion (NI) & Farthest Insertion (FI)

The final heuristics considered for the meta-learning model are two insertion heuristics. The main idea behind these heuristics is that the nodes are inserted iteratively based on some criterion and each iteration puts the node on the optimal place in the tour. More specifically, this means the heuristic starts with a subtour of only two nodes and adds each node at a place in tour which minimizes the length of the new subtour. Again, the starting node is chosen to be node with the lowest index. Two different insertion heuristics are considered. Nearest Insertion (NI) picks the node with minimal distance from the previous node to be added next, and Farthest Insertion (FI) picks the node with maximal distance from the previous node. Both heuristics have time complexity $\mathcal{O}(n^2)$ [36], however intuitively their performance depends on the structure of the graph. NI would prefer dense graphs as the order in which the nodes are added is less relevant in this case. In contrast, FI works better in large less dense graphs because in these instances, the length of large edges plays an important role. It is also worth mentioning that there exist versions of insertion heuristics which picks the node based on minimal or maximum distance to the current subtour instead of previous node. Such insertion heuristics can have better performance as a consequence. However, using these

criterion's to determine which node to select would result in an increase of complexity making the corresponding approaches less suitable for this thesis. The pseudo-code for Nearest Insertion and Farthest Insertion is provided below.

Algorithm 3 Nearest/Farthest Insertion algorithm

Input: n nodes, m edges, All edges containing distance from one node to another

1. Add lowest index node as starting city to tour
2. Find unvisited node with smallest/largest distance to last node to tour ▷ smallest for NI/ largest for FI
3. Insert node at place which minimizes the cost of the total subtour

if Any unvisted node left **then**

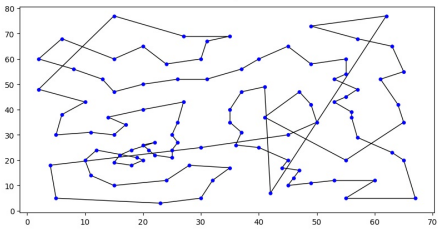
4. Return to step 2.

else

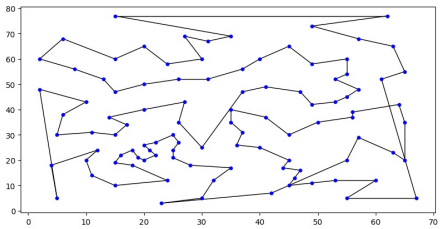
5. Finish the tour by adding the starting city to the tour

In order to illustrate the practical performance of the heuristics, Figure 1 presents the paths generated by the four heuristics on the eil101 instance. Upon examination, it becomes clear that the NN heuristic exhibits a significant distance between the first and last city in the tour, leading to the use of an inefficiently long edge spanning the entire instance. This behavior is typically for NN, making it most suitable for dense graphs without outliers, because here the final edge of the tour will lie close to the first one. Moreover, GR appears to struggle as it revisits certain cities, resulting in inefficiencies. It is important to note that GR does not actually revisit cities, but rather encounters cities that lie on the path between two others. The insertion heuristics demonstrate better performance when applied to instances exhibiting a circular pattern. This advantage stems from their nature of constructing an initial circular tour and subsequently expanding it by iteratively searching for the most efficient subtour. The figure reveals traces of a circular pattern, supporting the suitability of these heuristics for such instances. As a result, FI demonstrates a promising solution for this particular instance, as it reveals few inefficient lines. The optimal solution for this instance stands at 629 [18], leaving room for further improvement. However, considering its efficiency, FI offers a good solution. It should be noted that for other instances, FI may produce worse solutions compared to the other fast heuristics.

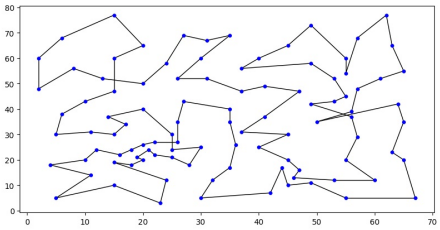
Figure 1: Heuristic Performances on eil101.tsp



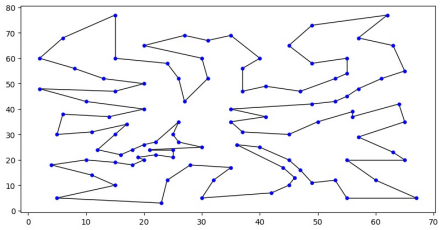
NN, length = 825



GR, length = 795



NI, length = 767



FI, length = 712

Table 1 provides more results of the heuristics on real life instances and their optimal solution. It is crucial to acknowledge that a few instances have been measured using distance metrics other than Euclidean distances in their respective applications. Nevertheless, in this thesis, we consistently use Euclidean distance for all instances. The rationale behind this choice is to observe how the meta-learning model performs on specific graph structures, rather than focusing on solving individual instances. Consequently, for these instances, the optimal solution remains unknown and is denoted in the table by "-". The instances denoted in grey represent instanced used in constructing the training set, further elaboration is provided in Section 5.2.

Figure 2 presents the calculation times of the four heuristics on real-life instances. It should be noted that these times are heavily dependent on the computational power of the computer used. However, their relative calculation times offer valuable insights. Notably, NN consistently achieves the quickest solutions among the four heuristics, with the longest calculation time not exceeding a few seconds. Both NI and FI can compute all tours within 20 seconds. On the other hand, GR exhibits outliers in computation time as the number of vertices approaches 5000, owing to the slightly more complex nature of the heuristic. Based on these findings, it is chosen to construct a dataset with at most 3000 vertices, as all heuristics can efficiently produce solutions within a few seconds.

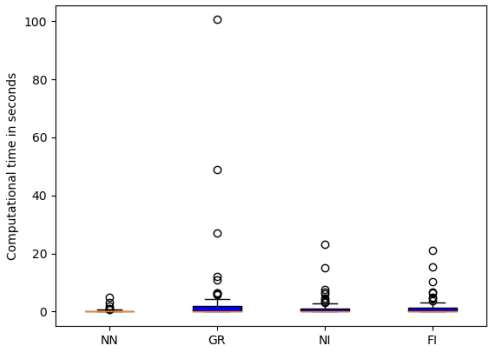


Figure 2: Calculation times

Table 1: Heuristic Performance and optimal solution TSPLIB

Instance	Nearest Neighbor	Greedy Algo	Nearest Insertion	Farthest Insertion	Optimal Solution
a280.tsp	3148	2952	3107	2958	2579
ali535.tsp	2671	2403	2466	2340	-
att532.tsp	112099	105864	107342	103000	27686
bier127.tsp	135751	141187	141672	132520	118282
ch130.tsp	7575	7844	7372	6920	6110
ch150.tsp	8194	7704	8066	7586	6528
d1291.tsp	59941	60266	62871	62117	50801
d1655.tsp	74893	72263	75669	75604	62128
d198.tsp	18620	19415	17560	16854	15780
d2103.tsp	87468	91363	87100	95527	80450
d493.tsp	43646	40838	42834	42397	35002
d657.tsp	62176	56568	61682	58658	48912
dsj1000.tsp	24630960	21703941	23231998	22737713	18659688
eil101.tsp	825	794	762	712	629
fl1400.tsp	26971	23587	24367	23196	20127
fl1577.tsp	27940	25707	25673	26313	22249
fl3795.tsp	34225	32781	32596	36062	28772
fl417.tsp	15114	13360	14260	13130	11861
fnl4461.tsp	227156	210508	224369	235348	182566
gil262.tsp	3241	2686	2977	2819	2378
gr137.tsp	1022	844	850	789	-
gr202.tsp	619	595	579	541	-
gr229.tsp	2014	2068	1888	1927	-
gr431.tsp	2516	2553	2287	2272	-
gr666.tsp	4110	3642	3769	3765	-
kroA100.tsp	26856	24084	26377	24597	21282
kroA150.tsp	33609	31778	32620	29994	26524
kroA200.tsp	35798	34535	37081	33304	29368
kroB100.tsp	29155	25815	26542	24183	22141
kroB150.tsp	32825	31422	34079	29987	26130
kroB200.tsp	36981	35984	36252	35704	29437
kroC100.tsp	26327	23302	26700	22648	20749
kroD100.tsp	26950	24267	26311	23069	21294
kroE100.tsp	27587	24739	27290	23485	22068
lin105.tsp	20362	16771	18829	15849	14379
lin318.tsp	54033	49910	52603	52419	42029
nrw1379.tsp	70015	65643	68756	71084	56638
p654.tsp	43411	40264	43293	38009	34643
pcb3038.tsp	175573	164445	168288	175760	137694
pcb1173.tsp	70277	67244	71012	73367	56892
pcb442.tsp	61984	61073	59390	61622	50778
pr1002.tsp	315596	308819	324586	323479	259045
pr107.tsp	46678	47542	53693	46867	44303
pr124.tsp	69299	64997	67928	63377	59030
pr136.tsp	120777	115433	113927	105554	96772
pr144.tsp	61650	65845	73120	60922	58537
pr152.tsp	85702	85272	87587	77001	73682
pr226.tsp	94685	97599	89793	82157	80369
pr2392.tsp	461207	453345	483379	485481	378032
pr264.tsp	58022	54974	56834	52429	49135
pr299.tsp	59899	63334	61317	53489	48191
pr439.tsp	131282	128749	133413	129040	107217
rat195.tsp	2761	2745	3033	2839	2323
rat575.tsp	8449	7733	8437	8347	6773
rat783.tsp	11255	10534	10966	11138	8806
rd100.tsp	9941	9228	9897	8818	7910
rd400.tsp	19168	17872	18749	18243	15281
rl1304.tsp	339797	285113	321354	323617	252948
rl1323.tsp	332094	307408	351775	347172	270199
tsp225.tsp	4828	4618	4569	4604	3916
u1060.tsp	281635	277843	282440	277640	224094
u1432.tsp	188815	182137	175645	186841	152970
u159.tsp	54669	49582	51869	52981	42080
u1817.tsp	71103	68517	70375	74192	57201
u2152.tsp	80179	75839	78725	81735	64253
u2319.tsp	278783	260570	262239	274964	234256
u574.tsp	46881	44768	45717	45907	36905
u724.tsp	55223	49119	52812	52344	41910
vm1084.tsp	301469	286182	282531	294184	239297
vm1748.tsp	408089	397390	412084	422840	336556

4.2 Performance evaluation

The quality of the model's predicted ranking should be assessed by comparing it with the actual ranking through measures of ranking accuracy [24]. Here, the label corresponds to the performance of the heuristics. For more detailed information regarding the label, please refer to the subsequent section. For this model, Spearman's correlation coefficient (ρ) is chosen as an evaluation measure [48]. For each observation i, ρ_i can be obtained by:

$$\rho_i = 1 - \frac{6 \sum_{l=1}^{L} (\hat{r}_{i,l} - r_{i,l})^2}{n(n^2 - 1)}. \tag{5}$$

Here L represents the number of labels, which is four in our case, $\hat{r}_{i,l}$ the predicted rank for label l and $r_{i,l}$ its true rank. Hence, a coefficient of 1 means a perfect prediction, whereas a coefficient of -1 would mean an inverted order. Finally, ρ can be obtained by averaging over all N observations:

$$\rho = \frac{1}{N} \sum_{i=1}^{N} \rho_i. \tag{6}$$

Confidence intervals for the correlation coefficient ρ can be computed by applying the Fisher transformation, which is *atanh* in this case [12]. This transformation causes the sampling distribution to converge to a normal distribution [12]. Consequently, a confidence interval with a confidence level of $(1-\alpha)\%$ can be derived by multiplying the $(1-\alpha)$-quantile of the standard normal distribution with an estimate for the standard error. The standard error estimate can be approximated by $\frac{1}{\sqrt{N-3}}$ [12]. As a result, the 95% confidence interval can be expressed as follows.

$$CI = tanh[atanh(\rho) \pm 1.96/\sqrt{N-3}]. \tag{7}$$

4.3 Neural Network

Although previous studies have demonstrated that various ML techniques produce effective meta-learning models for TSP [23, 24, 46], we use a neural network (NNet) as its main method of choice primarily. The reason is based on the positive outcomes achieved by a meta-learning model that used neural networks and simple heuristics in the bin packing problem [31] and in TSP using more complex heuristics [24]. Furthermore, the current study aims to enhance the performance of the previously designed networks.

4.3.1 Neural Network Architecture

The network is a multilayer perceptron (MLP), used to predict the performance of the heuristics. Although neural networks are also applicable for unsupervised learning [35], in this setting, it is used as a supervised learning model, specifically for providing a ranking of heuristics from best to worst on a given instance. To accomplish this, separate neural network models are designed for each heuristic, and their performance is predicted using a technique which is called label ranking. This technique maps an instance of TSP to a finite set of method labels $L = \{\theta_1, .., \theta_L\}$, where if $\theta_i < \theta_j$ it means that for this instance, method θ_i is preferred to be used over θ_j [50]. For this model, a label is assigned to each heuristic, and the goal is to predict the label for each of the four separate models. Because the regression will result in continuous predictions, the predictions are ordered afterwards to obtain the predicted integer ranks. The advantage over a classification model with the best performing method as the target variable is that the resulting outcome includes the order in which the methods are preferred [24]. For example, if the application allows, the user could test the two lowest ranked methods and choose the best one. Furthermore, this approach has less uncertainty than predicting the exact performance of different heuristics on an instance and building a ranking from that. Moreover, the utilization of label-ranking in the models has demonstrated noteworthy enhancements in accuracy when compared to traditional classification methods [22, 24]. According to the Universal Approximation Theorem [20], a complex enough MLP should be able to approximate the label ranking with high accuracy.

The network architecture consists of a feed-forward four-layer perceptron, comprising an input layer for each of the utilized meta-features, two hidden layers, and an output layer with a single neuron, which predicts the label rank. Therefore, having a more complex network than previous studies [24], which employed a single hidden layer, while maintaining computational feasibility. Other structures have been attempted, but they were found to be inferior. For more context, please refer to Section 6.2. The activation function used in the output layer is an adapted Tanh activation function.

$$\tilde{tanh}(x) = \frac{3}{2}(tanh(x) + 1) + 1. \tag{8}$$

This modification guarantees that the predicted rank falls within the range of 1 to 4. It is achieved by adjusting the standard Tanh activation function, which is originally bounded between -1 and 1. Furthermore, it has the property of monotonicity, indicating that there exists a one-to-one correspondence between the input x and the output. The

number of neurons in each hidden layer is equal to 8 times the number of input features, which is the same factor used in the bin packing problem model [31]. For each neuron in the hidden layer the Relu function is used given as [35]:

$$Relu(x) = max\{0, x\}. \tag{9}$$

This activation function has the advantage of faster training compared to other popular activation functions [25]. Additionally, it addresses the issue of vanishing gradients [35], which occurs when most gradients in a neural network are very small. The backpropagation algorithm [51] is used to train the network, starting from the output layer and computing the gradients backwards in the network. As mentioned before ρ is used as the accuracy measure. However, it's unwise to implement it directly as the loss function due to the construction of the four individual models. When trained individually, ρ cannot be used as it would require a simultaneous training process. Therefore, the Mean Squared Error (MSE) is chosen as the loss function for all four models, as it effectively addresses this issue. Nevertheless, ρ is however used in validating and testing the models.

If we want to obtain the number of parameters in the network, we calculate these as follows. Given that each model uses 21 meta-features (detailed in the subsequent section), we obtain 168 neurons in the first hidden layer, resulting in 168×2 (input neuron + bias parameter) = 336 parameters. The second layer adds 168×169 (neurons first layer + bias parameter) = 28392 additional parameters. Finally, the output layer contributes a weight for each neuron in the second hidden layer plus a bias parameter resulting in 169 extra parameters. Therefore, in total we obtain: $336 + 28392 + 169 = 28,897$ parameters in each of the four networks.

4.3.2 Neural Network Training and validation

Like several other machine-learning techniques, the neural network is susceptible to overfitting, particularly when the number of parameters is taken into account. To mitigate this issue, a validation process employing k-fold cross-validation is implemented to determine the optimal hyperparameters. In this process, the training data is randomly divided into k folds. The model is then trained on all folds except one, which serves as the validation set for evaluating the model's performance using the metric ρ. This procedure is repeated until all folds have been utilized as the validation data. Subsequently,

the performance of the hyperparameters is assessed by computing the average of the k values of ρ. The selection of an appropriate value for k is crucial to balance overfitting and computational feasibility. In this thesis, k is set to 5 to strike a suitable compromise, resulting in around 2,400 observations per fold. From the complete trainingset 70 instances are randomly withhold and included in the testset.

Table 2: Hyperparameter grid NNet

Ridge parameter	0	0.0001	0.001	0.01
Dropout rate	0	0.1	0.3	0.5
Batch size	12	24	36	
Learning rate	0.01	0.001	0.0001	

Table 2 presents the hyperparameter grid employed for the 5-fold cross-validation process. The first parameter being tuned is the Ridge regression parameter, which prevents overfitting by penalizing the sum of squared weights in each layer, by multiplying it by this parameter [35]. The presence of this penalty term discourages the presence of excessively large coefficients, promoting a smoother and more stable solution. Consequently, when encountering changes in the data, the model's reaction is more tempered, thereby preventing overfitting. The chosen options for this parameter are based on the need for relatively small values, given the predicted a true ranks are both bounded between 1 and 4. Consequently, the model is expected to encounter only minimal discrepancies between the predicted and true ranks due to the limited range of possible values. The second hyperparameter also addresses overfitting and utilizes a technique called Dropout [49]. Dropout randomly removes connections in the network during training. The dropout rate determines the probability of each neuron being dropped from the model during each iteration of the training process. A dropout rate of 0.5, for example, results in a model with approximately half the original number of neurons in each iteration. This reduces reliance on specific neurons, thereby minimizing overfitting. The selected values for this hyperparameter compare the effect of no dropout with different dropout rates. The third tuning parameter is the batch size used in each iteration of gradient descent optimization. After visual inspection on a subsample of the dataset, it was decided to keep the number of epochs fixed at 500. The selected batch sizes are chosen to strike a balance between ensuring computational feasibility during training while avoiding excessively small batch sizes. Lastly, the learning rate in gradient descent is optimized. The selected values for this hyperparameter lie within a specific range to avoid issues with the optimization process. Values lower than 0.0001 lead to early stabilization of

the loss function after only a few initial iterations, suggesting that the gradient becomes trapped in local optima. Moreover, higher values exceeding 0.01 exhibit a similar pattern, indicating that the gradient descent algorithm tends to skip over optima for such magnitudes. Concluding that both excessively low and excessively high learning rates result in rapid convergence towards suboptimal local minima, thereby necessitating the selection of a rate that strikes a balance and avoids such patterns. The decision to choose values within this range was informed by visual examination of the training process on a smaller subset of the data. Two different benchmark models will be introduced, and each of them will undergo a similar k-fold cross-validation process.

4.4 Benchmark models

In order to determine of a complex neural network model is the right choice for this meta-learning model, two additional machine-learning methods are applied which are less complex. Although, literature has shown that for other meta-learning models the choice of the exact machine-learning approach seems insignificant [22, 24, 31], it is good to check weather this holds for fast heuristics on large instances as well.

The first alternative method which is employed is K-Nearest-Neighbour (KNN). Not to be confused with the heuristic mentioned earlier, this machine-learning method makes predictions by averaging the target values on the K observations whose features are closest to that of the prediction. More specific, in this meta-learning model the predicted rank for each heuristic is determined by the K observations whose meta-features lie closest to the observation. Representing the same structure as the neural-network mainly, employing a different regression model for each heuristic using label ranking. The only hyperparameter which needs to be tuned is the number of neighbours K. Often the root of the size of the trainset is used, therefore this is choice as median of the grid. The validated parameters are: $K \in \{9, 29, 49, 69, 89, 109, 129, 149, 169, 189, 209\}$.

The second alternative model is a Decision Tree (DT). This method partitions the trainset into different nodes, each representing a specific splitting criterion based on one of the meta-features. For each heuristic a different tree is designed. Subsequently, the predicted rank can be obtained by examining the meta-features of the predicted observation, starting from the top of each tree and traversing down the tree until the final prediction is obtained. A notable advantage of this approach is that it offers valuable insights into the significance of particular features by visualizing the decision tree. By observing the

tree's structure, one can understand which features are considered more influential in the prediction process. To address potential overfitting issues, hyperparameters, such as the maximum depth of the trees and the number of features considered at each leaf, are tuned. The aim is to strike a balance between model complexity and generalization capability. By controlling the depth of the trees and limiting the number of features considered at each leaf, overfitting can be mitigated. Table 3 presents the grid search that is conducted to find the optimal model. A depth of 40 signifies no constraints on the tree's depth, while the "Root" measure acts as a preventative mechanism against overfitting. It achieves this by restricting the evaluation of a subset of possible meta-features, specifically by randomly selecting only the square root of all available features for consideration at each split. Please note that similar to the neural network, the hyperparameters for DT and KNN models are also validated simultaneously for all four models. In other words, the same hyperparameter values are used across all models during the validation process. This decision is taken to ensure computational feasibility and reduce the validation time. However, it is important to acknowledge that a sequential approach, in which hyperparameters are tuned independently for each model, may potentially yield more optimal results. The parameters are selected through cross-validation, as detailed in Section 4.3.2. The chosen parameters (K, Max Depth, and Features) are presented in Section 6.2.

Table 3: Decision Tree Grid

Maximum Depth	4	8	12	16	20	24	28	32	36	40
Features	All	Root								

4.5 Meta-features

An important aspect of the meta-learning model involves the translation of problem characteristics into meta-features [47]. For this thesis, the selection of meta-features is a combination of prior research studies [22, 24, 46] and original ideas. The idea of this setup is to include sufficient features such that the characteristics are identified, but still limit the complexity of the model. The identified characteristics must be computable within a few seconds, even for large problem instances. Furthermore, the selected features are categorized into three distinct groups.

The first group pertains to the statistical attributes of edges and vertices, including the number of vertices and edges, average edge and vertex cost, minimum and maximum costs, and other statistical properties. Table 4 presents a comprehensive list of these features. Within these features, the term "vertex cost" (C_i^V) denotes the average cost associated with all edges linked to vertex i.

Table 4: Vertex Edge meta-features

Notation	**Formula**	**Description**
V_{number}	length(V)	number of vertices (n)
C_{min}^V	$\min\{\mathrm{C}_1^V, ..., C_n^V\}$	Lowest vertex cost
C_{max}^V	$\max\{\mathrm{C}_1^V, ..., C_n^V\}$	Highest vertex cost
C_{avg}^V	$\frac{1}{n}\sum_{i=1}^n C_i^V$	Average vertex cost
C_{sd}^V	$\sqrt{\frac{1}{n-1}\sum_{i=1}^n (C_i^V - {}_{avg}^V)^2}$	Standard deviation vertex cost
C_{med}^V	$\mathrm{median}\{\mathrm{C}_1^V, ..., C_n^V\}$	Average vertex cost
E_{number}	length(E)	number of edges (m)
C_{min}^E	$\min\{\mathrm{C}_1^E, ..., C_n^E\}$	Lowest edge cost
C_{max}^E	$\max\{\mathrm{C}_1^E, ..., C_n^E\}$	Highest edge cost
C_{avg}^E	$\frac{1}{n}\sum_{i=1}^n C_i^E$	Average edge cost
C_{sd}^E	$\sqrt{\frac{1}{n-1}\sum_{i=1}^n (C_i^E - {}^C{}_{avg}^E)^2}$	Standard deviation edge cost
C_{med}^E	$\mathrm{median}\{\mathrm{C}_1^E, ..., C_n^E\}$	Median edge cost

The second group focuses on the network's structure and spatial distribution. The first thing which is considered are the properties of the eigenvalues of the distance matrix. The idea behind this is that when the instance contains a certain pattern or symmetry, the eigenvalues of its distance matrix will be very similar. However, calculating the eigenvalues for large matrices, which is often the case in most TSP instances, can be computationally expensive. To address this, the model will utilize the value of the condition number of the distance matrix as a feature. This can be computed relatively

quickly using the Lanczos algorithm [27]. The condition number is defined as:

$$CondNum = \frac{|\lambda_{max}|}{|\lambda_{min}|}. \tag{10}$$

In which λ_{max} represents the largest and λ_{min} the minimum eigenvalue of the distance matrix.

Regarding density, the surface of the rectangle in which the points of the instances are located is considered. If the surface of the rectangle is relatively small compared to the number of vertices and the median edge length, it indicates a dense graph. The rectangle can be calculated as follows:

$$Rect = (X_{max} - X_{min})(Y_{max} - Y_{min}). \tag{11}$$

Here, X_{max} and X_{min} represents the maximum and minimum X-coordinates and Y_{max} and Y_{min} represents the maximum and minimum Y-coordinates.

Density is also captured by analyzing the centroid of the instance. The feature *AvgCent* represents the average Euclidean distance between all points of the instance and the centroid. Symmetry is further addressed using the feature *DistinctRatio*, which examines the uniqueness of entries in the distance matrix. A low *DistinctRatio* indicates that many edges are similar. This is often the case in drilling problems that exhibit patterns on a grid. The *DistinctRatio* can be calculated as follows:

$$DistinctRatio = \frac{\text{number of unique non-diagonal entries}}{\text{Total number of non-diagonal entries}}. \tag{12}$$

In addition to symmetry and density, clustering is a significant characteristic of graph structures [46]. The determination of clusters and outliers plays a crucial role in selecting the right heuristic. For example, when dealing with a large number of outliers, NN may encounter difficulties. This is because it is highly likely that the final edges will primarily connect to outliers, resulting in a significant increase in the tour's length. To find the clusters, the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm is applied [15]. The reason behind this choice is twofold: the algorithm exhibits essential computational performance with a time complexity of ($\mathcal{O}(n^2)$) [15], making it suitable for the meta-learning model at hand, and it is capable of identifying more complex cluster shapes compared to alternative methods. The DBSCAN algorithm

operates as follows: given a distance threshold ϵ and a minimum number of points m, it selects all points that are within a distance of ϵ from at least m other points, referred to as core points. Clusters are formed based on these core points. Non-core points that are within a distance of ϵ from these core points are also assigned to the clusters. However, they do not expand the cluster, meaning that other points that are only within a distance of ϵ from non-core points are not assigned to the cluster. The remaining points are considered outliers. The number of outliers and the number of clusters are treated as features in this model. The values of ϵ and m are obtained using a grid search over the possible values shown in Table 5. The evaluation score used is the Calinski-Harabasz score, which calculates the ratio of the minimum distance within clusters to the minimum distance between clusters [2]. The clustering with the highest score will be used to extract the features. The validation process is conducted individually for each instance, potentially yielding varying optimal parameters across instances. The grid is chosen based on visual inspection, considering approximately 25 randomly selected instances. Utilizing this grid, the algorithm consistently identifies clusters for each instance in a manner that aligns closely with how human visual inspection would assign them.

Table 5: DSBSCAN GRID

ϵ	$3C^E_{min}$	$4C^E_{min}$	$5C^E_{min}$	$6C^E_{min}$	$7C^E_{min}$	$8C^E_{min}$	$9C^E_{min}$
m	3	4	5				

Figure 3 shows the outcome of DBSCAN with the grid search on one of the drilling problem instances. It can be seen that the algorithm detect six clusters and a total of 31 outliers, which is inline with human visual inspection.

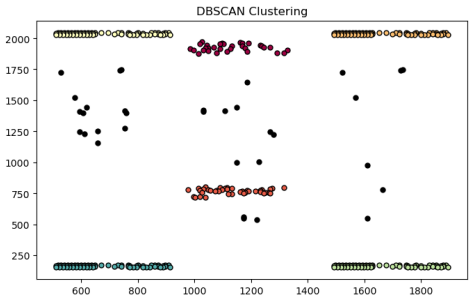


Figure 3: DBSCAN on TSP instance

The final category pertains to the solution and involves the computation of various bounds. The inclusion of these bounds in the model is meaningful only if they can

be computed efficiently. Therefore, the following three specific bounds are chosen for consideration. Firstly, the minimum spanning tree length (MST) is included, which is calculated using Prim's algorithm [39]. This metric represents the length of the minimum spanning tree in the graph. Secondly, the bound Ord is computed by sorting all edges from shortest to longest and summing up the lengths of the first n edges. This bound provides insight into the cumulative length of the shortest edges in the graph. Lastly, the $NNbound$ is introduced, which sums up the lengths of the nearest neighbors for each vertex. It is important to note that this bound differs from the nearest neighbor heuristic, as it does not necessarily result in a connected path. The inclusion of this bound aims to explore its relationship with the other bounds, as it may offer valuable information. By examining the relationships among these bounds, valuable insights can be gained. For instance, if the $NNbound$ is closely aligned with the length of the MST, it indicates that the minimum spanning tree has few branches, resulting in a more circular shortest path. This observation suggests that the insertion heuristics would likely perform relatively well in such scenarios. Overall, the incorporation of these bounds enables an exploration of their interrelationships and potential significance in guiding the performance of the model.

In total, the model utilizes 21 different features. For a comprehensive overview of all the features, please refer to table in appendix B. Additional features that have been suggested in the literature include the exact coordinates of the centroid [46] and clustering coefficients [7]. However, validation on smaller data sets has indicated that these features do not contribute to the overall performance of the model. One possible explanation is that the clustering coefficient exhibits a nearly perfect correlation with the average clustering coefficient (C^V_{avg}) feature. Therefore, including both features does not provide any additional information. Furthermore, regarding the centroid feature, many instances in the training data are shifted such that the centroid coincides with the origin, resulting in a low variance for this feature. Consequently, it has been decided to exclude these features from the model, as they do not contribute significantly to its performance. Some features for the two benchmark models (KNN and DT) are standardized based on size. To be more precise, the bound features and $Rect$ are divided by E_{number}, while the number of outliers and clusters are divided by V_{number}. This was also tested on a small scale for the neural network, but the results were somewhat inferior due to the standardization, in contrast to KNN and DT.

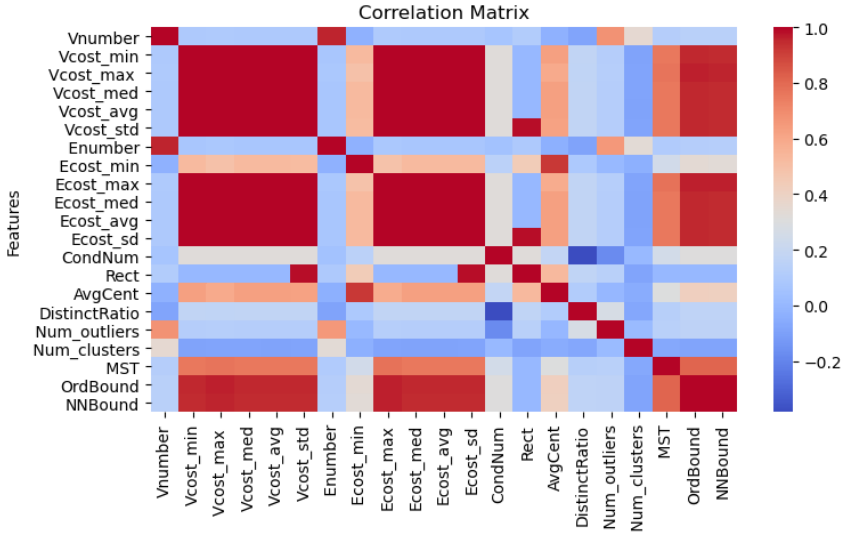


Figure 4: Correlation Matrix

When analyzing the correlations among the various features, distinct patterns emerge, revealing logical relationships. Figure 4 illustrates these patterns through a heatmap of the correlation matrix. Notably, two distinct correlation blocks can be identified. The first correlation block encompasses all the vertex and edge features, which exhibit correlations with one another. This observation aligns with our intuition, as larger graphs tend to have more vertices and longer edges. Additionally, this correlation extends to the group of features related to bounds. It is expected that these bounds would exhibit correlations among themselves, as they are constructed in a way that often involves shared edges. Despite the presence of clear correlations between certain features, it has been determined to include these features in the model. This decision is based on two primary reasons. Firstly, some of these features have been established as valuable in existing literature. Secondly, although multicollinearity often leads to increased variances in models, the relationship between certain features may hold significance for heuristic selection. For instance, the relation between the bounds contains vital information pertaining to the performance of specific heuristics. For further elaboration, please refer to Section 4.5.

5 Data

The nature of the applications necessitates the use of artificial data, as a significant amount of large instances is necessary to obtain sufficient training and testing data for the model. In contrast to prior studies, relying on subgraphs of existing instances of TSP is not feasible due to the majority of these instances being smaller than the required size for this application. As a result, the decision was made to generate artificial instances of TSP. Two different methods are used mainly, complete random instances and modified instances using data augmentation.

5.1 Randomly generated instances

The objective of constructing the data is to ensure that the generated instances exhibit a certain degree of symmetry, patterns, and density. To achieve this, the following design is proposed. Initially, the size of the instance is determined by the number of nodes, denoted by n, which is drawn from a discrete uniform distribution over the interval [100, 3000]. Subsequently, the density parameter λ is determined, with a uniform probability distribution across three levels: "Sparse," "Medium," or "Dense." Based on these parameters, two distinct sets of nodes are constructed.

Dataset A & B

Set A exist of n coordinates which are drawn from a bivariate normal distribution. This distribution has the origin as mean and covariance matrix Σ. With: $\Sigma = \begin{bmatrix} 10^s & 0 \\ 0 & 10^s \end{bmatrix}$. Here $s \in \{1, 2, 3\}$ and depends on λ. Note that the density of the graph decreases in s, because the variance of the samples increases. This effect can be seen in figure 5. This data construction method yields random point placement in a circular pattern, thereby simplifying the representation of a city with a densely populated city center and sparsely populated suburbs at the outskirts of the city.
Set A does not incorporate any discernible patterns, distinguishing it from structured applications such as the circuit board example. Therefore, another n coordinates are created in set B.

In this set a complete grid is constructed starting from the origin and with equal step size h in both vertical as horizontal direction till point (n,n) is reached. Afterwards the grid is shifted such that the center lies in the origin. This results in a total of $(\frac{n}{h})^2$

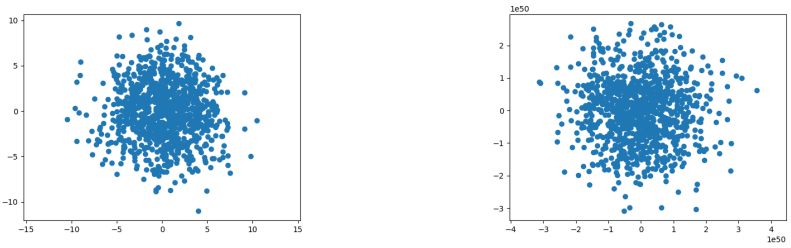


Figure 5: Set A, Left: $s = 1$. Right: $s = 3$.

points. Here, h is determined by density level λ. Afterwards, the final n points are drawn randomly out of the constructed grid. Note that we should have $h \geq \lceil \sqrt{n} \rceil$, to be able to draw n nodes. Define this as h^*. If $h = h^*$, we obtain a almost perfect grid which is not a typical TSP case. Therefore, it is chosen that $h \in \{\frac{3}{4}h^*, \frac{3}{8}h^*, \frac{3}{16}h^*\}$. Note that when the h is increasing the graph becomes more dense and as a result the pattern becomes more clear, which is illustrated by figure 6. The final coordinates for the instance are obtained based on mixing parameter ϕ, which is drawn uniform on the unit interval. Then $\lceil n\phi \rceil$ nodes are picked randomly out of set A and $\lceil n(1-\phi) \rceil$ nodes are picked randomly out of set B. The idea is that ϕ determines the level of symmetry in the graph, with low levels of ϕ indicating a lot of patterns in the instance.

Figure 6: Set B, Left: $h = \frac{3}{16}h$. Right: $h = \frac{3}{4}h$.

5.2 Modified instances

While it is challenging to replicate real-life scenarios accurately using artificial data, researchers have found success in literature by employing data augmentation techniques on real instances [24, 52]. These techniques have proven effective in simulating real-world variations. Some commonly used data augmentation techniques include edge and vertex dropping, vertex rotation/flipping, scaling of edges or vertices, and noise injection [52]. The underlying principle behind these techniques is to generate a diverse set of augmented data without introducing imbalances in the dataset. For the meta-learning model at hand, the objective is to modify instances in a way that induces variations in the heuristic performances. The model aims to learn the relationship between specific changes in instance characteristics and the resulting variations in heuristic performance. To facilitate this learning process, two additional sets of augmented data are created specifically for training purposes. These augmented datasets enable the model to capture a broader range of scenarios and enhance its ability to generalize to unseen instances.

Dataset C

Set C is comprised of exceptionally large instances of TSP, consisting of over 18,000 nodes. These instances are exclusively utilized for training purposes. Set C encompasses six distinct instances, namely: brd14051, d15112, d18512, pla7397, rl11849, and usa13509. These instances serve as benchmarks for various applications and incorporate real-life scenarios involving cities in the United States and Germany, as well as the drilling sequence problem [41]. To generate the instances in set C, the parameter n is randomly selected from the interval [100, 3000]. Subsequently, n random vertices are chosen from the original instance, resulting in the creation of a new instance. This process is repeated 1000 times for each of the six instances within set C.

Dataset D

The final dataset, denoted as D, consists of smaller instances of TSP that are derived from real-life applications. These instances are used both for training and testing purposes. However, applying data augmentation techniques directly on the test instances, and then using the augmented instances for training, introduces a bias, as the test data would still contain traces of the training data. To address this issue, a random selection of 26 test is made out of the test instances. The instances themselves will not be used for training; rather, only their augmented versions will be utilized. The data augmentation

process is as follows. Firstly, each instance is shifted such that its centroid becomes the origin. Next, the originality parameter, denoted as o, determines the fraction of vertices in the instance that will remain unaltered. Specifically, three values are chosen for $o \in \{0.1, 0.25, 0.5\}$, resulting in enough mutations consider it a new instance. This ensures a sufficient number of mutations to consider them as new instances. For the remaining fraction of vertices (1 - o), two different techniques are employed.

The first technique involves rotating the instance 90 degrees counter-clockwise using the rotation matrix: $\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$. This technique aims to preserve the structure, density, and clustering characteristics of the original vertices while introducing new elements. The second technique, called scaling, shifts the instances randomly in both the horizontal and vertical directions. The shift magnitude is determined by the mean distance from the centroid. The direction of the shift (north/south and east/west) is chosen randomly with equal probability. This technique maintains the instance structure while introducing variations in the position. Therefore, the size of the shift is not random but fixed. Finally, a small random expect is added using noise generation. Noise parameter e is used to determine the level of noise. It is chosen that $e \in \{0.05, 0.1, 0.25\}$, leaving the noise minimal compared with the random generated instances. For the new generated instance which exist of the fraction original and altered instances. A fraction of e extra vertices are randomly created on the rectangle on which these point lie. To allow for outliers an extra 10% outlier margin is taken inconsideration. To elaborate, the X coordinate is drawn uniformly on the interval $[1.1X_{min}, 1.1X_{max}]$, and the Y-coordinate is independently drawn in the same manner for the Y-coordinates.

Finally, four different combinations of configurations are utilized to create a wide variety of data. In the first combination, the entire fraction of $1 - o$ vertices is dropped, and noise is introduced as a replacement. In the second combination, half of the altered set is dropped, and the remaining vertices undergo rotation, followed by the addition of noise. In the third combination, half of the altered set is dropped, and a shift is applied to the remaining vertices, followed by the addition of noise. In the fourth combination, a third of the altered set is dropped, rotation is applied to another third, a shift is applied to the final third, and noise is added. To account for the random effect of choosing which technique is applied to each vertex, for each combination of instance, parameter, and technique, five new instances are created. This results in over 4600 new instances. In total, the four trainsets combined result in over 12,000 observations, which are used for

training and validating the models.
To demonstrate the effect of these data augmentations, consider Figure 7, which depicts one of the original instances before the data augmentation process is applied.

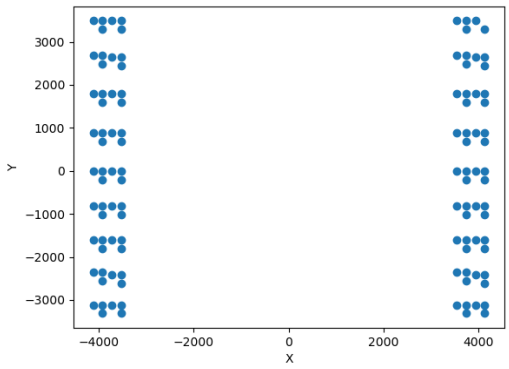
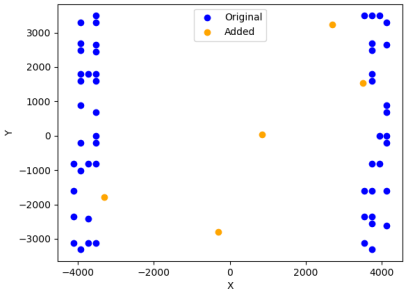


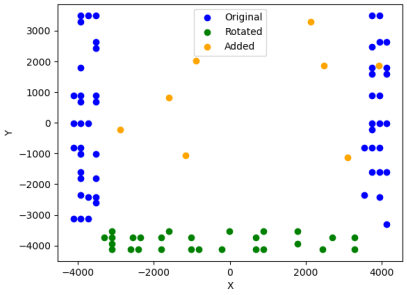
Figure 7: Instance Pr107.tsp original

Visual inspection identifies a clear pattern in this instance with two large clusters or multiple smaller ones. Figure 8 presents the four different combinations of the data augmentation are applied on this instance using $o = 0.5$ and $e = 0.1$. It becomes clear that each technique changes different characteristics of the instances. For example shifting creates additional clusters, whereas rotation seems to widen the cluster in a L-shape. Adding noise results in outliers and dropping observations slightly adjusts the pattern of the instance.

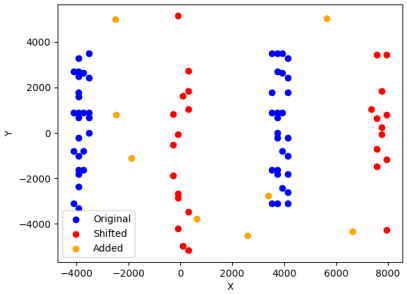
Figure 8: Data augmentation on pr107.tsp



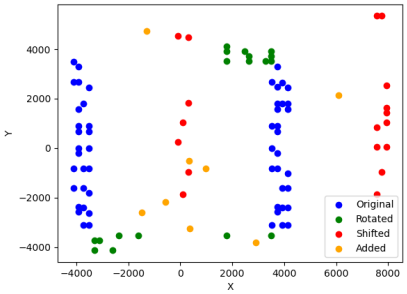
Drop, Noise



Drop, Rotate , Noise



Drop, Shift, Noise



Drop, Shift, Rotate, Noise

5.3 Test dataset

For the test dataset a selection of instances from the TSPLIB library [41] is used. The library instances are a combination of network wiring, drilling and classical TSP problems. Figure 9 illustrates a histogram depicting the performance ranks of four heuristics on the these test instances. The histogram demonstrates that none of the heuristics exhibit strict dominance over the others. However, on average, GR and FI outperform NN and NI. A complete list of all of these instance and the corresponding performance of the heuristics on these instance is provided in Table 1, which can be found in Section 4.1.

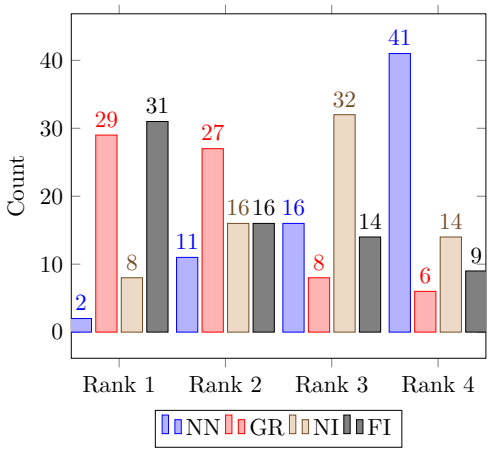


Figure 9: Histogram of Algorithm Performance on TSPLIB

Table 6 presents the mean rank of the heuristics on the test set. The "Grey" set comprises instances used to construct dataset D, these instances themselves are only used for testing, these are the grey instances in Table 1. Whereas, the "White" instances represent the set of instances not used for constructing dataset D. A slight variation in heuristic performance is observed between the two sets. However, since the grey instances were randomly selected, it is expected that this would not significantly affect the model's performance. On average, GR outperforms the other heuristics.

Testset	**NN**	**GR**	**NI**	**FI**
Grey	3.25	2.08	2.50	2.17
White	3.41	1.76	2.85	1.98
Total	**3.36**	**1.87**	**2.73**	**2.04**

Table 6: Mean Ranks Testset

To assess the different models performances, it is compared to a alternative method where the model with the lowest mean rank is selected for each instance. As mentioned in Table 6, this method would select GR for all instances. The evaluation is conducted on a test set consisting of the 26 grey instances used for constructing dataset D and the 44 white instances which were left untouched in the data generation process, along with an additional 70 instances extracted randomly from the test dataset prior to training. This number results in a balanced testset. Therefore, the predicted best rank for these 140 instances is compared against the performance of the GR.

6 Results

6.1 Data properties

This section focuses on the evaluation of the data. Table 7 presents the mean ranks of the four heuristics on the randomly generated instances at different density levels. It is evident that NI performs relatively well performs relatively well on average when applied to dense graphs with limited structure, as observed in dataset A. This advantage arises from the many favorable insertion points around the vertices due to the high density. Conversely, in dataset B, where close neighbors are less common, the insertion heuristics has access to fewer advantageous insertion points. However, this advantage of NI diminishes as the graph density decreases, which likely impacts the ranks of NN as well. As for sparse instances, the performance of both insertion heuristics becomes similar on both datasets, while GR emerges as the dominant heuristic. In dataset B, GR outperforms the other heuristics in 95% of the cases, which results in low mean ranks. Nonetheless, since dataset B only contributes 8% of the total dataset, the observed dominance of GR does not appear to create a significant class imbalance.

Table 7: Mean Ranks Set A Set B

SET A	**NN**	**GR**	**NI**	**FI**
Dense	3.90	3.00	1.25	1.85
Medium	2.73	1.06	2.90	3.32
Sparse	2.71	1.10	2.95	3.24
Total set A	**3.11**	**1.74**	**2.37**	**2.81**

SET B	**NN**	**GR**	**NI**	**FI**
Dense	2.60	1.08	3.15	3.17
Medium	2.64	1.06	3.19	3.10
Sparse	2.59	1.07	3.16	3.18
Total set B	**2.61**	**1.07**	**3.17**	**3.15**

GR is the dominant heuristic in dataset C, as shown in Table 8. However, this dataset presents an interesting observation where the other three heuristics achieve similar mean ranks. It is desirable for the model to identify the specific characteristics that lead to better performance for each heuristic, especially considering the high similarity among these original instances.

Finally, the impact of data augmentations is also presented in Table 8. Comparing these results with Table 6, we observe that the data augmentations have only a slight effect on the mean ranks. Specifically, NI shows an average increase of half a rank, while both GR and FI experience a slight decrease in rank. Interestingly, in more than half of the cases, the best performing heuristic shifts from one to another. This indicates that data augmentation does have an influence on the characteristics of the instances. Notably, rotation appears to result in the most significant rank change among the four augment-

SET C	NN	GR	NI	FI
brd14051	2.93	1.12	2.78	3.17
d15112	2.85	1.08	2.88	3.19
d18512	2.80	1.08	2.90	3.22
pla7397	3.85	1.15	3.21	2.06
rl11849	2.83	1.11	3.09	2.97
usa13509	3.50	1.47	3.06	1.96
Total set C	**3.10**	**1.18**	**2.96**	**2.76**

SET D	NN	GR	NI	FI
Drop, Noise	3.47	2.02	3.01	1.50
Drop, Noise, Rotation	3.40	1.72	3.00	1.87
Drop, Noise, Shift	3.29	1.80	3.15	1.75
Drop, Noise, Rotation, Shift	3.30	1.74	3.06	1.90
Total set D	**3.35**	**1.80**	**3.06**	**1.78**

Table 8: Mean Ranks Set C & Set D

ation techniques.

Dataset	NN	GR	NI	FI
Test	3.36	1.87	2.73	2.04
Train	3.15	1.43	2.97	2.45

Table 9: Mean Ranks Test Trainset

The mean ranks of the total training set exhibit slight differences compared to the test set, as can be found in Table 9. These differences indicate an imbalance between the two datasets, which may negatively impact the performance of the prediction models on the testing set. Such discrepancies between the training and testing sets arise from the challenges associated with simulating real data using artificial data.

6.2 Model validation

The performed cross-validation provided valuable insights into the optimal hyperparameter configuration. The bivariate heat plots in Figure 16 in appendix C depict the results from the complete search grid, highlighting the key findings during the cross-validation process. One of the main observations is that measures designed to prevent overfitting play a crucial role in the model's performance. Specifically, a low dropout rate and a low ridge parameter consistently lead to poorer model performance, irrespective of the other parameters. This emphasizes the need to maintain an appropriate level of network density and utilize regularization methods. The best hyperparameter configuration, as shown in Table 10, is used to train the neural network. Additionally, the table presents the performance of the worst configuration, demonstrating a substantial difference in performance and underscoring the importance of validating the model. Recall that ρ is the Spearman's correlation coefficient introduced in equation 6.

Table 10: Optimal hyperparameters

	Batch size	Dropout rate	Ridge parameter	Learning rate	ρ
Best	36	0.3	0.001	0.001	0.770
Worst	36	0	0	0.0001	0.508

In addition to the tested configurations mentioned above, several other experiments were conducted on a smaller scale. These experiments involved exploring different network structures. For instance, the effect of using a classification network versus a regression network was examined. Furthermore, instead of training separate networks for each heuristic, a single comprehensive network was trained. Additionally, alternative loss functions, such as ListNet Loss [3], which prioritize the overall accuracy of all four ranks rather than individual rank accuracy, were investigated. However, all of these alternative approaches yielded similar results, primarily indicating that the network consistently predicts values close to the mean ranks for each instance and heuristic. This suggests that the network is unable to capture the data's variance and therefore prone to underfitting. Consequently, the employed network structure, which consists of separate regression networks using Mean Squared Error (MSE) loss, emerges as the only viable structure for this meta-learning model.

When conducting cross-validation on the two benchmark models (KNN and DT), some further evidence of overfitting can be observed. Figure 10 illustrates that these models exhibit better performance when employing measures that mitigate overfitting. Specifically, limiting the depth of the tree and constraining the number of features in DT obtain favorable results. However, for shallow tree depths, solely restricting the depth proves to be sufficient. On the other hand, in the case of KNN, overfitting can be mitigated by limiting the number of neighbors. The optimal hyperparameters determined for these models are as follows: a tree depth of 8 without restricting the number of features for DT, and the optimal number of neighbors for KNN is 69.

6.3 Model comparison

To assess whether the models exhibit differences in performance, we compute the correlation coefficients and their corresponding confidence intervals using the methods described in Section 4.3. Table 11 presents the Spearman estimate of ρ, along with the lower bound (LB) and upper bound (UB) of the 95% confidence interval. Prior studies have suggested that these models should demonstrate similar performance when applied to more

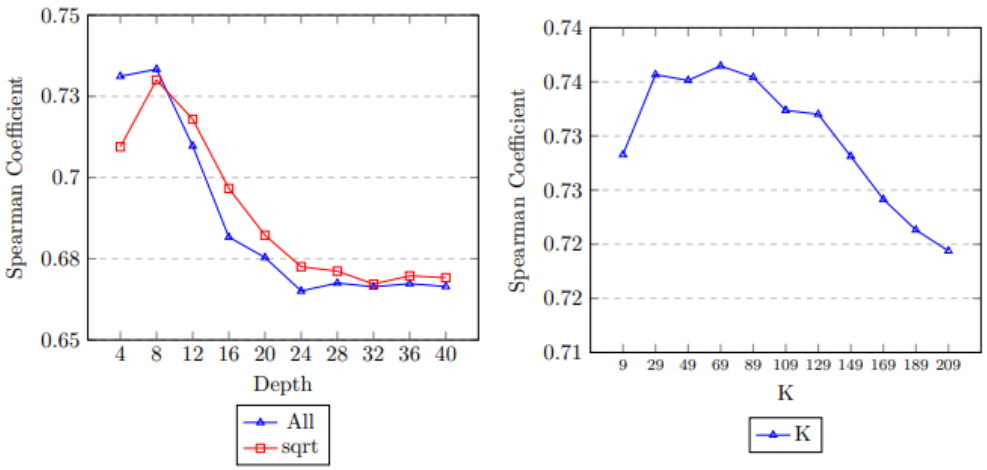


Figure 10: k-fold Validation for DT and KNN

complex heuristics [23, 24]. Therefore, it is crucial to test whether this hypothesis holds for the fast greedy heuristics employed in this thesis. Upon analyzing Table 11, we observe that the majority of the intervals overlap with each other. However, the NNet consistently exhibits higher upper and lower bounds. While the confidence intervals suggest a slightly superior performance for the NNet, formal tests for the difference of the mean performances would be needed to obtain more statistical evidence. The table distinguishes between "Train", representing the 70 instances taken from the complete trainset, and "Test", comprising the 70 instances from the library [41]. Although the model's correlation coefficients are slightly lower compared to those of models involving more complex heuristics [24], it demonstrates a clear correlation between the predictions and the true ranks.

Table 11: Correlation Coefficients for the Models

	NNet			KNN			DT		
	LB	ρ	UB	LB	ρ	UB	LB	ρ	UB
Train	0.651	0.769	0.850	0.607	0.737	0.829	0.603	0.734	0.827
Test	0.499	0.657	0.773	0.410	0.589	0.723	0.440	0.611	0.740

Further evidence of the superior performance of the neural network is apparent when considering the second evaluation measure. Table 12 presents the accuracies of the different models as a ratio of correctly predicted ranks to the total number of predicted ranks. In the table, the "Correct" column represents the fraction of instances where

the complete ranking of the four heuristics is predicted correctly out of the 24 possible rankings. Remarkably, the neural network is able to correctly predict the complete ranking in more than a third of the instances, while the other models achieve far fewer successes. However, in practical applications, this level of accuracy may be less significant because of time limitations, allowing only a few heuristics to be tried before deciding on a method to use. The other columns represents the ratio of instances in which the best performing heuristics is given label at most m. For example, most applications, only a single heuristic can be calculated within some given time constraints. In this context, the column "$m = 1$" represents the fraction of instances for which the model correctly identifies the best-performing heuristic out of the four options. Here, we observe that all models are able to correctly predict the best heuristic in the vast majority of cases. Nonetheless, the neural network outperforms the other models by accurately predicting the best heuristic in 10% more cases. If the application's time constraints permit the exploration of two heuristics, we can observe column "$m = 2$". All models are almost always able to identify the best performing heuristic in this case. In other words, if the true rank is equal to 1, the models almost always predict either rank 1 or rank 2. However, when the application allows the exploration of three out of the four heuristics, the neural network consistently identifies the best performing one accurately, while the other two models still encounter difficulties in some situations. Overall, it can be concluded that the neural network demonstrates the highest accuracy among the three models, making it a promising approach for ranking heuristics.

Table 12: Accuracy of Different Models

		Accuracy			
		Correct	$m-1$	$m=2$	$m=3$
NNet	Train	0.390	0.857	0.970	1.000
	Test	0.343	0.729	0.886	1.000
KNN	Train	0.214	0.757	0.957	0.971
	Test	0.186	0.657	0.843	0.986
DT	Train	0.314	0.743	0.971	0.986
	Test	0.200	0.657	0.857	0.986

The performance measures provide valuable insights into the predictive capabilities of the different models, but they do not account for their computational complexity. It is equally important to understand the efficiency gains achieved when comparing more complex models with simpler alternatives. Consequently, a tradeoff must be made between

the time consumed by training, validating, and testing the models and the time saved by obtaining more efficient tours. The significance of the models lies in their ability to achieve a substantial reduction in tour length. If the reduction in tour length is not considerable, the computational cost of calculating the features and making predictions may outweigh the benefits. Therefore, it is essential to assess the tradeoff between computational cost and efficiency gains. To address this, a fourth alternative approach is considered, which involves the simple method of selecting the overall best heuristic for each instance, which, in this case, is GR. This approach serves as a benchmark to compare the performance of the more complex models. By comparing the efficiency gains of the complex models with the straightforward GR method, we can determine whether the additional computational cost of the complex models is justified by the significant reductions in tour length they achieve.

The neural network requires the most time for training and validation, estimated between 15 to 45 minutes for training and several days for validation, depending on the CPU's processing power. In this thesis, the computational environment utilized is Google Colab, which employs a 64-bit Intel x86 CPU with 12 GB of RAM. The estimates presented in this work are based on computations performed within this specified hardware configuration. The decision tree model takes a comparatively shorter time, approximately 3 to 5 minutes for training and around an hour for validation. KNN, on the other hand, requires no training time and only a few minutes for validation. Once trained and validated, all models can make predictions almost instantly. Obviously, the alternative method of always selecting GR has close to zero training, validation and prediction times.

The boxplots presented in Figure 11 illustrate the temporal savings achieved by utilizing different models concerning the lengths of the acquired tours. The blue bars in the visual representation illustrate the efficiency of the tours achieved by each heuristic. This efficiency is quantified by the ratio of the difference between the length of the obtained tour and the length of the optimal tour, in comparison to the length of the best tour attained by any of the four heuristics. The red bars use a different metric to evaluate the efficiency of the methods. The metric is defined as the ratio of the difference between the length of the obtained tour and the best-known length for each instance, divided by the best-known length. These optimal solutions were derived from the TSPLIB library [41] for the real-life test instances and are acknowledged as the optimal solutions for this thesis. Instances for which their optimal solution using Euclidean Distance is unknown are excluded from this analysis.

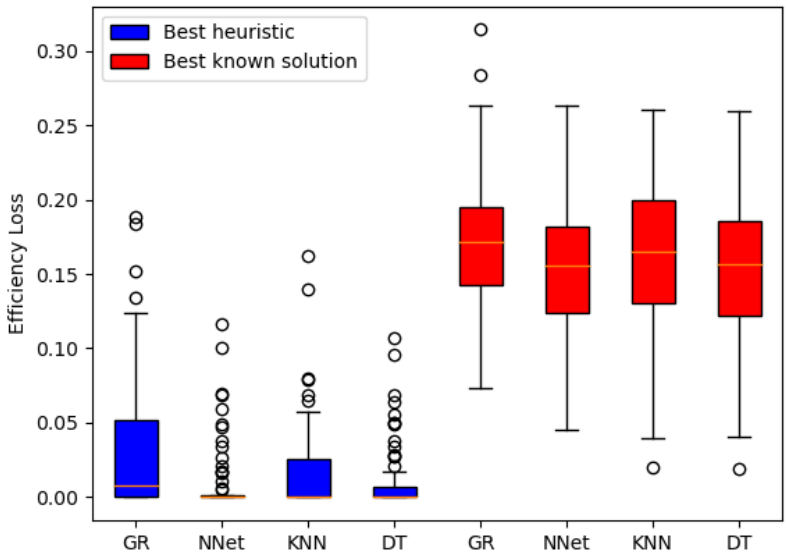


Figure 11: Models performances against GR

In our comparative analysis of the blue bars, it is evident that all models demonstrate superior performance when compared to the GR heuristic. On average, the neural network model suggests a heuristic that exhibits approximately 2.4% greater efficiency than GR. Similarly, the other two models perform slightly less efficiently than NNet, but still achieve a remarkable efficiency gain of over 2% compared to GR. Moreover, relying solely on GR as the default choice proves to be accurate in only 44% of the instances. In contrast, as depicted in Table 12, the models can accurately predict the optimal choice in the vast majority of cases. Furthermore, an exclusive reliance on GR would lead to outliers with an efficiency loss of nearly 20%, while the neural network model's extreme cases only result in a maximum efficiency loss of 12%. Analyzing the red bars illustrates that all the models still experience considerable efficiency loss compared to the optimal solutions, this is a consequence of the inherent greedy nature of the four heuristics employed. Nonetheless, all models outperform the use of GR alone. They consistently remain within 25% of the optimal solution, with a median efficiency of approximately 15%, representing a 2% increase in efficiency compared to GR. These results indicate that the model holds significant potential as a valuable improvement for various applications, offering a more favorable trade-off between efficiency and computational complexity than

a single heuristic approach. Upon comparing the three models, we observe similar tour efficiencies. The NNet exhibits a slightly more efficient median, while the other two models achieve more efficient outliers. DT ranks as the second-best option in terms of prediction quality indicators. However, when considering both the computation times and the ability to determine feature importance, the efficiency loss in DT compared to NNet might not be of significant concern. Consequently, for applications where the time or computational resources required to train the NNet become prohibitive or when a thorough understanding of influential features is crucial, DT emerges as the best choice.

6.4 Effect of artificial data

In this thesis, a substantial set of artificial instances of the Traveling Salesman Problem (TSP) is employed. It is crucial to analyze the implications of using these instances. Hence, this section conducts a detailed examination of the results obtained from the best-performing model, which utilizes a neural network. Table 13 expands upon previous tables by dividing the "Test" instances into two distinct groups. The "Grey" instances consist of real data instances used to generate dataset D (referred to as the grey instances in Table 1 in the section 4.1), while the "White" instances are those that were not utilized in the generation of dataset D. The analysis reveals that the performance difference between the "Train" and "White" instances is relatively small. This suggests that the artificial data of the form presented in this thesis, effectively simulates the real data, and the model's performance generalizes well to unseen data. However, a substantially larger performance difference is observed for the "Grey" instances. This outcome may seem counterintuitive since these instances should bear greater resemblance to a portion of the training data compared to the "White" instances. One potential explanation for this disparity could be the difference in mean ranks (as demonstrated in Table 6), in conjunction with the relatively small size of the "Grey" set. Consequently, these instances may present a more challenging prediction task due to the limited representation in the training data. Alternatively, it is plausible that data augmentation techniques applied during the creation of dataset D transform these "Grey" instances to a degree that has a deceiving effect on the predictions. This hypothesis is reinforced by the observed difference in mean ranks when comparing Table 6 to Table 8. Such transformations may have an impact on the model's ability to generalize effectively to these instances, thus influencing its predictive performance.

Table 13: NNet Model performance

Dataset	ρ	Correct	One	Two	Three
Train	0.769	0.386	0.857	0.971	1.000
Grey	0.569	0.308	0.577	0.808	1.000
White	0.709	0.364	0.818	0.955	1.000

In order to investigate this phenomenon further, a detailed analysis of the impact of data augmentation is presented in Table 14. The table displays the in-sample ρ values for each instance of dataset D. It is evident that none of the data augmentation techniques yield a lower coefficient; instead, all techniques exhibit significantly lower correlation coefficients compared to the overall in-sample coefficient of 0.789. This observation suggests that dataset D possesses distinctive characteristics that differ significantly from the other datasets.

Data Augmentation	ρ
Drop, Noise, Rotation, Shift	0.57
Drop, Noise	0.41
Drop, Noise, Rotation	0.55
Drop, Noise, Shift	0.54

Table 14: Data augmentation effects on dataset D

Finally, it is important to test diversification of the model on the different datasets and heuristics. For example, when the artificial data is unable to capture features present in the real-life instances which are important to certain heuristics the model might face challenges in predicting the ranks for these heuristics. Table 15 presents the results in terms of the ratio of correctly predicted ranks by the NNet model, indicating its predictive power on each heuristic. The analysis reveals that the performance differences among the heuristics are not large, with GR being the easiest to predict and NI being the most challenging. GR tends to have a higher frequency of rank 1, whereas NI displays more variability across ranks. Despite this variation, the model demonstrates the capability to predict ranks for each heuristic. Furthermore, when considering the artificial data, we observe a similar pattern to that in Table 13. Specifically, the instances used in constructing dataset D exhibit an overall lower accuracy across all heuristics. This finding further supports the previous claims that dataset D possesses unique characteristics that are challenging for the model to capture, potentially due to the data augmentation techniques and the relatively small sample size of only 26 instances in this set. Consequently,

the model's average predictions for this set are not as accurate. Importantly, the results do not indicate any evidence that the use of artificial data biases the model towards favoring a specific heuristic. In conclusion, the NNet model demonstrates a reasonable level of diversification in predicting ranks for different heuristics. While dataset D poses challenges for the model, it does not indicate any inherent bias towards a particular heuristic. The model's performance variations across the heuristics are more likely influenced by the nature of the heuristics themselves and the characteristics of the datasets rather than any systematic bias introduced by the artificial data.

Table 15: Accuracy predicted ranks NNet Testset

	NN	GR	NI	FI
Train	0.586	0.729	0.543	0.671
Grey	0.500	0.462	0.423	0.615
White	0.545	0.795	0.500	0.636

6.5 Features importance

In order to identify the most significant features, an analysis is conducted on the initial three layers of the decision tree. The corresponding figures, namely 12, 13, 14, and 15, are provided in appendix A. Please recall that certain features, such as the number of outliers and number of clusters, are divided by the size of the dataset, as elaborated in Section 4.4. Additionally, as previously mentioned, the entire dataset underwent standardization. However, for the sake of enhancing interpretability, the standardization process is reversed for the threshold values depicted in the figures. Nevertheless, it is important to note that the thresholds pertaining to the number of clusters and outliers remain divided by the number of vertices in the dataset. It is evident that these four trees exhibit distinct structural variations. In the case of NN and FI, the initial split is influenced by the number of edges, whereas GR relies on the number of vertices as a crucial factor. On the other hand, NI employs the distinct ratio for its splits. It is worth noting that the condition number appears within the first three layers in all heuristics, emphasizing its significance as a feature. For NN, features associated with clusters appear to play a significant role. Specifically, a lower number of clusters and reduced presence of outliers contribute to an improved heuristic ranking. This is due to the heuristic performing well within clusters, while being penalized for transitions between clusters or outlying vertices, as described in Section 4.1.

7 Conclusion

This thesis explores the opportunities to use meta-learning for the traveling salesman problem applications which require low complexity methods on large scale instances. Therefore, fast heuristics are applied on a set of meta-features which capture essential characteristic of the instances such as density, spatial distribution and clustering. These features have the property that they can be calculated fast and include properties related to the edges and vertices, graph structure and performance bounds. The heuristics best suited for such applications and used in this study are: Nearest Neighbor, Greedy Algorithm, Nearest Insertion and Farthest Insertion. The performance of these heuristics are ranked and obtained on a large dataset. This dataset is created using a combination of random generated instances and instances obtained based on real life cases using data augmentation techniques. Three models are built around a regression machine-learning techniques and make use of label ranking and different techniques to prevent overfitting. The used ML techniques are a simple K-nearest neighbour model as well as more complex decision trees and neural networks.

7.1 Model performance

We successfully conducted training and validation for all models within a reasonable temporal framework, bearing in mind the one-time nature of the training process. Notably, the neural network's training phase exhibited the most significant temporal and computational resource consumption. To address this, we used the computational capabilities of Google Colab, deploying a computing setup featuring a 64-bit Intel x86 CPU equipped with 12 GB of RAM. The process of identifying optimal parameters for the neural network demanded several days of validation efforts. In contrast, the training and validation processes for the alternative models demonstrated comparably shorter temporal demands, typically spanning minutes. Evaluating the ranking using each of the obtained model takes less than a seconds. The performance of the models indicates indicate that models are able to identify patterns in the extracted characteristics of the instances. Different aspects, such as the number of clusters, the condition number, and spatial distribution properties, exhibit correlations with the heuristics selection. Consequently, these factors play a crucial role in constructing the meta-learning model. It is essential to employ cross-validation during model training, as hyperparameters have a significant impact on the predictive power of the models, thereby revealing their suscept-

ibility to overfitting. The best-performing model out of the three tested is the neural network, the performance of the DT is only slightly lower, and the KNN shows the worst performance. The NNet exhibits remarkable proficiency in identifying the best-performing heuristic in the vast majority of cases and virtually always suggests one of the two best heuristics. Based on the Spearman correlation coefficient, it is evident that all three models establish clear correlations between the test predictions and the true heuristic performances. The best model achieves estimated correlation coefficients of approximately 0.75. While these results are slightly lower than the correlation coefficients obtained by models from the literature, that utilizing more complex heuristics, they still highlight the predictive power of meta-learning models, particularly for greedy and fast heuristics on large instances.

to time constraints and the practical requirements of applications, a straightforward alternative approach might involve selecting the best performing heuristic on average. However, this thesis demonstrates that adopting such an approach leads to a loss of efficiency, where efficiency is defined as the ratios of the obtained tour length using the model to the optimal and best available tour length or best heuristic. On average, tours constructed using one of the meta-learning models exhibit a 2% efficiency gain compared to this simple alternative. Remarkably, this efficiency gain persists even for DT and KNN, indicating the dominance of the meta-learning model over this alternative strategy.

7.2 Choice of machine-learning technique

Based on literature, it appears that for some meta-learning models, the specific choice of machine-learning technique may not be critical, as multiple techniques can yield good-quality results. The results of this study provide evidence to support this claim. Although the more complex neural network model achieves the best overall mean performance, these results do not show substantial differences in terms of correlation coefficients when based on the confidence intervals. Nevertheless, the bounds on correlation in the confidence intervals are consistently higher for the neural network. With respect to predictive accuracy, measured as the ratio of instances in which the best-performing heuristic is among the first m predicted labels, the dominance of the neural network over the decision tree and K-nearest neighbour seems to be larger. Regarding predictive accuracy, the neural network demonstrates a more dominant performance over the decision tree and K-nearest neighbor models. However, in real-life scenarios, the training and

validation times may become important factors in choosing the appropriate technique. The complex structure of the neural network leads to significantly longer training and validation times compared to the other two models. Therefore, it is sensible to apply the neural network in static scenarios where the model does not require frequent modification once constructed. In such cases, the longer training and validation times are not a major concern as the model remains relatively unchanged over time. On the other hand, the decision tree and K-nearest neighbor models may be preferred in more dynamic scenarios where the use of the model requires frequent updates or modifications, and faster training and validation times are critical. Moreover, DT provides slightly better results and more interpretability and may therefore be the better choice between the two.

7.3 Artificial data and Feature importance

The results obtained from the comparison between artificial and real instances indicate that the methods used for generating artificial data successfully simulate the patterns observed in real instances. This is evidenced by the small difference in results between the artificial and real instances. As a result, we can conclude that even in cases where the TSP application lacks sufficient data to construct a large training dataset required for meta-learning, artificial data can serve as a viable substitute. The augmentation techniques that have been demonstrated to be successful in this thesis encompass shifting, rotating, introducing noise, and node elimination.

As previously mentioned, the characteristics related to the spatial distribution and density of the graphs play a significant role in determining the best heuristics for solving instances of the TSP. Among these features, one key finding is the notable influence of the condition number of the distance matrix on heuristic performance. The condition number attempts to identify patterns in the graph of the instance, making it an important feature for predicting the effectiveness of each of the four heuristics. Additionally, features related to clustering and the distinct ratio of edge lengths emerge as crucial in the predictive modeling process. These characteristics provide valuable insights into the structure and topology of the TSP instances, contributing to the meta-learning model's ability to accurately discern the most appropriate heuristic for solving each specific instance. By understanding the importance of these spatial distribution-related features, it becomes possible to develop more effective meta-learning models for TSP, as these features offer valuable clues about the inherent complexities and characteristics of the instances. Despite the presence of substantial correlations among certain features, their relationships retain vital information concerning the graph's attributes.

7.4 Model limitations and future research.

Interesting follow up steps for this study can include the extension to more complex versions of the traveling salesman problem. Some relevant applications might require adaptations from the classical Euclidean TSP case. For example, the PCB applications may require a incomplete network, while other applications might experience an asymmetric case. The current model structure will most likely still provide good results. However, the list of features should be extended as the current features don't obtain any information regarding the connectivity of the graph, therefore missing potential crucial information. Moreover, the heuristics should be adapted to these new settings. A second extension to the classical TSP case could be the SET TSP problem. In this variation, the vertices are assigned to sets, and the objective is to visit all sets instead of visiting all individual vertices. To illustrate this problem, we can translate the traditional problem to a scenario where a salesman has multiple potential locations to meet his potential buyer who works in a certain subregion. The salesman's goal is to efficiently meet every buyer by planning a route that excludes unnecessary locations. An illustrative example relevant to this thesis is the data network design example mentioned in the introduction. Here the challenge lies in connecting numerous network elements, some of which may belong to the same data center or network hub. In such cases, optimizing the route between elements within the same set may be unnecessary. Due to the nature of this problem, the differences in performance between the heuristics are often more pronounced compared to the classical case, as illustrated by Figure 17 in appendix D. Early experiments applying the meta-learning model to this problem have yielded promising results, as the mean ranks of the four heuristics are closer to each other than in the traditional case. Hence, consistently favoring the option with the best average performance will result in more efficiency loss. This suggests that the meta-learning model can effectively leverage the characteristics of the instances to make more balanced and informed heuristic choices. However, a potential challenge arises from the time constraints of the applications. In particular, the greedy heuristic, which was previously able to provide solutions within seconds for large instances, now requires minutes to complete its computations. This increased computation time may not be feasible in time-critical scenarios, necessitating the consideration of alternative problem-specific heuristics that can compute solutions faster.

Other future research could explore the potential effect of certain features, such as the condition number, on improving meta-learning models that incorporate more complex heuristics. Specifically, investigating whether the condition number can enhance the predictive capabilities of such models is of interest. Lastly, some application may require a trade-off between more complex heuristics and computational complexity. Meaning that some application may budget extra computational time if this results in substantial efficiency gain in the tours length. In relation with PCB example, this could occur if the price tags of more high-end products allow for some delay. Therefore, a meta-learning model can be developed which predicts some weighted objective function between the computational and operational efficiency. Based on findings in this study, artificial data using augmentation techniques used in this thesis, could extract valuable information of instances which are essential for this objective. Particularly in these scenarios where large datasets are required but often unavailable, artificial data serves as a valuable substitute for training the models.

A Decision Trees

Figure 12: DT NN

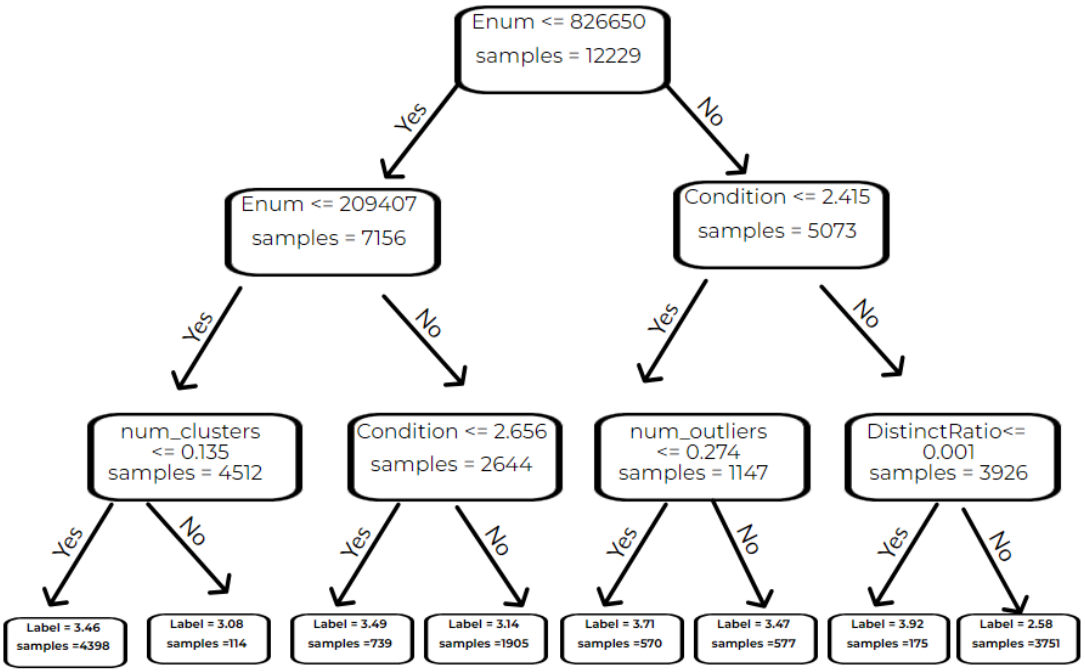


Figure 13: DT GR

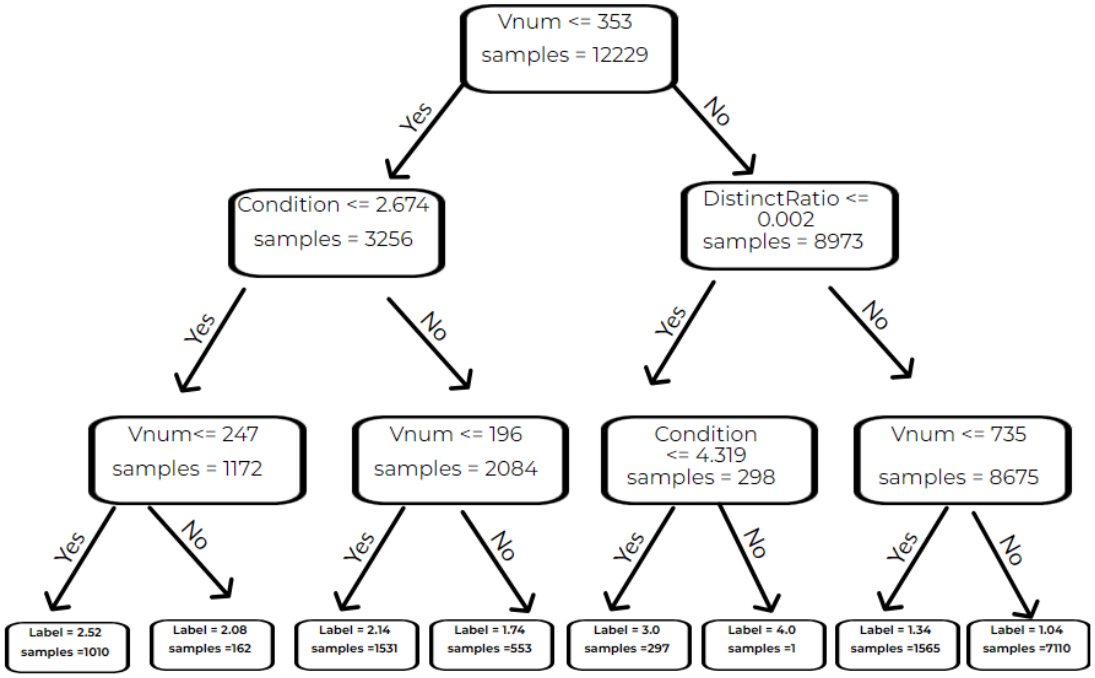


Figure 14: DT NI

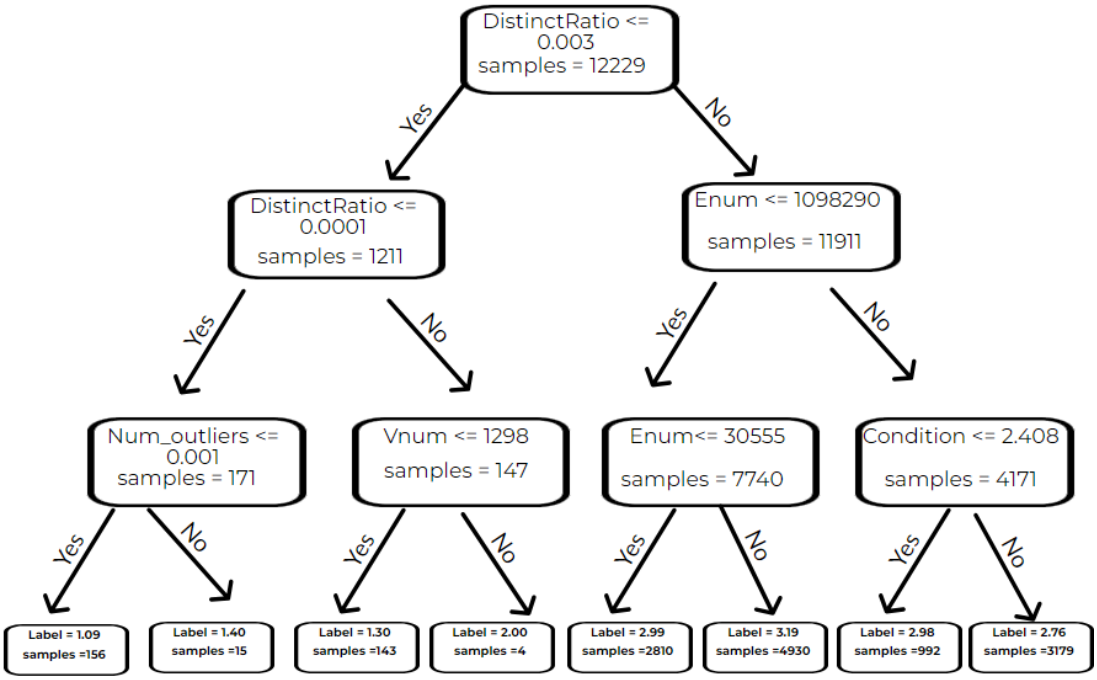
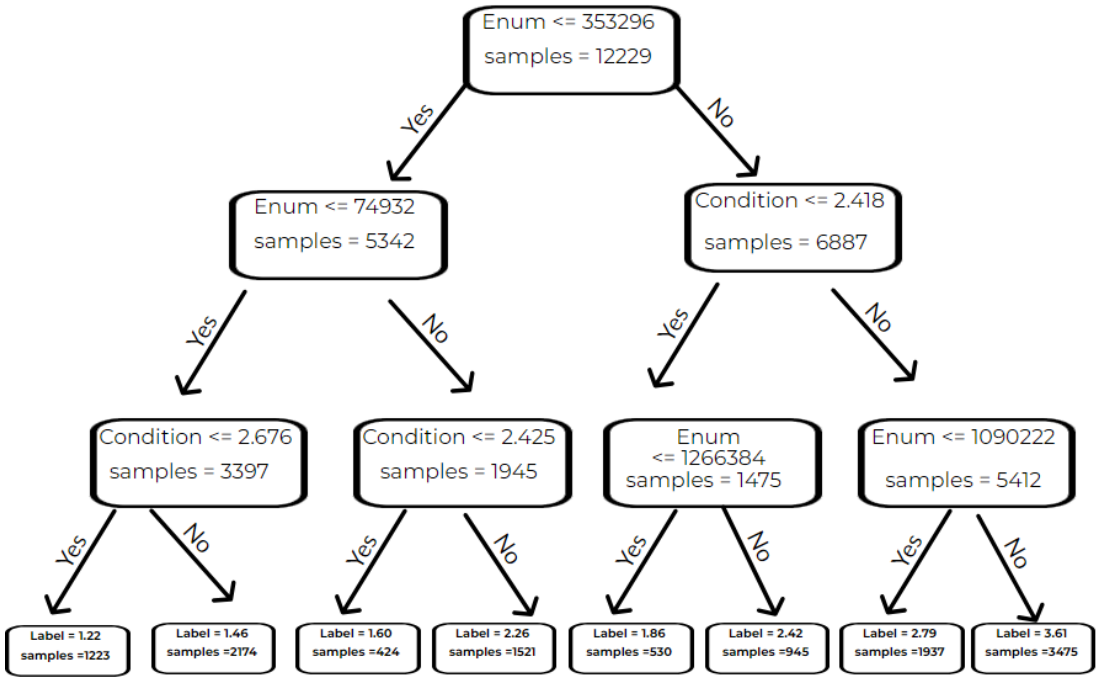


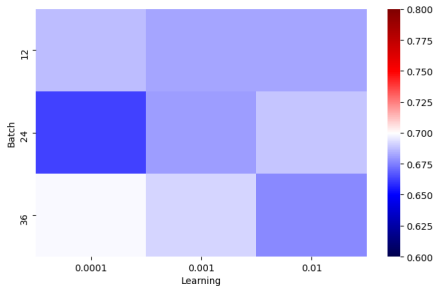
Figure 15: DT FI



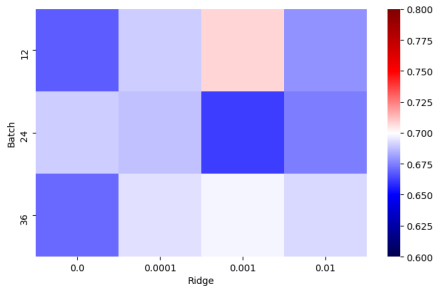
Vertex & Edge		Graph Structure		Bounds	
Notation	**Description**	**Notation**	**Description**	**Notation**	**Description**
V_{number}	number of vertices(n)	CondNum	Condition number of the distance matrix, obtained by Lanczos algorithm [27]	MST	Length of the minimum spanning tree, obtained using Prim's algorithm [39]
C^V_{min}	Lowest vertex cost	Rect	Rectangle in which the instance lies	Ordbound	Lower bound containing the shortest edges
C^V_{max}	Highest vertex cost	AvgCent	Mean distance from all point to the centroid	NNBound	Lower bound containing the nearest neighbours
C^V_{avg}	Average vertex cost	DistinctRatio	Fraction of distinct entries in the distance matrix		
C^V_{sd}	Standard deviation vertex cost	NumOutliers	Number of outliers, obtained using DBSCAN [15]		
C^V_{med}	Median vertex cost	NumClusters	Number of clusters, obtained using DBSCAN [15]		
E_{number}	number of edges				
C^E_{min}	Lowest edge cost				
C^E_{max}	Highest edge cost				
C^E_{avg}	Average edge cost				
C^E_{sd}	Standard deviation edge cost				
C^E_{med}	Median edge cost				

C k-fold validation NNet

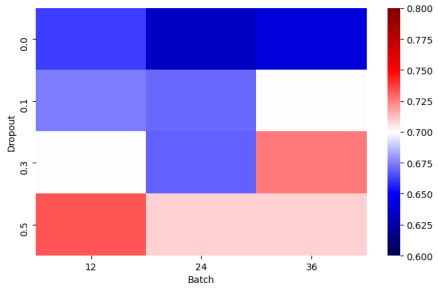
Figure 16: 5-fold cross-validation results



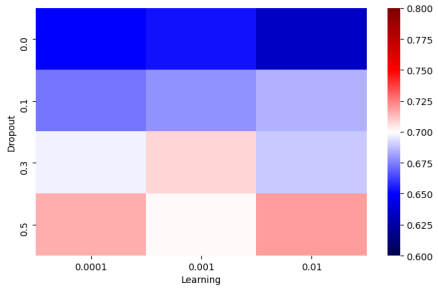
Batch size - Learning rate



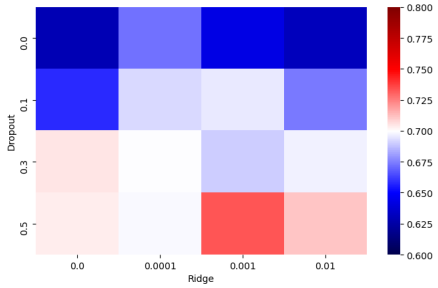
Batch size - Ridge parameter



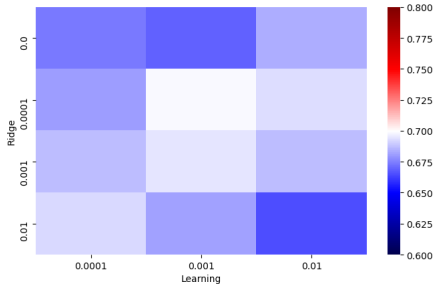
Batch size - Dropout rate



Dropout rate - Learning rate



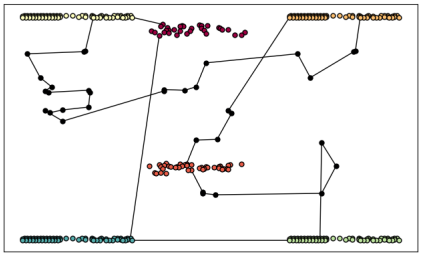
Dropout rate - Ridge parameter



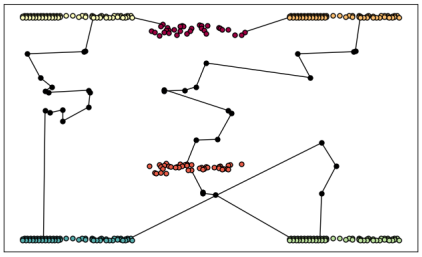
Learning rate - Ridge parameter

D SET TSP

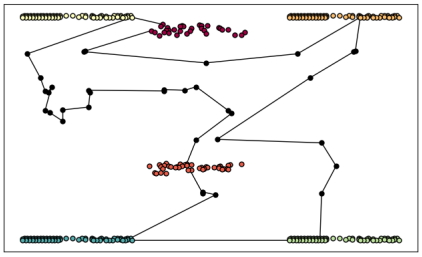
Figure 17: Set Heuristic Performances on fl417.tsp



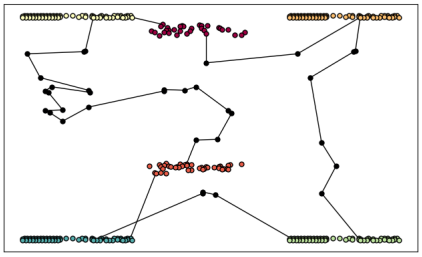
NN, length = 8985



GR, length = 7751



NI, length = 8054



FI, length = 7562

E Online appendix

All the heuristics, models, features, and the data generation process are implemented in Python and are accessible in the online appendix at the following link: https://github.com/668865MB/Meta-Learning-using-Neural-Networks-in-Large-Scale-Traveling-Salesman-Problems. The online appendix includes supplementary code that contains heuristics and models not included in the final version of this thesis for various reasons. Additionally, extensions of the heuristics from the traditional TSP case to the SET TSP and Bottleneck TSP are also provided. These extensions may prove useful in future studies.

References

[1] RD Angel, WL Caudle, R Noonan, and ANDA Whinston. Computer-assisted school bus scheduling. *Management Science*, 18(6):B–279, 1972.

[2] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.

[3] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007.

[4] Marco Caserta and Stefan Voß. A hybrid algorithm for the dna sequencing problem. *Discrete Applied Mathematics*, 163:87–99, 2014.

[5] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[6] Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[7] L da F Costa, Francisco A Rodrigues, Gonzalo Travieso, and Paulino Ribeiro Villas Boas. Characterization of complex networks: A survey of measurements. *Advances in physics*, 56(1):167–242, 2007.

[8] Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

[9] George Dantzig. *Linear programming and extensions*. Princeton university press, 1963.

[10] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[11] Corinne Feremans, Martine Labbé, and Gilbert Laporte. Generalized network design problems. *European Journal of Operational Research*, 148(1):1–13, 2003.

[12] Edgar C Fieller, Herman O Hartley, and Egon S Pearson. Tests for rank correlation coefficients. i. *Biometrika*, 44(3/4):470–481, 1957.

[13] Michael R Garey, David S. Johnson, and R Endre Tarjan. The planar hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976.

[14] Gregory Gutin and Abraham P Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

[15] Michael Hahsler, Matthew Piekenbrock, and Derek Doran. dbscan: Fast density-based clustering with r. *Journal of Statistical Software*, 91:1–30, 2019.

[16] Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[17] Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1(1):6–25, 1971.

[18] Laurent Herault. Rescaled simulated annealing—accelerating convergence of simulated annealing by rescaling the states energies. *Journal of Heuristics*, 6:215–252, 06 2000.

[19] John J Hopfield and David W Tank. "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

[20] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560, 1990.

[21] David S Johnson and Lyle A McGeoch. The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1(1):215–310, 1997.

[22] Jorge Kanda, Andre Carvalho, Eduardo Hruschka, and Carlos Soares. Using meta-learning to classify traveling salesman problems. pages 73–78, 2010.

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

[24] Jorge Kanda, Andre De Carvalho, Eduardo Hruschka, Carlos Soares, and Pavel Brazdil. Meta-learning to select the best meta-heuristic for the traveling salesman problem: A comparison of meta-features. *Neurocomputing*, 205:393–406, 2016.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[26] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[27] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. 1950.

[28] Jan Karel Lenstra and AHG Rinnooy Kan. Some simple applications of the travelling salesman problem. *Journal of the Operational Research Society*, 26(4):717–733, 1975.

[29] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[30] Mohd Asyraf Mansor, Siti Zulaikha Mohd Jamaludin, Mohd Shareduwan Mohd Kasihmuddin, Shehab Abdulhabib Alzaeemi, Md Faisal Md Basir, and Saratha Sathasivam. Systematic boolean satisfiability programming in radial basis function neural network. *Processes*, 8(2):214, 2020.

[31] Feng Mao, Edgar Blanco, Mingang Fu, Rohit Jain, Anurag Gupta, Sebastien Mancel, Rong Yuan, Stephen Guo, Sai Kumar, and Yayang Tian. Small boxes big data: A deep learning approach to optimize variable sized bin packing. In *2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 80–89. IEEE, 2017.

[32] Karl Menger. Untersuchungen über allgemeine Metrik. Vierte Untersuchung. Zur Metrik der Kurven. In *Selecta Mathematica: Volume 1*, pages 333–368. Springer, 2011.

[33] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[34] Chase C Murray and Amanda G Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54:86–109, 2015.

[35] Michael A Nielsen. *Neural Networks and Deep Learning*, volume 25. Determination Press, San Francisco, CA, USA, 2015. Pages 3-10, 77-79, 80-88, 154-163.

[36] Christian Nilsson. Heuristics for the traveling salesman problem. *Linkoping University*, 38:00085–9, 2003.

[37] Hoon Liong Ong and JB Moore. Worst-case analysis of two travelling salesman heuristics. *Operations Research Letters*, 2(6):273–277, 1984.

[38] Christos H Papadimitriou. The euclidean travelling salesman problem is NP-complete. *Theoretical computer science*, 4(3):237–244, 1977.

[39] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[40] H Donald Ratliff and Arnon S Rosenthal. Order-picking in a rectangular warehouse: a solvable case of the traveling salesman problem. *Operations research*, 31(3):507–521, 1983.

[41] Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

[42] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[43] Daniel J Rosenkrantz, Richard Edwin Stearns, and Philip M Lewis. Approximate algorithms for the traveling salesperson problem. In *15th Annual Symposium on Switching and Automata Theory (swat 1974)*, pages 33–42. IEEE, 1974.

[44] Alexander Schrijver. On the history of combinatorial optimization (till 1960). *Handbooks in operations research and management science*, 12:1–68, 2005.

[45] siemensnixdorf / Tilburg University. Optimale besturing van de boormachine bij de productie van printkaarten bij siemens nixdorf. Unpublished, 2021. Case study designed by Tilburg University.

[46] Kate Smith-Miles, Jano Van Hemert, and Xin Yu Lim. Understanding tsp difficulty by learning from evolved instances. In *Learning and Intelligent Optimization: 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers 4*, pages 266–280. Springer, 2010.

[47] Kate A Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):1–25, 2009.

[48] Charles Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 100(3/4):441–471, 1987.

[49] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[50] Shankar Vembu and Thomas Gärtner. Label ranking algorithms: A survey. In *Preference learning*, pages 45–64. Springer, 2010.

[51] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[52] Tong Zhao, Wei Jin, Yozen Liu, Yingheng Wang, Gang Liu, Stephan Günnemann, Neil Shah, and Meng Jiang. Graph data augmentation for graph machine learning: A survey. *arXiv preprint arXiv:2202.08871*, 2022.