Learning to Make Routing Decisions Based on Expert Examples

Job Schouten

2022.MME.8701

Report Cover Image:
Ham, B. (2021, December 10). Machine learning speeds up vehicle routing | MIT News | Massachusetts Institute of Technology. https://news.mit.edu/2021/machine-learning-speeds-vehicle-routing-1210

Learning to Make Routing Decisions Based on Expert Examples

by

Job Schouten

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday August 31, 2022 at 10:00 AM.

Student number:	4288262	
Project duration:	March 1, 2022 – August 31, 2022	
Thesis committee:	Dr. B. Atasoy,	TU Delft, Chair & Supervisor
	P. Zattoni Scroccaro,	TU Delft, Supervisor
	Dr. P. Mohajerin Esfahani,	TU Delft

TU Delft

Preface

This report results from nine months of work on my graduation assignment. However, it means much more to me than that. With this master's thesis, my wonderful time as a student at TU Delft also ends. I look back on the great results I achieved and a time in which I gained a lot of knowledge, not only related to Mechanical Engineering but also about myself. In addition, I have met interesting people and friends for life with whom I have had many great moments. All this has allowed me to grow into the person I am today, and now I am ready to face the next step in my career.

I want to thank my supervisors, Bilge Atasoy and Pedro Zattoni Scroccaro, for their time and effort in helping me during this project. Thanks to our many interesting discussions and brainstorming sessions, I was able to keep going even if I was sometimes stuck. In addition, you were always willing to go the extra mile to help me, and I really appreciated that. I would also like to thank my family and friends for all their support. And in particular, my fellow graduating friends with whom I spent almost daily the past few months and who have provided a lot of fun during this period.

Job Schouten
Delft, August 2022

Abstract

Logistics and mobility services play a major role in our society, and efficient routing is a crucial part of this. However, even though routing problems have been widely researched, the solutions provided by algorithms do not always match drivers' expectations. Routing costs used by these algorithms are often based on one or a few parameters, but in real-world operations, many factors and sometimes hard-to-define aspects are responsible for this. Drivers can consider these different aspects and some studies found that experienced drivers often plan better delivery routes than the optimization tools. In this research, we focus on using expert decision data as examples for learning the costs of routing and train a policy that can make decisions more in line with the expectations of the expert. We formulate state-action representations for the TSP and CVRP, which we use to interpret these routing problems as inverse optimization and multiclass classification problems. Additionally, we propose multiple policy training approaches as well as state feature vector transformations that can be used based on the characteristics of the routing problems. These different training configurations are utilized to train different existing algorithms with training data sets consisting of example state-action pairs. The performance of the trained models is compared to each other and the optimal solution. The optimal solution acted as expert example and was used to create the training data. We demonstrate that both inverse optimization and multiclass classification algorithms are able to imitate expert decision-making for new problem instances from example data. However, we also show a large variation in performance depending on the problem, state features, algorithm formulations and training configuration.

Contents

List of Figures

List of Tables

Nomenclature

$CVRP$	Capacitated Vehicle Routing Problem
IO	Inverse Optimization
MLP	Multi-layer Perceptron
OR	Operational Research
OvO	One-vs-One
OvR	One-vs-Rest
RBF	Radial Basis Function
RL	Reinforcement Learning
RNN	Recurrent Neural Networks
SV	Support Vector
SVM	Support Vector Machine
TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem

1 Introduction

Logistics and mobility services play a major role in our current society. Ordering packages, groceries, or dinner is becoming more popular, and, in most cases, we expect next or even same-day delivery. Even express orders are possible nowadays, promising delivery minutes after placing the order. Besides that, many people use mobility services, such as Uber or taxis, whenever they want and expect fast responses and transportation.

Vehicle routing is required to fulfil customer demand in all these cases, usually in an urban environment. Finding the best route to visit a set of locations is often referred to as a routing problem and can have different formulations. The two most common formulations of these routing problems are the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). In the TSP, a single vehicle must visit the set of locations and return to its starting point. A VRP is similar to a TSP. However, here multiple vehicles are used to visit the locations. In addition, constraints, such as time or capacity, often apply to the problem. These problems aim to determine a set of routes visiting all locations while minimizing the routing cost.

Many variations of these routing problems exist and have been researched, focusing on different costs or constraints for determining the optimal route. However, even though the TSP and VRP are intensively studied, the "theoretical optimal" solutions provided by routing algorithms do not always match the expectations of route planners or drivers. This mismatch is because the costs these algorithms minimize are often based on a few parameters, like distance or travel time, which have been studied extensively in the literature. In real-world operations, a good solution is influenced by several other factors that are also important. These factors may include road conditions, traffic lights, rush hours, facilities along a road and many others. Some aspects might even be challenging to define explicitly or may derive from the interdependencies among street segments. This large amount of different and sometimes hard-to-define aspects makes it complicated to estimate actual costs and incorporate them into an algorithm.

On the other hand, human drivers and route planners can take these different aspects into consideration, sometimes subconsciously, when determining a route. As a result, some studies found that experienced drivers often plan better delivery routes than the optimization tools (Canoy et al., 2021; Chen et al., 2021). In this research, we use this experience and expert decisions as examples. We use machine learning to learn the weights and costs of certain routing decisions to determine a policy that can mimic expert decision-making. The benefits of this would be that this policy and thus the model would be able to make better decisions that are more in line with the drivers' expectations. Furthermore, it would offer a way in which new drivers can also use and benefit from the knowledge of more experienced drivers.

The idea of learning from expert example data has been around for many years and mainly falls under supervised or imitation learning. Supervised learning is an approach where the algorithm is trained on input data labelled for a particular output. In line with this, imitation learning involves learning the optimum policy from expert demonstrations. Many supervised, and imitation learning approaches exist (Abbeel & Ng, 2004; Klein et al., 2012; Kotsiantis, 2007).

This research uses two approaches and interpretations for learning a policy to make routing decisions using expert examples. The first approach we use is inverse optimization (IO). In IO, one assumes that when given a signal, an expert solves an optimization problem to determine the action to take. The goal is to create an agent which can imitate this. However, this agent does not know the objective function and costs the expert uses to solve this problem. Therefore, the agent must learn this using a set of signals and corresponding expert actions.

We interpret the routing problem for the second approach as a multiclass classification problem. In multiclass classification, an algorithm is trained using a data set of samples and corresponding labels. Feature vectors often represent these samples. The goal is for the algorithm to learn which label belongs to a specific sample so that the algorithm can predict the correct label when presented with a new sample.

This study demonstrates how we can interpret a routing problem and its solution as a set of state-action pairs. We propose state-action representations for two separate routing problems with different characteristics and show how we can use these state-action formulations for our IO and classification approaches. Depending on the formulation of the problems, we provide multiple training strategies. Finally, we analyze and compare the performance of several existing solution algorithms for different problems regarding learning from routing examples and solving new routing problem instances.

In this research, we want to determine if we can use an IO and multiclass classification approach to learn the costs of a routing problem and train a policy which can then be used to solve new routing decisions by imitating expert decision-making. Therefore, the main question we try to answer is:

Can we imitate expert decision-making for routing problems by learning only from example decision data utilizing inverse optimization and multiclass classification solution approaches?

To answer this question, we consider related sub-questions like, how do we formulate routing problem states and corresponding actions? What is the performance of the trained models compared to the expert? Furthermore, how do the performance of IO and different multiclass classification approaches compare to each other when solving routing problems?

The remainder of this report is organized as follows. Chapter 2 reviews the relevant current research on IO and multiclass classification to routing and transportation problems. After that, Chapter 3 discusses the formulation of the different routing problems we consider in the study. The IO and multiclass classification approaches are discussed in Chapter 4 and Chapter 5, respectively. Chapter 6 discusses the numerical experiments and results, and finally, Chapter

7 concludes the findings of this research.

2 Literature Review

The origins of the TSP as a mathematical problem date back to the 1930s. However, it was not until 1954 that a breakthrough occurred when G. Dantzig et al. (1954) published a description of a method for the TSP by solving an instance with 49 nodes, an over-whelming size at that time. Soon after, the first formulation of the VRP was introduced by G. B. Dantzig and Ramser (1959) as the "Truck Dispatching Problem". This problem concerned itself with the optimum routing of a fleet of gasoline delivery trucks with a limited capacity between a bulk terminal and multiple service stations. Later, Clarke and Wright (1964) further generalized the problem forming the basis for the VRP as we know it.

Both the TSP and VRP are NP-hard combinatorial optimization and integer programming problems (Jünger et al., 1995; Toth & Vigo, 2002). This makes them interesting but also complicated to solve, and so since their introduction, research performed on both problems has grown rapidly (Cook et al., 2011; Eksioglu et al., 2009; Konstantakopoulos et al., 2020), and a wide variety of formulations focusing on different problems or constraints have been created. When considering the VRP, some of the most common examples are the Capacitated Vehicle Routing Problem (CVRP), where the vehicles have limited capacity and each request has a specific demand; the Vehicle Routing Problem with Time Windows, where each request must be served within a specific time frame; the VRP with Pick-up and Delivery, where goods have to be picked-up and delivered in specific amounts at the locations; or the Dial-A-Ride-Problem that involves transporting people between different locations (Pillac et al., 2013).

The TSP and VRP are among the most studied combinatorial optimization problems (Sharma et al., 2018). Because of this, many different solution methods have been proposed. Most of these are operations research-based (OR) methods. However, in recent years machine learning has been a popular approach for finding new ways to solve these routing problems.

On a high level, OR solution algorithms can be divided into three types: exact, heuristic, and metaheuristic, with each category containing multiple methods, as can be seen in Figure 2.1. The primary distinction between these methods is that exact methods solve the issue to the best possible or optimal answer, whereas (meta)heuristic solutions approximate the optimal solution. As a result, the solutions identified by these (meta)heuristic algorithms are not always the best solutions possible. However, (meta)heuristic approaches are frequently considerably faster and, therefore, more interesting than exact methods. Most studies on this aim to identify heuristic approaches that get as near to the optimal solution as possible in the shortest time. For more information on different approaches, the reader is referred to literature

(Elshaer & Awad, 2020; S. C. Ho et al., 2018; Konstantakopoulos et al., 2020; Oyola et al., 2017a, 2017b; Zhang et al., 2022).

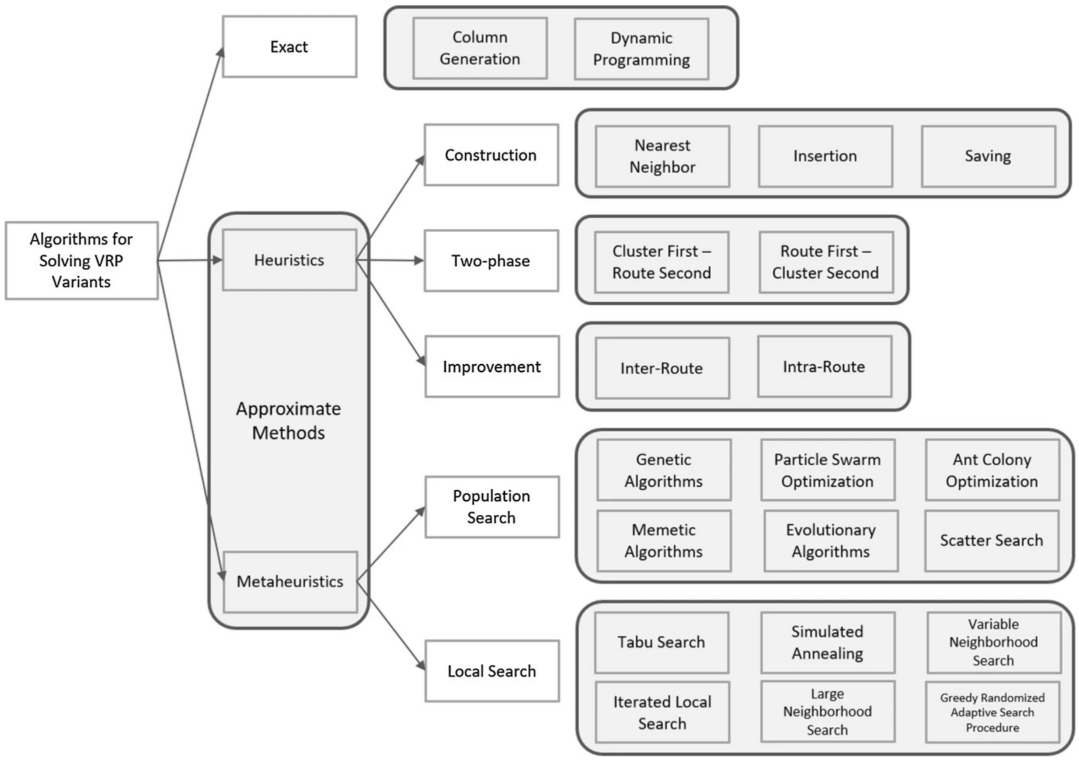

Figure 2.1: Classification of OR based algorithms for the VRP (Konstantakopoulos et al., 2020).

In recent years, machine learning has been a popular technique for solving routing problems, with (deep) reinforcement learning (RL) being one of the most popular approaches used, which learns by trying different actions and evaluating the response it receives from the environment (Nazari et al., 2018). Some early methods were proposed by Bello et al. (2016) and Vinyals et al. (2015), who introduced the concept of a Pointer Network. These models were based on recurrent neural networks (RNN) and inspired by sequence-to-sequence models (Luong et al., 2015; Sutskever et al., 2014), which have been extensively studied in the field of machine translation. These sequence-to-sequence models are useful if a mapping from one sequence to another is required. Although these methods provide good results on the TSP, using an RNN is restrictive when we try to solve more complicated problems for which the system's representation varies over time, such as with VRPs. In order to resolve this complication, Nazari et al. (2018) proposed an end-to-end framework that uses RL to solve VRPs based on pointer networks and attention mechanisms. After that, Kool et al. (2018) proposed a different model using RL entirely based on attention. Finally, another example is the "Learn to Improve" method proposed by H. Lu et al. (2020) for solving the VRP, which learns to iteratively refine the solution with an improvement operator selected by a reinforcement learning-based controller.

2.1. Inverse Optimization for Routing Problems

In the previous section, we mentioned different approaches for solving the TSP and VRP using operational research and machine learning. However, these approaches require a predefined cost that they will try to minimize. In real-world situations, however, this cost can depend on many aspects and may not be so easily defined. However, an IO approach can be used to determine unspecified parameters of an optimization problem that make a given observed solution optimal (Burton & Toint, 1992), possibly in the presence of some state-action constraints. The approach is particularly suitable when the action is an optimal decision for a certain unknown cost function (Akhtar et al., 2022) and can help to obtain more practical costs from previous experts' decisions. In this way, the optimization model for these costs could replicate the decision-making behaviour of the expert.

The interest in IO among the mathematical programming community started with the research of Burton and Toint (1992), which was the first to propose an IO approach for linear programming by investigating the inverse shortest path problem. This was soon followed by Ahuja and Orlin (2001) and Jianzhong Zhang and Zhenhong Liu (1996) in a more generalized context, who studied the inverse linear optimization based on optimality conditions and dual theory. IO models have also been researched in the context of integer programs (Schaefer, 2009; Wang, 2009) and network problems (Burton & Toint, 1992; Hochbaum, 2003; Jianzhong Zhang & Zhenhong Liu, 1996). Recently, research has expanded to include the use of IO when dealing with noisy and imperfect data (Aswani et al., 2015; Esfahani et al., 2015).

The applications of IO models can be found in a wide range of domains, like geoscience (Burton & Toint, 1992; Neumann-Denzau & Behrens, 1984), healthcare (Chan et al., 2014; Erkin et al., 2010), energy (Ratliff et al., 2013; Saez-Gallego et al., 2016), production planning (Troutt et al., 2006) and warehouse management (Rummukainen, 2021). Also, in the transportation area, IO is studied, for instance, in the context of cost estimations related to collaboration among carriers in liner shipping (Agarwal & Ergun, 2010; Zheng et al., 2015). However, concerning routing problems, little research on applying an IO approach can be found in literature.

One of the studies found is a recent paper by Chen et al. (2021) on the use of an IO approach for CVRPs. They propose an IO formulation based on dualization to derive a cost matrix by learning from the decisions of experienced drivers (experts). In this way, the goal is that the routing model, with respect to the learned costs, could provide solutions as good as those given by experts. To train the model, Chen et al. (2021) follow the example of earlier research (Bärmann et al., 2018; Bärmann et al., 2017) and propose an online learning approach.

Two other studies on the topic of routing were performed by You et al. (2016) and Xu et al. (2018). You et al. (2016) proposed an urban freight modelling framework that considers both spatial-temporal constraints, based on an adaptation of an activity-based passenger model called the household activity pattern problem (Chow & Recker, 2012; Recker, 1995). The problem is modelled as a custom VRP, and an IO approach was used to update coefficients of different objectives to observed data such as GPS traces. The models' application was demonstrated by developing and executing a custom Sequential Selective Vehicle Routing Problem to show how to anticipate drainage truck activity using GPS trajectories to infer operational parameters. Xu et al. (2018) proposed an IO model to learn the parameters of heterogeneous travellers' route behaviour to infer shared network state parameters, such as link capacity dual prices, in real-time. Parameter learning in their model is accomplished by solving a set of inverse shortest route problems with an invariant common prior.

Finally, a real-world example of a routing system using IO is given by Rönnqvist et al. (2017). They describe an online route recommendation system for long-haul truck drivers in the Swedish forest industry. This system, which has been in operation since 2009, finds the best route, depending on road features and driver preferences, when multiple competing objectives are involved. One of its distinctive features is the use of an IO approach that reduces a loss, balancing various factors by using more than 100 weights. The system has resulted in a change from manually to automatically determined routes that are collectively established by all stakeholders, facilitating standardization, collaboration, and reduced costs.

2.2. Multiclass Classification for Routing Problems

Supervised classification algorithms aim at producing a learning model from a labelled training set. In other words, the objective is to create a simple model of the distribution of class labels in terms of predictive features. The derived classifier is then used to assign class labels to testing instances where the predictor feature values are known, but the class label value is unknown. Simple classification problems frequently use only binary labels, whereas multiclass classification problems use more than two labels, only one of which can be given to a collection of features. Various classification variations and approaches have been developed throughout the years (Chih-Wei Hsu & Chih-Jen Lin, 2002; Kotsiantis, 2007).

The applications of (multiclass) classification can be found in a wide range of domains, like ecology (Bourel & Segura, 2018), food chemistry (Berrueta et al., 2007), and healthcare (Akbar et al., 2020; J. Lu et al., 2005). For example, classification is used in this last domain to aid in diagnoses decisions (Ramaswamy et al., 2001). For the domain of transportation, multiclass classification is also used. However, most of these applications focus on categorising data or predicting already established routes. As is the case in two studies by Duca et al. (2017) and Lo Duca and Marchetti (2020), they use multiclass classification to predict ship routes.

For the applications on vehicle routing, only one recent study by Mandi et al. (2021) was found using a form of classification for vehicle routing. In this research, the writers formulate a multiclass classification problem where they define the next stop in a route as the class of the set of visited stops. They used a set of previous routing data to train a neural network model which learns the transition probabilities between locations in a CVRP problem formulation. With these transition probabilities, they solve the maximum likelihood routing problem instead of the conventional VRP to make the routing decisions. To the best of our knowledge, this is the only study that implements (multiclass) classification for solving routing problems.

2.3. Summary

In summary, most of the research on applying machine learning to solve routing problems has been done in the field of (deep) reinforcement learning. Using supervised learning or similar approaches where the routing costs are unknown, and the algorithm has to learn from examples has been comparatively less researched. IO and multiclass classification are both approaches that have already been explored in many different contexts. However, this is not the case in the area of routing problems. Our research is an early step in this, in which we investigate their application to two different routing problems and suggest different training and learning methods from routing examples. We then test these approaches and compare the performance of various existing algorithms for solving these problems. We discuss the results and propose future possibilities for learning to solve routing problems from examples.

3 Routing Problems

A TSP or VRP is generally defined on a graph $G = (V, E)$, with vertex set $V = \{v_0, ..., v_n\}$ and edge set $E = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$, where v_0 often represents the depot, and the other vertices in V are the requests that need to be served. The goal of these routing problems is to determine, with a single or multiple vehicles, a set of routes such that each vertex (except the depot) is visited exactly once while solving an objective function and satisfying constraints. A common example of this is minimizing the overall routing cost.

In this research, we define two different routing problems for which we will test the algorithms we use. The first problem we refer to as the TSP, and the second problem we refer to as the CVRP. Details and the formulation of each problem are described below. The main difference between both problems is that in the TSP, only one vehicle is used to visit all the nodes. For the CVRP, multiple vehicles are used to visit the nodes. However, the vehicles only have a limited capacity, and each node has a specific demand that must be fulfilled. These demands remain the same but are redistributed across the nodes for each new problem instance. Both problem formulations will be evaluated with fixed customer locations, where the nodes are in the same place for each new problem instance, and with random customer locations, where the node locations are different in each problem instance. Figure 3.1 illustrates an example of a TSP and CVRP solution.

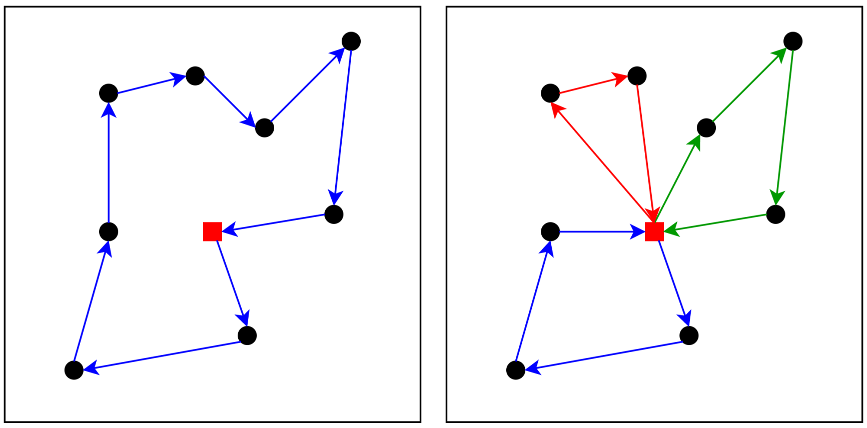

Figure 3.1: Examples of a TSP (left) and CVRP (right) solution.

We defined these two routing problems with their characteristics for multiple reasons.

These problems are simplified representations of route problems we encounter daily in the real world. We already argued in the introduction that learning to imitate experts' decisions can have advantages. In addition, the difference in characteristics regarding the fixed and random locations allows us to test two different scenarios in which we learn and apply experts' preferences.

The random locations for each new problem instance ensure that the costs of different routing choices also vary according to the problem. Therefore, this scenario considers a universal use for learning to make routing decisions based on more general preferences. The fixed locations, however, are similar to a more area-specific application. Because only a limited network is considered to learn and route inside, the edges, and the routing costs, stay constant. The focus here is on developing a policy for routing decisions in a specific location, such as a neighbourhood or city, where a model is trained solely on routing decisions from this area.

As we mentioned before, for the algorithms we use to solve these problems, we will not provide the objective function or specify the routing costs used by the expert to determine the routes. Instead, we only provide a data set containing N routing examples, which is used to train the algorithms to make routing decisions. This data set consists of multiple state-action pairs $\{\hat{s}_i, \hat{u}_i\}_{i=1}^{N}$. To create these state-action pairs, we assume that a route, the solution to the routing problem, is formulated using a series of actions or decisions u. Each of the decisions being which node the vehicle will visit next. The state s represents the system state of the problem as it was when making the decision and is formulated as a feature vector. An example of this is illustrated in Figure 3.2. In this system we consider the set of nodes $V = [0, 1, 2, 3, 4]$. In the first system state, the vehicle is located at the depot. This state is translated into a feature vector s_1 for which the next node the vehicle will visit represents the corresponding action u_1. After the vehicle has travelled to this node, this is the next system state. Different features are used depending on the problem, which is discussed below.

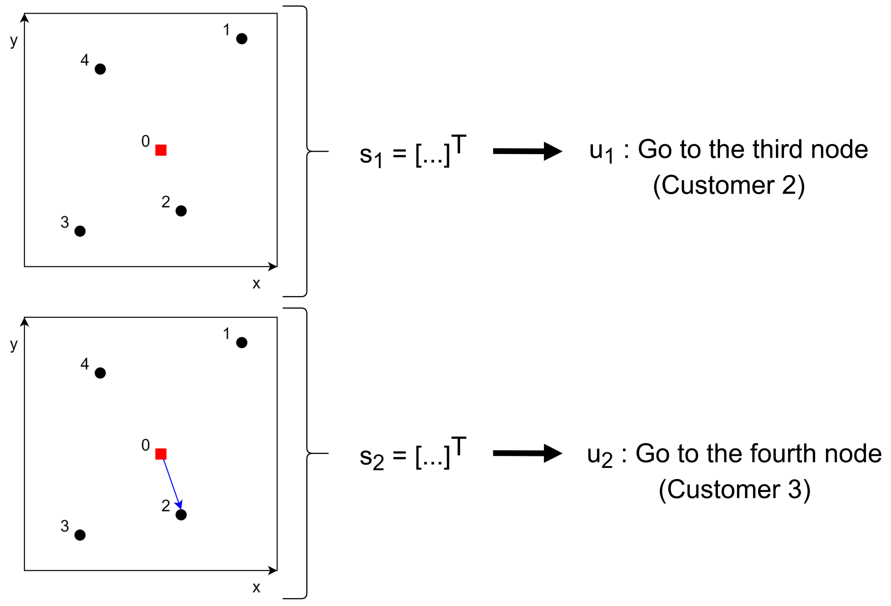
y
x
0
1
2
3
4
$s_1 = [...]^T$
u_1 : Go to the third node (Customer 2)
$s_2 = [...]^T$
u_2 : Go to the fourth node (Customer 3)

Figure 3.2: Two consecutive state-action pair examples: A system state is represented by a feature vector s and the next node the vehicle will visit is the corresponding action u.

3.1. Traveling Salesman Problem

In the TSP a single vehicle must visit a set of vertex $V = \{0, \dots, N\}$ consisting of N nodes, for which vertices $i = 1, \dots, N$ correspond to customers and vertex 0 corresponds to the depot. The vehicle will start at the depot and must also return to the depot after visiting all the customers. Furthermore, all customers may only be visited once, and the goal of the problem is to minimize the total routing cost. $C = \{c_{ij}, i, j \in V\}$ is the cost matrix, in which c_{ij} is the travel cost from node i to node j, which is equal to the euclidean distance between the nodes. The following mathematical formulation describes the problem.

Mathematical Formulation

Decision variable:

x_{ij} Equal to 1 if a vehicle travels directly from vertex i to vertex j; 0, otherwise. $i, j \in V$

Formulation:

$$\begin{aligned} \text{minimize} \quad & \sum_{i,j \in V} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{j \in V} x_{ij} = 1 \quad \forall i \in V \\ & \sum_{i \in V} x_{ij} = 1 \quad \forall j \in V \\ & \sum_{i,j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subset V \\ & x_{ij} \in \{0, 1\} \quad \forall i, j \in V \end{aligned}$$

To formulate a state s used in the data set, we need to define what information we can use as features for describing the system's state at any point in the problem for every instance. The features that were selected for this are the x-coordinates and y-coordinates of every node, the information if a node has already been visited or not and, in case of the random customer locations also, the distances of the node that the vehicle is located at that point to all the other nodes and the distances between all nodes and the depot. In the following example, we will explain how the feature vector is constructed.

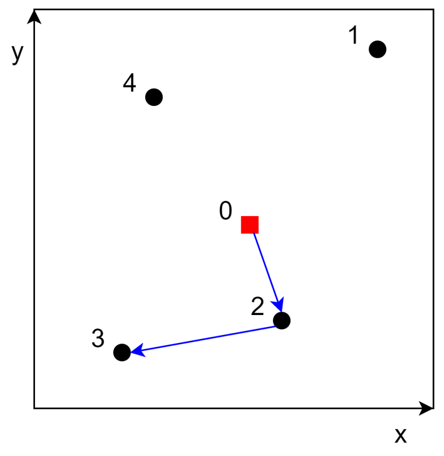

Figure 3.3: TSP state example with four customers (black dots), depot (red square) and traveled route of the vehicle (blue arrows)

Consider Figure 3.3, which illustrates a system state we must represent as a feature vector. This system consists of 5 nodes, of which four customers and the depot are randomly located on a graph. The vehicle started at the depot and followed the partial route visiting customers 2 and 3, respectively. Therefore, currently, the vehicle is located at customer 2. To describe this system, we use the features mentioned above. We represent the information on if a node is already visited or not as an array of ones and zeros, where zero means a node is visited and one means the node is not visited and thus is still available. This results in the following features and corresponding feature vector:

x-coordinates:	$[x_0, x_1, x_2, x_3, x_4]^\top$
y-coordinates:	$[y_0, y_1, y_2, y_3, y_4]^\top$
distance to current node:	$[d_{20}, d_{21}, 0, d_{23}, d_{24}]^\top$
distance to depot:	$[0, d_{01}, d_{02}, d_{03}, d_{04}]^\top$
available nodes:	$[1, 1, 0, 0, 1]^\top$

Feature vector:

$$[x_0, x_1, x_2, x_3, x_4, y_0, y_1, y_2, y_3, y_4, d_{20}, d_{21}, 0, d_{23}, d_{24}, 0, d_{01}, d_{02}, d_{03}, d_{04}, 1, 1, 0, 0, 1]^\top.$$

There are multiple ways to approach the training of a decision-making model for solving the routing problem. The first and most straightforward way is to train one decision-making policy for the entire statespace $\mathbb{S}$. In this case, the same policy is used to make a decision, no matter the state of the problem. However, another option would be to train different policies depending on the number of available nodes left in the system. To do this, we divide the data set of state-action pairs used for training in n_d individual data sets, which we use to learn a policy for every number of available nodes left. Note that we do not need to learn a policy when only a single customer is left or when the vehicle has visited all the customers and has to return to the depot.

Figures 3.2 and 3.3 show that the customers in a specific system are arbitrarily numbered, starting with the number 1. However, the numbering of the customers has no relation to the routing choices and only serves as a reference for interpreting the decisions made by the algorithm. This is illustrated in Figure 3.4.

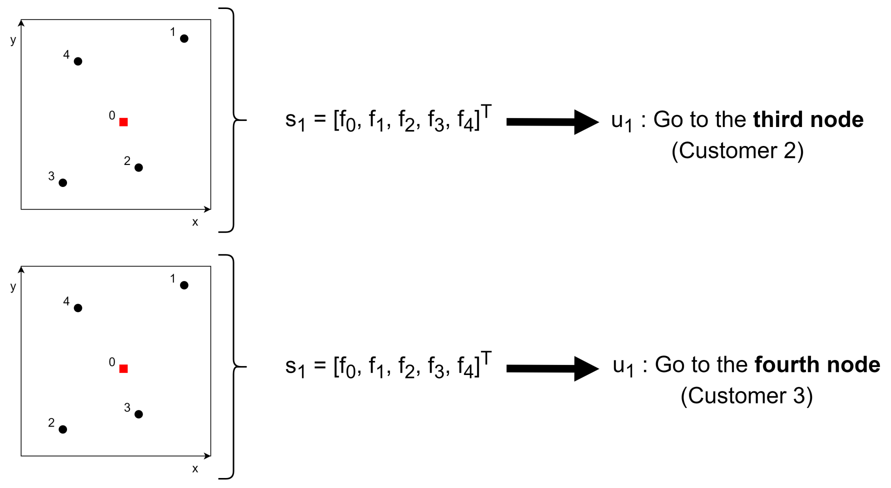
y
x
0
1
2
3
4
$s_1 = [f_0, f_1, f_2, f_3, f_4]^\top$
u_1 : Go to the **third node** (Customer 2)
$s_1 = [f_0, f_1, f_2, f_3, f_4]^\top$
u_1 : Go to the **fourth node** (Customer 3)

Figure 3.4: Relation between node labels and decisions. For the same system different labeling outputs a different decision for the same routing location.

In this example, we consider the same system, represented by a simplified feature vector s only consisting of a single feature f per node, with different labelling of the customers. As a result of the different labelling, the corresponding feature vector is different for the same order of the nodes, resulting in different actions for travelling to the same location. So as we can see, the labelling will not influence the location the vehicle will travel. This allows us to change the order of the second feature vector so that both system's states and actions are represented the same, as shown in Figure 3.5.

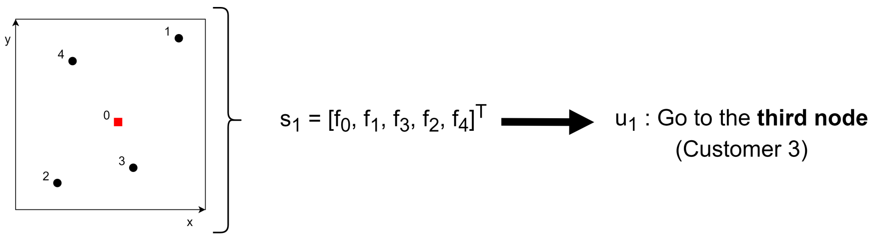
y
1
4
0
3
2
x
$s_1 = [f_0, f_1, f_3, f_2, f_4]^T$
u_1 : Go to the **third node**
(Customer 3)

Figure 3.5: Reordering of the feature vector results in different decision.

We can use this information to reorder and reduce the feature vector depending on the current system state and previous decisions. The main benefit of this is that it can decrease the computational time needed for both training and testing of the algorithm and reduce the variety of possible actions depending on the nodes left to visit. In this next part, we demonstrate how we apply the reordering and reducing for the feature vector we proposed above.

Example: Feature Vector Reordering and Reducing

Consider the same system as above, however now we will label the nodes in the system with letters, where the first node is still the depot now label with letter Z and the other nodes the customers labeled A to D. The node order of this problem corresponds to $[Z, A, B, C, D]$, see Figure 3.6.

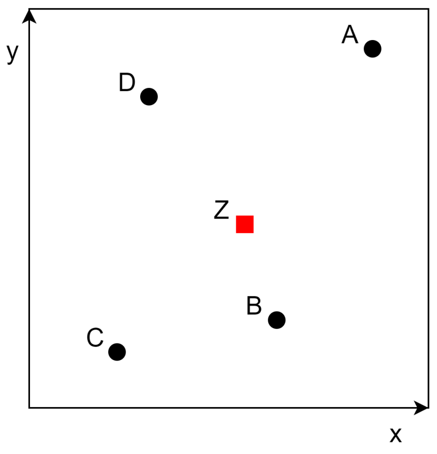

Figure 3.6: TSP state example with four customers (black dots), depot (red square)

In the first decision, the vehicle moves to the third node in this system, which using the node order, corresponds to customer B. After this, we now have a new state for which a feature vector can be constructed. However, we reorder the node order and, thus, the features so that the nodes that were already visited are moved to the end of their respective feature group. If we do this, the node order now corresponds to $[Z, A, C, D, B]$ and the new feature vector for this state will be:

Feature vector:
$[x_Z, x_A, x_C, x_D, x_B, y_Z, y_A, y_C, y_D, y_B, d_{BZ}, d_{BA}, d_{BC}, d_{BD}, 0, 0, d_{ZA}, d_{ZC}, d_{ZD}, d_{ZB}, 1, 1, 1, 1, 0]^\top$.

As we can see, the features corresponding to customer B, indicated in red, have now moved. It is important to note that the node order in the feature vector is now changed, so the decisions made depending on the state will not correspond to our original node labels, as we will now show. Therefore, it is important to keep track of the performed changes. Depending on this new feature vector, the next decision is to move to the third node. This is the same as our first decision; however, because we reordered the node order and feature vector, the third node corresponds to customer C. After this decision, we can now reorder the node order again, resulting in $[Z, A, D, B, C]$ with the feature vector for the new state:

Feature vector:
$[x_Z, x_A, x_D, x_B, x_C, y_Z, y_A, y_D, y_B, y_C, d_{CZ}, d_{CA}, d_{CD}, d_{CB}, 0, 0, d_{ZA}, d_{ZD}, d_{ZB}, d_{ZC}, 1, 1, 1, 0, 0]^\top$.

The result of this reordering is that the available nodes and their features will always be in front of the node order and their corresponding feature parts. Therefore, the decision will always be to go to one of the first nodes in the order, reducing the variation in actions depending on the number of available nodes. Suppose we now consider the approach of training a different policy depending on the number of available nodes left. In that case, we can argue that we do not need to include the features of nodes that have already been visited before the current node. So we can reduce the length of the feature vector, which will consequently reduce the computational time needed. Reducing the last feature vector will result in:

Feature vector:
$[x_Z, x_A, x_D, x_C, y_Z, y_A, y_D, y_C, d_{CZ}, d_{CA}, d_{CD}, 0, 0, d_{ZA}, d_{ZD}, d_{ZC}, 1, 1, 1, 0]^\top$.

Note that reduction of the feature vector can only be applied when training multiple decision-making policies depending on the number of available nodes left and not in the case of training a global decision-making policy because of the varying vector size.

3.2. Capacitated Vehicle Routing Problem

In the CVRP a number of vehicles K with a limited capacity Q is used to visit a set of vertex $V = \{0, \dots, N\}$ consisting of N nodes, for which vertices $i = 1, \dots, N$ correspond to customers and vertex 0 corresponds to the depot. The vehicles will start at the depot and must also return to the depot at the end of their route. Furthermore, all customers have a demand $q_i, i \in V \backslash \{0\}$ and may only be visited once. The goal of the problem is to minimize the total routing cost. $C = \{c_{ij}, i, j \in V\}$ is the cost matrix, in which c_{ij} is the travel cost from node i to node j, which is equal to the euclidean distance between the nodes. The following mathematical formulation describes the problem.

Mathematical Formulation

Decision variable:

x_{ijk} Equal to 1 if a vehicle k travels directly from vertex i to vertex j; 0, otherwise. $i, j \in V$, $k \in [K]$

Formulation:

$$
\begin{aligned}
\mathit{min} \quad & \sum_{k\in[K]} \sum_{i,j\in V} c_{ij}x_{ijk} \\
\mathit{subject\,to} \quad & \sum_{k\in[K]} \sum_{j\in V} x_{ijk} = 1 \quad \forall i \in V\backslash\{0\} \\
& \sum_{j\in V} x_{ijk} = \sum_{j\in V} x_{jik} \quad \forall i \in V,\, k \in [K] \\
& \sum_{j\in V\backslash\{0\}} x_{0jk} = 1 \quad \forall k \in [K] \\
& \sum_{i\in V} \sum_{j\in V\backslash\{0\}} q_j x_{ijk} \le Q \quad \forall k \in [K] \\
& \sum_{i,j\in S} x_{ijk} \le |S| - \left\lceil \frac{1}{C} \sum_{j\in S} q_j \right\rceil \quad \forall S \subset V\backslash\{0\},\, k \in [K] \\
& x_{ijk} \in \{0,1\} \quad \forall i,j \in V,\, k \in [K]
\end{aligned}
$$

Like with the TSP, we formulate a vector using multiple features to represent the system's state. However, because of the differences between the problem, we also use different features. For example, in this problem, we have to deal with multiple vehicles routing simultaneously. Furthermore, we also have the customers' demands and the vehicle's capacities, which are both important and change depending on the problem instance and state.

The first features we use are the vehicle for which we have to make the current decisions, its current capacity and location. The vehicle that has travelled the smallest distance is selected to move next. To indicate which vehicle this is, we use a one-hot encoding. This means that we use a vector with an equal length to the number of vehicles, where every element of the vector is equal to zero except for the considered vehicle, which is equal to one. For example, in the case of 3 vehicles, and we are considering the second vehicle, the one-hot encoding results in $[0,\, 1,\, 0]$. We use the same approach of one-hot encoding to represent the location of the current vehicle, where we use a vector with the length equal to the number of nodes of which the element corresponding with the current node is equal to one. The other features we use are the current capacities of all vehicles, the current location of all vehicles again in a one-hot encoding, the initial demand of each customer and which customer the vehicle can visit depending on if the customer has already been visited and the vehicle still has enough capacity to serve the demand. Furthermore, in the case of the random customer locations, we include the x-coordinates and y-coordinates of every node, the distances of the node where the current vehicle is located at that point to all the other nodes and the distances between all nodes and the depot.

Like with the TSP, there are multiple ways in which we can approach the training of a decision-making model for solving the CVRP as well. We can again use the so-called global policy training approach, where we are training one decision-making policy for the entire statespace $\mathbb{S}$. However, the fixed node locations also allow us to take a node-orientated training approach. This means that we train and use different policies depending on the node where the vehicle is located. To do this, we divide the data set of state-action pairs used for training in n_n individual data sets, which we use to learn a policy for every individual node in the system.

3.3. Data Set Creation

Data sets are generated in advance for both problems that we use in this research to evaluate the performance of the different models. Both a training data set and a test data set are generated for each problem, containing a predetermined number of different problem instances. In the case of the training data set, these problem instances need to be transformed into state-action pairs. To do this, we determine the optimal route by solving each problem instance using Gurobi (Gurobi Optimization LLC, 2022). The optimal solution will be used as the expert routing examples we use for training. We split each optimal route into individual actions, with each new node in the route being an action. Then we determine the corresponding feature vectors representing the state for each action and combine these in the data set, which will be provided to our model for training.

Now that we have defined the problems, different learning approaches and described how the data sets of state-action pairs are constructed to train our models. We discuss the different algorithms that will be used in this study.

4 Inverse Optimization

As was mentioned before, in IO an agent aims to learn the behaviour of an expert who makes decisions based on an exogenous signal, which in this research is represented by the state of the routing problem. To do this, it is assumed that upon receiving a state $s \in \mathbb{S} \subseteq \mathbb{R}^n$, the expert optimizes a parametric optimization problem over a set of feasible actions $\mathbb{U}(s) \subseteq \mathbb{R}^m$, which also depend on the system state s. This optimization problem is formulated as

$$\min_{u \in \mathbb{U}(s)} F(s, u), \tag{4.1}$$

where $F : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$. In this optimization problem, we assume that for every state $s \in \mathbb{S}$, the set of minimizers $\mathbb{U}^*(s) := \arg\min_{u \in \mathbb{U}(s)} F(s, u)$ is non-empty. Furthermore, we assume that the agent has no prior knowledge of the objective function F representing the expert's preferences. Thus the agent is unable to predict the experts response u^{ex} to a particular system state s prior to training. The agent does however has access to a data set consisting on N states which are paired with corresponding expert actions $\{\hat{s}_i, \hat{u}_i^{\text{ex}}\}_{i=1}^N$, for which,

$$\hat{u}_i^{\text{ex}} \in \arg\min_{u \in \mathbb{U}^*(\hat{s}_i)} F(\hat{s}_i, u) \quad \forall\, i \in [N].$$

This state-action data, indicated with the hat notation $\hat{\cdot}$, can be used to learn the experts objective function. To do this, we assume that the unknown objective function $F(s, u)$ can be represented by a parameterized function $F_\theta(s, u)$, which is part of a parametric hypothesis space $\mathcal{F} = \{F_\theta : \mathbb{S} \times \mathbb{U} \to \mathbb{R} \mid \theta \in \Theta\}$, where $\theta \in \Theta$ represents the parameters to be learnt. The mapping $\theta \mapsto F_\theta$ and the choice of space Θ depend on the problem at hand. Using this we can formulate the optimization problem that the agent tries to solve as

$$\min_{u \in \mathbb{U}(s)} F_\theta(s, u). \tag{4.2}$$

Optimizing this for a system state $s \in \mathbb{S}$ as input will return an action $u \in \mathbb{U}$. This output action is considered as the agent action $u_\theta^{\text{ag}}(s)$ with

$$u_\theta^{\text{ag}}(s) := \arg\min_{u \in \mathbb{U}(s)} F_\theta(s, u).$$

4.1. Loss Functions

Ideally, the agent would aim to identify a hypothesis closely resembling the behaviour of the expert. To do this, the available data set $\{\hat{s}_i, \hat{u}_i^{\text{ex}}\}_{i=1}^N$ is used to training the agent so it can learn the parameters in the hypothesis. For this training a loss function ℓ_θ is used to quantify the inaccuracy of the parameter used in a objective function $F_\theta \in \mathcal{F}$. The goal during training would be to find the parameter θ which minimize the total loss. We can formulate this optimization problem as

$$\min_{\theta \in \Theta} \ell_N(\theta) := \min_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \ell_\theta(\hat{s}_i, \hat{u}_i^{\text{ex}}), \tag{4.3}$$

where $\ell_\theta(s, u) : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$.

Different loss functions are discussed in literature and existing IO models mainly differ based on their specification of the loss function (Akhtar et al., 2022). A computationally attractive loss function in the context of IO is the suboptimality loss (Esfahani et al., 2015) which is formulated as

$$\ell_\theta^{\text{sub}}(s, u^{\text{ex}}) := F_\theta(s, u^{\text{ex}}) - \min_{u^{\text{ag}} \in \mathbb{U}(s)} F_\theta(s, u^{\text{ag}}). \tag{4.4}$$

For the candidate hypothesis, the suboptimality loss quantifies the difference between expert and agent decisions. The object function in the hypothesis space that best describes the expert action given an external input, regardless of the real cost incurred by the agent, is found via minimization of this loss function. It is worth noting that the loss is only zero when the expert action is the cost minimizer for the hypothesized cost, and the loss is explicitly dependent on the system state s.

While in some situations the suboptimality loss is very useful, Zattoni Scroccaro et al. (2021) showed that this is not always the case. Particularly in many practical applications, and hence they introduced the generalized suboptimality loss as

$$\ell^{\text{gsub}}(s, u^{\text{ex}}(s)) := F_\theta(s, u^{\text{ex}}) - \min_{u^{\text{ag}} \in \mathbb{U}(s)} \left\{ F_\theta(s, u^{\text{ag}}) - \gamma d(u^{\text{ex}}, u^{\text{ag}}) \right\}, \tag{4.5}$$

with $\gamma \geq 0$ and where the distance function $d(u^{\text{ex}}, u^{\text{ag}}) > 0$ if $u^{\text{ex}} \neq u^{\text{ag}}$ and $d(u^{\text{ex}}, u^{\text{ag}}) = 0$ if $u^{\text{ex}} = u^{\text{ag}}$. As the name suggests the generalized suboptimality loss can be used for a more wider range of problems like when solving a linear program. Therefore, in this research we will be using this loss in the context of our IO model. Furthermore, for the distance function we apply the 0-1 distance $d(u^{\text{ex}}, u^{\text{ag}}) = I(u^{\text{ex}}, u^{\text{ag}})$ with $I(u^{\text{ex}}, u^{\text{ag}}) = 0$ if $u^{\text{ex}} = u^{\text{ag}}$, else $I(u^{\text{ex}}, u^{\text{ag}}) = 1$ (Zattoni Scroccaro et al., 2021).

When combining the generalized suboptimality loss (4.5) and the optimization problem (4.3) for a given data set $\{\hat{s}_i, \hat{u}_i^{\text{ex}}\}_{i=1}^N$, the training phase of the IO approach results in the optimization program

$$\min_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \left(F_\theta(\hat{s}_i, \hat{u}_i^{\text{ex}}) - \min_{u_i^{\text{ag}} \in \mathbb{U}(\hat{s}_i)} \left\{ F_\theta(\hat{s}_i, u_i^{\text{ag}}) - \gamma I(u^{\text{ex}}, u^{\text{ag}}) \right\} \right). \tag{4.6}$$

4.2. Modeling Choices

We use the formulation presented by Zattoni Scroccaro et al. (2021). The parameterized objective function $F_\theta(s, u)$ is defined as

$$F_\theta(s, u) := s^\top Q u, \tag{4.7}$$

where $\theta = \text{vec}(Q)$, meaning that the parameter vector θ can be reshaped into a parameter matrix $Q \in \mathbb{R}^{n \times m}$.

Now that we have defined our hypothesis function we can combine this with problem (4.2) and formulate the optimization problem that our IO agent tries to optimize as

$$\min_{u \in \mathbb{U}(s)} s^\top Q u. \tag{4.8}$$

The parameter vector θ is found using the generalized suboptimality loss by solving optimization problem (4.6). This is done by reformulating the problem as a single minimization problem as demonstrated by Zattoni Scroccaro et al. (2021). It is important to mention that in our research the constraint set of decisions $\mathbb{U}(s)$ consists of a finite number of discrete elements. Because of this a auxiliary variable β_i can be introduced for each state-action pair in our training data set, and we can use the so-called epigraph reformulation of the problem (Boyd & Vandenberghe, 2004). Furthermore, a regularization term $\mathcal{R}(\theta) = \|\theta\|_2^2$ and parameter $\lambda > 0$ was added to penalise the magnitude of θ. Resulting in the single minimization problem formulated as

$$\begin{aligned} \min_{\theta, \beta_1, \ldots, \beta_N} \quad & \lambda \|\theta\|_2^2 + \frac{1}{N} \sum_{i=1}^{N} \beta_i \\ \text{s.t.} \quad & \left\langle \theta, \phi(\hat{s}_i, \hat{u}_i^{\text{ex}}) \right\rangle - \left\langle \theta, \phi(\hat{s}_i, u^{ag_i}) \right\rangle + \gamma d(\hat{u}_i^{ex}, u_i^{\text{ag}}) \leq \beta_i \quad \forall u_i^{\text{ag}} \in \mathbb{U}(\hat{s}_i),\ i \in [N] \\ & \theta \in \Theta. \end{aligned} \tag{4.9}$$

Finally, in our implementation, we use Gurobi (Gurobi Optimization LLC, 2022) to solve optimization problem 4.9 for a training data set to obtain the parameter vector θ. This vector is then rescaled to the parameter matrix Q, which is used with optimization function 4.8 to formulate the agent's decisions when testing the model.

5 Multiclass Classifications

A classification problem in machine learning refers to a predictive modelling problem where a class needs to be predicted for a given sample. To solve this problem, a model will determine the optimal way to map input data samples to certain classes using the training data set. As such, the training data set must be sufficiently representative of the problem and have several examples for each class. There are several different types of classification algorithms which have been developed. Many algorithms use binary classification, which refers to classification tasks containing two classes.

On the other hand, a multiclass classification problem is a problem in which there are more than two classes, and each sample can only be classified as one of them. Not all classification predictive models support multiclass classification. However, algorithms designed for binary classification can be adapted to solve multiclass problems with the One-vs-Rest (OvR) or One-vs-One (OvO) methods (see Appendix B).

This research will use different multiclass classification algorithms to evaluate and compare their performance in solving the TSP and VRP. The algorithms that we use are described in the sections below.

5.1. Support Vector Machines

A support vector machine (SVM) is a binary classification algorithm used for multiclass classification using the OvO approach. The SVM classifier works by determining a hyperplane that separates the two classes of the input samples. The dimensions of this hyperplane are equal to the number of features of the input sample. These hyperplanes act as decision boundaries for the classification of new input samples. Samples on either side of the hyperplane will be classified into the corresponding class. There are usually several potential hyperplanes from which to choose. However, the classifier attempts to locate a plane with the greatest distance between data points from both classes, called the margin. Maximising the margin allows new data points to be categorised with more certainty. The support vectors (SV) are samples closest to the hyperplane and impact the hyperplane location and orientation, see Figure 5.1. The position of the hyperplane will change if the support vectors are removed.

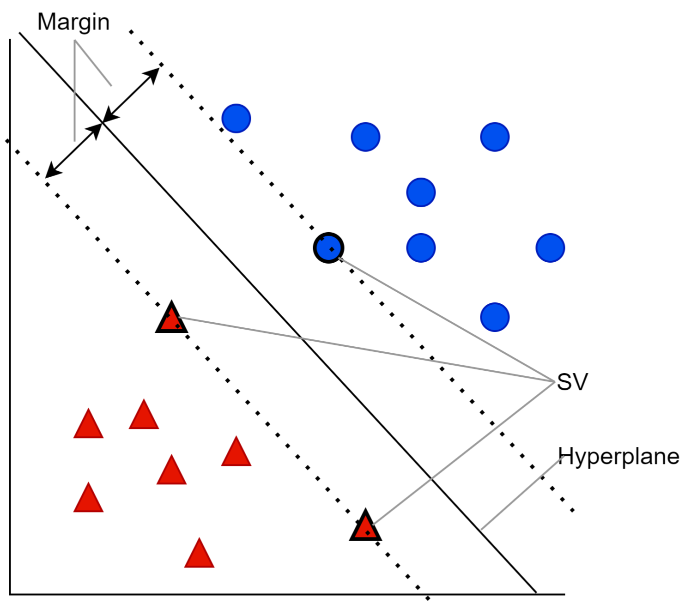

Figure 5.1: Support Vector Machine hyperplane

Depending on the features, there are many cases in which it is impossible to separate the input samples using a hyperplane in the current dimensions (the number of which is equal to the number of features) as most data is not linearly separable. We can use feature transformation to map the sample vector into a higher dimensional space using a function ϕ to solve this problem. However, the downside of this is its high computational complexity. To deal with this, we can use the so-called kernel trick. It allows us to operate in the original feature space without computing the coordinates of the data in a higher dimensional space. To better explain this, we use the following example.

Consider two data points in 2 dimensions, which we need to map to a 4-dimensional space:

$$a = [a_1,\, a_2]^\top,$$
$$b = [b_1,\, b_2]^\top.$$

Mapping these features results in:

$$\phi(a) = [a_1^2,\, a_1 a_2,\, a_2 a_1,\, a_2^2]^\top,$$
$$\phi(b) = [b_1^2,\, b_1 b_2,\, b_2 b_1,\, b_2^2]^\top.$$

If we would calculate the dot product of these mappings we end up with:

$$\begin{aligned}\phi(a)^\top \phi(b) &= \sum_{i,j=1}^{2} a_i a_j b_i b_j \\ &= (a_1 b_1 + a_2 b_2)^2 \\ &= (a^\top b)^2 \\ &= K(a, b).\end{aligned}$$

The function $K(a, b)$ is called the kernel function of which different formulations exist. Some of the most popular kernel functions are:

$$\begin{aligned}
\text{linear:} \quad & K_{\text{lin}}(a,b) = a^\top b \\
\text{polynomial:} \quad & K_{\text{poly}}(a,b) = (\gamma(a^\top b))^d \\
\text{radial basis function (rbf):} \quad & K_{\text{rbf}}(a,b) = e^{-\gamma\|a-b\|^2} \\
\text{sigmoid:} \quad & K_{\text{sig}}(a,b) = \tanh(\gamma(a^\top b)),
\end{aligned}$$

where d is the degree and γ influence parameter. This parameter defines how far the influence of a single training sample reaches.

We now discuss the formulation of the SVM classifier (Cortes & Vapnik, 1995). Given a data set of training samples $\{\hat{x}_i, \hat{y}_i\}_{i=1}^N$ for which $x_i \in \mathbb{R}^n\ \forall i \in [N]$ and $y_i \in \{-1,1\} \forall i \in [N]$ then the prediction is given by

$$y = \text{sign}(w^\top \phi(x) + b), \tag{5.1}$$

where $w \in \mathbb{R}^n$ is the weight and $b \in \mathbb{R}$ is the bias. To determine the weight, the following optimization problem is solved:

$$\begin{aligned}
\min_{w,b,\zeta} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^N \zeta_i \\
\text{s.t.} \quad & \hat{y}_i(w^\top \phi(\hat{x}_i) + b) \le 1 - \zeta_i \quad \forall i \in N \\
& \zeta_i \le 0 \quad \forall i \in [N],
\end{aligned} \tag{5.2}$$

where ζ_i is the distance of a sample from its margin boundary and C is a regularization parameter.

In order to be able to use the kernel functions we described above, we need to determine the dual of the above optimisation problem. This is formulated as

$$\begin{aligned}
\min_{\alpha} \quad & \frac{1}{2}\alpha^\top D\alpha - e^\top \alpha \\
s.t. \quad & y^\top \alpha = 0 \\
& 0 \le \alpha_i \le C \quad \forall i \in [N],
\end{aligned} \tag{5.3}$$

where D is an N by N positive semidefinite matrix. With our formulation of the kernel functions we can define this matrix as $D_{ij} = \hat{y}_i\hat{y}_j K(\hat{x}_i, \hat{x}_j)$. Furthermore,$e$ is a vector of all ones and α is called the dual coefficient vector, which is used in the classification of new samples. The prediction is now given by:

$$y = \sum_{i \in SV} \hat{y}_i \alpha_i K(\hat{x}_i, x) + b, \tag{5.4}$$

where SV is a set of all the support vectors.

5.2. Multi-Layer Perceptron

Multi-Layer Perceptron (MLP) is a supervised learning algorithm that uses a feed-forward neural network structure and can be used for multiple tasks, including classification. The network consists of three types of layers: the input layer, output layer and hidden layer. An example of an MLP is illustrated in Figure 5.2.

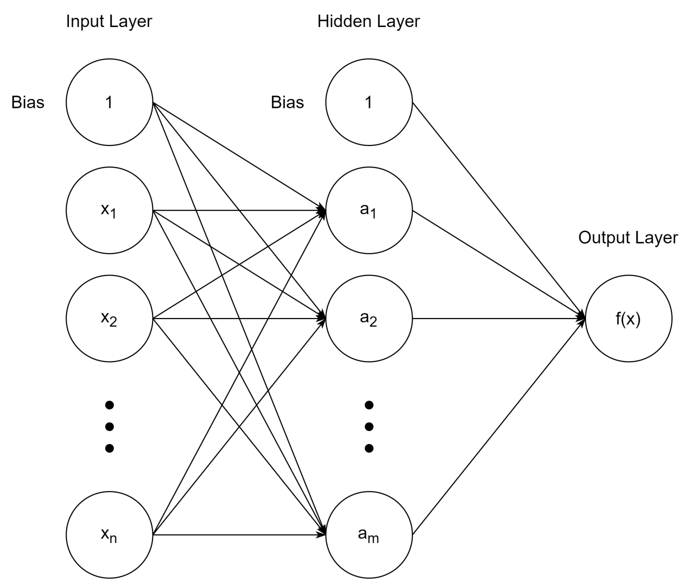

Figure 5.2: Multi-Layer Perceptron Network

The input layer consists of a set of neurons, each representing an element from the state's feature vector x used as the input signal to be processed. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation, followed by a non-linear activation function $g(\cdot) : R \rightarrow R$, like the ReLU of the hyperbolic tan function. Between the input and output layers are one or more layers of hidden neurons. Their name refers to the fact that these neurons are not directly reachable either from the input or output. The output layer receives the values from the last hidden layer and transforms them into output values. For Figure 5.2, where there is one hidden layer, we can represent the model as:

$$f(x) = W_2 g(W_1^T x + b_1) + b_2. \tag{5.5}$$

If there are more than two classes, $f(x)$ would be represented by a vector of size N_{classes} The required task, which in this case is classification, is performed by this output layer. For multiclass classification, a softmax function is applied as output function.

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{c=1}^{N_{\text{classes}}} \exp(z_c)}, \tag{5.6}$$

where z_i represents the ith element of the input to softmax in this case $f(x)$, which corresponds to class i, and N_{classes} is the number of classes. The result is a vector y containing the probabilities that state x belongs to each class. The output is the class with the highest probability.

The weights used by the neurons in the MLP are trained using stochastic gradient descent (SGD) (Rumelhart et al., 1986). SGD updates parameters using the gradient of the loss function with respect to a parameter that needs adaptation. In this case the cross-entropy loss is used which for $N_{\text{classes}} > 2$ is formulated as (Y. Ho & Wookey, 2020)

$$l(\hat{y}, y) = -\frac{1}{N} \sum_{c=1}^{N_{\text{classes}}} \sum_{i=1}^{N} \hat{y}_{i,c} \log(y_{i,c}), \tag{5.7}$$

where N is the number of training examples and $\hat{y}_{i,c}$ is the cth element of one-hot vector $\hat{y}_i$ of length N_{classes} for which the correct class is equal to one and y_i is the probability vector output by the MPL. Furthermore also, an L2-regularization term that penalises complex models is added to the loss to avoid over-fitting, resulting in

$$L_W(\hat{y}, y, W) = l(\hat{y}, y) + \frac{\alpha}{2N}\|W\|_2^2, \tag{5.8}$$

where $\alpha > 0$ is a non-negative hyperparameter that controls the magnitude of the penalty. With the SGD, the weight is updated as

$$W^{t+1} = W^t - \epsilon \nabla L_W^t, \tag{5.9}$$

where t is the iteration step and $\epsilon > 0$ is the learning rate of the model. The algorithm stops when it reaches a preset maximum number of iterations; or when the improvement in loss is below a certain predefined number.

MLP was shown to have multiple advantages over standard classification algorithms (Benediktsson et al., 1990; Gardner & Dorling, 1998). The first and most important benefit is that no prior assumptions about the distribution of training data are needed. Furthermore, another advantage of the MLP approach is that no decision about the relative importance of the various input features is required, the weights are adjusted during training to choose the most discriminating inputs. Benediktsson et al. (1990) suggested that with care, the traditional classification algorithms can be used to classify more accurately than an MLP. However, this requires considerably more insight and effort from the analyst using the algorithms.

5.3. Random Forest and Decision Tree

A random forest is a meta estimator that trains several decision tree classifiers on different sub-samples of the data set and applies averaging to classify given samples, improving predicted accuracy and controlling over-fitting when compared to a single decision tree classifier (Breiman, 2001).

Decision Tree

Decision trees are a non-parametric supervised learning method for classification. The objective is to build a model that predicts the class of a sample based on basic decision rules derived from data features. This is accomplished by formulating true/false statements using the features and repeatedly splitting the data set until all data samples belonging to each class are separated. The data is therefore structured in a tree form. A node is added to the tree for each statement. The first node is referred to as the root node. Following each statement, the data set is divided, and new nodes are created based on the value of a feature. Each split will create new branches and segments the feature space into disjoint regions (James et al., 2013). All data points for which the statement is true are represented by one branch of the split. The other branch represents the remaining data points. As a result, the feature space narrows with each split in the tree, and each sample belongs to just one section. The aim is to keep splitting the feature space and applying statements until there are no more statements to apply or the maximum tree size is reached. The final nodes formed are known as leaf nodes. A leaf node corresponds to a particular class. Alternatively to assigning a specific class, the probability of each class in a leaf node, which is the fraction of training samples of the class in a leaf, can be output.

In a decision tree, the algorithm attempts to divide the data set into the smallest subset possible for each split. Just as with other machine learning methods, this is done by minimising a loss function. Because data samples from different classes are separated, the loss function should assess a split by comparing the proportion of data samples from each class before and after the split. To do this, Gini impurity is used as the loss function to compare the class distribution before and after the split (Breiman et al., 2017). This loss is a measure of variance across the different classes and is formulated as

$$l_{\text{node}} = \sum_{c=1}^{N_{\text{classes}}} p_c(1 - p_c), \tag{5.10}$$

where N_{classes} is the number of classes and p_c the probability of picking a sample from class c.

When using decision trees, it is important to take into account the possibility of over-fitting, which occurs when over complicated trees are generated that do not generalise the data correctly. To prevent this problem, mechanisms such as pruning, setting the minimum number of samples required at a leaf node, and limiting the tree's depth are required. Furthermore, decision trees can be unstable since little changes in the training data might result in the generation of an entirely different tree. However, this can be reduced by using multiple decision trees inside an ensemble, such as random forest.

The random forest algorithm is a perturb-and-combine technique designed explicitly for trees (Breiman, 1998). This means that a varied group of classifiers is formed by manipulating the data distribution and changing the generation method. As a result, there is more variation in the models. Furthermore, forests produce decision trees with somewhat decoupled prediction errors. Ensembles containing a more diversified group of trees typically result in higher prediction accuracy. The ensemble prediction is given as the average of the individual classifier predictions.

5.4. Modeling Choices

To use classification algorithms for solving the TSP and CVRP, we need to interpret these routing problems as classifications problem. The system's state is described as a feature vector in the routing problems. We consider this as the sample which needs to be classified. The next node on the route is determined depending on the system's state. Therefore, we can consider this node as the class belonging to the system state. Using this approach, the number of nodes determines the number of classes in the classification problem.

Important to note is that the classification algorithms we formulated above do not necessarily consider the constraints that apply to the problem we want to solve. For example, this can be the case when considering the CVRP we formulated above. Here the classifiers could easily violate the capacity constraints and choose a next node for the vehicle to visit while it has not enough capacity to serve its demand. To make sure that no constraints are violated, we validate the suggested action before it is output, where if the classifier would suggest going to a node that violates any constraint and is therefore infeasible, the closest feasible node will be output instead.

To conclude, we described three distinct classification methods. The first is SVM, a binary classification technique that may be used to solve multiclass classification problems using the OvO approach. As a second algorithm, we described the MLP, which utilises a feed-forward neural network structure to determine the probability of a state belonging to a specific class, and finally, the random forest approach, which employs several different decision trees and averages of individual classifier predictions to determine the class of a sample. In the next chapter, we will compare the performance of these approaches with each other and the IO method we formulated before.

6 Numerical Experiments

In this chapter, we will go through the numerical experiments we conducted to evaluate the performance of the different algorithms we discussed earlier. As previously stated, the algorithms were evaluated on two different routing problems, the TSP and CVRP. The specifics of these problems are provided below. To solve both problems, eight different algorithms were considered, which are as follows:

1. Inverse Optimization (IO)
2. Support Vector Machine with linear kernel (SVM Linear)
3. Support Vector Machine with polynomial kernel (SVM Polynomial)
4. Support Vector Machine with rbf kernel (SVM Rbf)
5. Support Vector Machine with sigmoid kernel (SVM Sigmoid)
6. Multi-layer Perception (MLP)
7. Random Forest
8. Decision tree

As previously stated, the objective of both routing problems is to minimize the routing cost, which is equal to the total routing distance. A training data set was prepared in advance for both problems. We used Gurobi (Gurobi Optimization LLC, 2022) to find the optimal solutions for many problem instances, which were then used to create the state-action pairings. All the algorithms were trained using the same data set for each problem. During testing, we compare the performance of the algorithms to each other and to the optimal solutions we want them to mimic. Furthermore, because we use distance as a routing cost, we also compare the findings to a greedy solution approach to evaluate performance.

Python 3.7 was used to implement all algorithms and conduct all experiments. As previously indicated, the IO model was implemented using Gurobi (Gurobi Optimization LLC, 2022). In the IO model we used a regularization parameter $\lambda = 1$ and $\gamma = 1$. Besides that, we included a constant term equal to 1 at the beginning of the feature vector as the bias term. The other multiclass classification algorithms were implemented using Scikit-learn 1.1.1 (Pedregosa et al., 2011). In the case of the SVM we used regularization parameter $C = 1$, degree $d = 3$ for the polynomial kernel, and influence parameter $\gamma = 0.5$. Furthermore, in the MLP, 100 hidden layers were used, and the Random Forest consisted of 100 decision trees. All other parameters were left at their default settings in Scikit-learn 1.1.1.

6.1. Traveling Salesman Problem

The TSP was formulated as we detailed in Chapter 3. For evaluating the different models, we considered a TSP consisting of 10 nodes V, a single depot and nine customers. The x-coordinates and y-coordinates of the customer locations are in the range $[0, 1]$. In every problem instance, the depot location was fixed, positioned at the point (0.5,0.5) in the middle of the feasible location region. Furthermore, the distances between nodes are equal to the euclidean distances.

For the training of the models, we evaluated different approaches using a global policy and multiple policies depending on the number of available nodes left in the system, which we refer to as a decisions policy. Both policy types were combined with either no feature vector transformations or reordering. In the case of the decisions policy, the reducing feature vector technique was also applied, as discussed in Chapter 3. The models were trained in an offline batch manner, with each batch including 10 problem instances. It is worth noting that each problem instance is made up of 10 state-action pairs. The training data set had a total of 500 problem instances.

Training Time

We also looked at the training time for the different configurations to evaluate the models with different policies and feature vector transformations. Table 6.1 shows the different training times for the IO model during the experiments performed to obtain the results presented in section 6.1.1.

Table 6.1: Training times of the IO model for different policy training configurations during the experiments presented in Table 6.2. Trained over 500 TSP training instances.

Training Policy	**State Feature Vector Transformation**	**Training Time IO Model (s)**
Global	None	6854
Global	Reorder	2050
Decisions	None	2592
Decisions	Reorder	2661
Decisions	Reduce	2487

The training times show that the reordering feature vector transformations greatly reduce training time when using the global policy approach. For the decisions policy, however, we see that this is less so, where all training times are somewhat equal, with the reordering being the highest. However, although these results suggest some benefits with regard to training time for the reordering and reducing feature vector techniques, it must be mentioned that these results might not be entirely reliable. This is because the experiments were performed on an HP zBook Studio G5 - intel core i7 device which was also used for other activities during the experiments. We noticed these activities strongly influenced the training time (see Appendix C). This makes it hard to draw any firm conclusions concerning the difference in training time between the training configurations, and further research is needed to confirm these findings.

6.1.1. Results

As mentioned in Chapter 3, the models were trained and tested on a TSP with fixed customer locations and a TSP with random customer locations. The performance in these problem variations was evaluated by solving 200 different TSP instances and calculating the average routing cost.

For the TSP with fixed customer locations, we found that all models were able to imitate the expert, which is the optimal solution exactly (see Appendix D). Note that when we consider the TSP with fixed locations, every individual problem instance in training as well as in testing is the same. Therefore, this result is expected, showing that all models can learn how to solve a single TSP problem instance from an example.

We find more varying, and therefore interesting, results when focusing on the TSP with random customer locations. Table 6.2 provides an overview of the models' performances for each training configuration. The average optimal cost of these test instances, which the models are trying to imitate, is equal to 2.83, and the average routing cost using a greedy approach is equal to 3.08.

Table 6.2: TSP (random locations): Average routing cost per model for different policy training configurations. Trained over 500 TSP training instances and evaluated over 200 TSP test instances.

		Average Routing Cost						**Optimal**	2.83
								Greedy	3.08
Training Approach	**State Feature Vector Transformation**	**IO**	**SVM Linear**	**SVM Polynomial**	**SVM Rbf**	**SVM Sigmoid**	**MLP**	**Random Forest**	**Decision Tree**
Global	None	3.36	**3.05**	3.10	3.19	4.13	3.00	3.28	3.55
Global	Reorder	3.38	3.06	**3.02**	**3.08**	3.90	**2.97**	**3.13**	**3.44**
Decisions	None	3.49	3.33	3.20	3.35	**3.75**	3.18	3.39	3.64
Decisions	Reorder	3.35	3.12	3.09	3.13	3.77	3.03	3.23	3.57
Decisions	Reduce	**3.33**	3.11	3.09	3.11	3.77	3.01	3.21	**3.44**

When analyzing these results, we find that for each training configuration, the best performance is obtained by the MLP model, with its overall best performance using the global policy training approach with feature vector reordering. This global policy training with feature vector reordering approach obtained, for almost all models, their individual best performance, except for the IO, SVM Linear and SVM Sigmoid models. In the case of the IO model, the best performance was obtained by the decisions policy training with reducing feature vector. The best performance for the SVM Linear model resulted from the global policy training without any feature vector transformation. Interestingly, the best performance of the SVM Sigmoid model was obtained using the decisions policy training approach without any feature vector transformations. This approach, however, resulted in the worst individual performances of all other models. Besides that, the average cost of the SVM Sigmoid model was still higher than that of all the other models, making it the worst-performing model overall.

Comparing the results of the global and decisions policy training approach, we can see that for most models, except for the IO and SVM Sigmoid models, the decisions policy training approach does not increase performance. Furthermore, considering the feature vector transformations, we observe that for the global and the decisions policy training approach, the performance of almost all models increases when reordering, and in the case of the decisions policy reducing, the state feature vector compared to using no transformation.

As we mentioned, using the distance as routing cost also allows us to compare the model performances to using a greedy solution approach. In the current results, we see that the SVM Linear, SVM Polynomial and MLP models all outperform this greedy approach using the global policy training with feature vector reordering, with the SVM Linear and MLP models also obtaining better results in some other training configurations.

Figure 6.1 shows the development of the average overall routing cost for each model during training using the global policy training with the feature vector reordering approach. After each training batch, the performance of the models was evaluated over 200 different TSP test instances. Figure 6.2 visualizes the performance after training (see Table 6.2), along with the standard deviation of each model for this training configuration compared to the average optimal and greedy routing costs. We see that the results are consistent with the different model performances at the end of training, as we would expect. Most models' standard deviation is similar to that of the optimal and greedy approaches, with the exceptions of the IO, SVM Sigmoid and Decision Tree models, which are larger.

Because the average cost development of all training configurations is relatively similar, we only focus on the global policy with the feature vector reordering approach in this section, as this training approach obtained the best performances for most models. The average cost development, as well as the plotted results from Table 6.2 for all other training configurations, are provided in Appendix E.

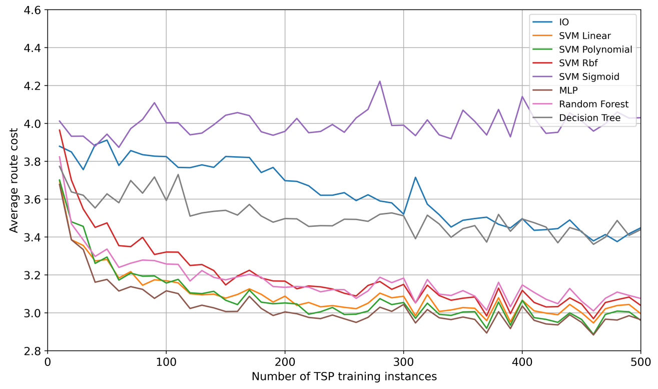

Figure 6.1: TSP (random locations): Cost development during training using global policy training and feature vector reordering. Evaluated over 200 TSP test instances after each batch of 10 TSP training instances.

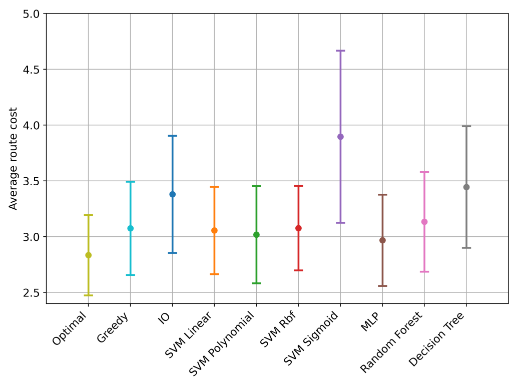

Figure 6.2: TSP (random locations): Average routing costs with standard deviation per model after training over 500 TSP instances using global policy training and feature vector reordering. Evaluated over 200 TSP test instances.

Examining the data in Figure 6.1, we observe some interesting similarities and differences between the average cost development during the training of the models. The majority of the models follow a similar development curve, where the average routing cost decreases rapidly during the first 200 TSP training instances, after which this decline slows down and almost stabilizes after about 400 TSP training instances. There are three models, however, for which the development of the average cost is different. For both the IO and Decision Tree models, average costs also decrease. However, the curves follow a slower and more steady rate than the other models. Furthermore, for these two models, the decline in the average routing cost does not seem to stabilize at the end of training, which indicates that they may reduce further when more training instances are available. Finally, the average routing cost of the SVM Sigmoid model shows no improvement during training.

Influence of Training Size and Distance Features

As we mentioned before, for some models, the average routing costs stabilized around a certain value at the end of the training. However, this is not the case for all models, and the performance of these models may be further improved. To gain a better insight into this, we repeated the experiment using global policy training with feature vector reordering with a larger training data set. In addition, it is important to note that for a certain state, the distances from the current node to every other node are included in the feature vector, which means that the cost of each possible action for this given state is included in the feature vector since the routing costs are equal to the distances. To investigate how this affects the performance of the different models, we also performed an additional experiment using global policy training with feature vector reordering, in which the distances between nodes are not included in the state feature vectors. We used a training data set of 1000 TSP instances for both experiments, training in batches of 10 instances. The results of the model's performances with and without the distances included in the feature vector are provided in Table 6.3 and in Figure 6.3 and Figure 6.4, respectively.

Table 6.3: TSP (random locations): Average routing cost per model for global policy training and feature vector reordering. Evaluated over 200 TSP test instances.

		Average Routing Cost						**Optimal**	2.83
								Greedy	3.08
TSP Training Instances	**Distances Included in Feature Vector**	**IO**	**SVM Linear**	**SVM Polynomial**	**SVM Rbf**	**SVM Sigmoid**	**MLP**	**Random Forest**	**Decision Tree**
500	Yes	3.38	3.06	3.02	3.08	3.90	2.97	3.13	3.44
1000	Yes	3.26	3.02	2.97	3.02	3.90	2.96	3.09	3.41
1000	No	3.79	3.66	3.09	3.09	3.93	2.99	3.10	3.45

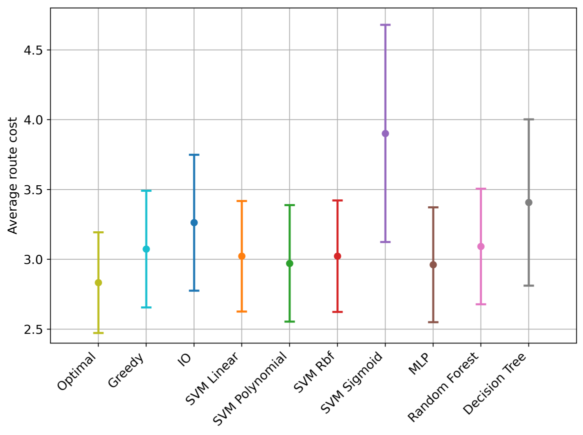

Figure 6.3: TSP (random locations): Average routing costs with standard deviation per model after training over 1000 TSP instances using global policy training and feature vector reordering. Distances included in the state representation. Evaluated over 200 TSP test instances.

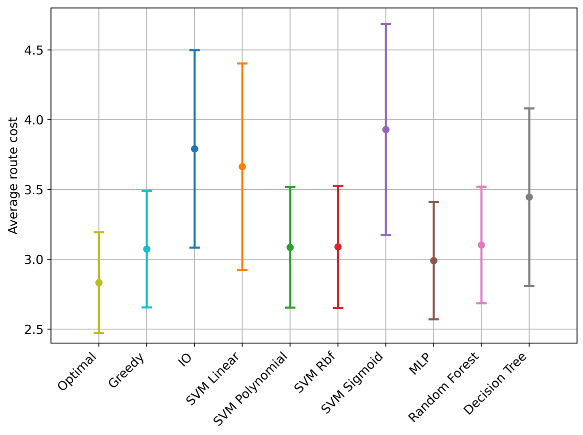

Figure 6.4: TSP (random locations): Average routing costs with standard deviation per model after training over 1000 TSP instances using global policy training and feature vector reordering. Distances not included in the state representation. Evaluated over 200 TSP test instances.

First, comparing the results of the experiment with a training size of 500 TSP instances to that of the larger training data set including 1000 instances, both with distances included in the feature vector, we see that for all but one model, the average routing cost has decreased with a larger training size. The exception is the SVM Sigmoid model, for which the performance is the same. This matches our previous observation that the SVM Sigmoid model is not improving during training. The most considerable improvement was made by the IO model, which is also in line with our previous findings, as the IO model still had the highest rate of decline at the end of training in Figure 6.1.

If we now look at the influence of including the distance in the state feature vector, we see that the performances of all models lower when the distance is not included in the state representation. This is especially the case for the IO and SVM Linear models, which perform considerably worse than they did when the distances were included in the feature vector. This means that the actions of both these models were highly dependent on these features and, thus, the knowledge of the costs for specific actions. Removing the distance from the state representation had the lowest impact on the MLP and Random Forest models. Both these models obtained almost equal performance in both experiments, with the MLP model still outperforming the greedy routing strategy without the distances.

Figure 6.5 and Figure 6.6 show the development of the average routing cost for each model for the experiments with and without the distances included in the state feature vector, respectively.

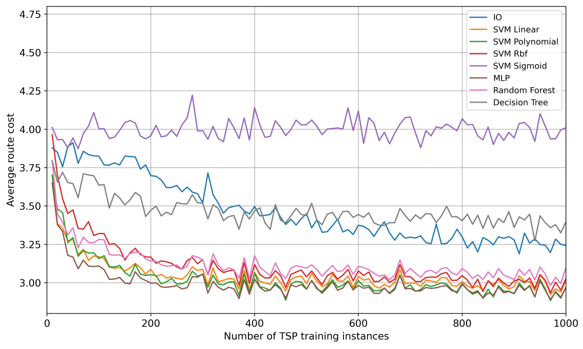

Figure 6.5: TSP (random locations): Cost development during training using global policy training and feature vector reordering. Distances included in the state representation. Evaluated over 200 TSP test instances after each batch of 10 TSP training instances.

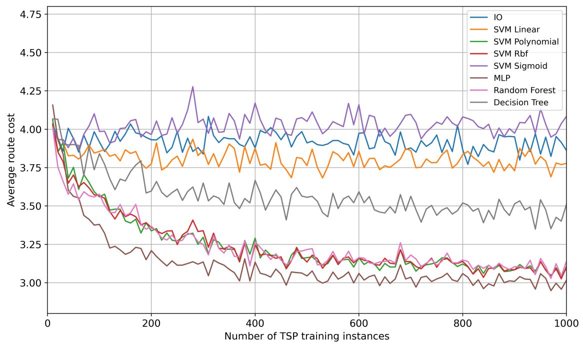

Figure 6.6: TSP (random locations): Cost development during training using global policy training and feature vector reordering. Distances not included in the state representation. Evaluated over 200 TSP test instances after each batch of 10 TSP training instances.

In the experiment where the distances were included in the feature vector, Figure 6.5, we see that after 500 training instances only for the IO model, there was still a relatively large decrease in the average routing cost. However, this seems to almost stop after 800 training instances. For the other models, only a marginal decline can be observed after the first 500 training instances.

Now comparing these developments to the experiment where no distances were included in the feature vector, we see clear differences. Some models, including the SVM polynomial, SVM Rbf, MLP, Random Forest and Decision tree, still follow a similar development curve. However, this decline is slightly slower, and also, the average route costs at the end of training are somewhat higher than when the distances were included in the feature vector. In addition to these models, we can observe that the average route costs for the remaining models decrease just a little or not at all during training. In the case of the SVM Sigmoid, this means not much has changed. However, for the IO and SVM Linear models, this is a major difference as they are now unable to improve their performance during training.

Larger Problem Size

Finally, besides the experiment performed above, we also evaluate the performance of our models on different problem sizes to see how this influences the results. To obtain insight into this, we trained the models using a global policy approach with feature vector reordering on 500 TSP problem instances consisting of 20 nodes with random customer locations. The models were trained in an offline batch manner, with each batch including ten problem instances.

Figure 6.7 hows the average routing cost for each model during training. After each training batch, the performance of the models was evaluated over 200 different TSP test instances. Furthermore, Table 6.4 and Figure 6.8 provide the performance after training along with the standard deviation of each model compared to the average optimal and greedy routing costs.

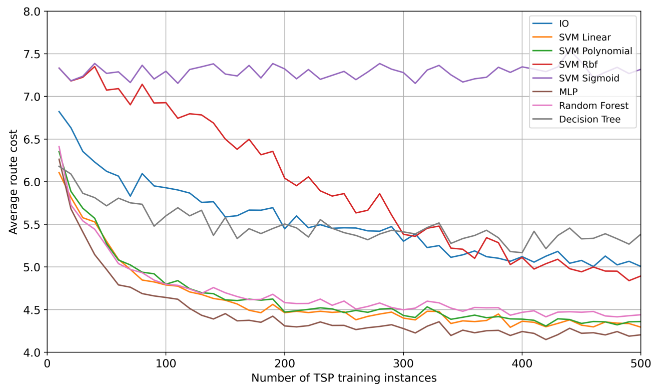

Figure 6.7: TSP (random locations, 20 nodes): Cost development during training using global policy training and feature vector reordering. Evaluated over 200 TSP test instances after each batch of 10 TSP training instances.

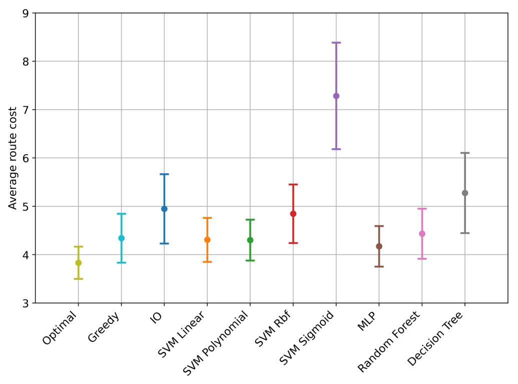

Figure 6.8: TSP (random locations, 20 nodes): Average routing costs with standard deviation per model after training over 500 TSP instances using global policy training and feature vector reordering. Evaluated over 200 TSP test instances.

Table 6.4: TSP (random locations, 20 nodes): Average routing cost per model for global policy training and feature vector reordering. Evaluated over 200 TSP test instances.

Average Routing Cost									
Optimal	**Greedy**	**IO**	**SVM Linear**	**SVM Polynomial**	**SVM Rbf**	**SVM Sigmoid**	**MLP**	**Random Forest**	**Decision Tree**
3.83	4.34	4.95	4.30	4.30	4.85	7.29	4.17	4.44	5.28

When analyzing the results and comparing them to previous experiments, we find that the development of the average routing cost with a problem size of 20 nodes (Figure 6.7) for most models is relatively the same as when we considered a problem size of 10 nodes (Figure 6.1). One notable difference is the development in average cost for the SVM Rbf model, which develops slower and has a higher average routing cost at the end of training compared to the majority of models with a problem size of 20 nodes. However, there is still a relatively high rate of decline at the end of training, suggesting that with a larger training data set, the performance can possibly reach the same level as most other models. Furthermore, after the models are trained over 500 TSP instances, we see that the SVM Linear, SVM Polynomial, and MLP models can perform better than the greedy routing strategy, with the best performance obtained by the MLP model. Overall this experiment indicates that with a large enough training data set, most models can deal with different sizes of problems, obtaining relatively the same performances.

6.2. Capacitated Vehicle Routing Problem

Chapter 3 describes the formulation of the CVRP we use to evaluate and assess the different model performances. For the experiments we consider a CVRP consisting of 11 nodes V, among which ten customers and the depot. The x-coordinates and y-coordinates of the customer locations are in the range $[0, 1]$, and in every problem instance, the depot location was fixed. Additionally, the number of vehicles $K = 3$ and each has a maximum capacity

$Q = 20$. The demands of each customer $q_i \ \forall i \in V \backslash \{0\}$ change in every CVRP instance. For each problem instance, these values are sampled without replacement from a set of possible demands $d = [3, 4, 5, 5, 5, 6, 6, 6, 7, 8]$. The distances between the nodes correspond to the euclidean distances.

As described in Chapter 3, when considering fixed locations, the different models can either be trained by using a global policy or creating multiple policies, one for each node in the system, which we refer to as the nodes policy approach. Like with the TSP, we trained the models using an offline batch approach, with each batch including ten different problem instances. Note that each problem instance consists of 12 state-action pairings. A total of 500 problem instances were included in the training data set.

6.2.1. Results

For the results, we first discuss the CVRP formulation with fixed customer locations followed by the CVRP formulation with random customer locations.

Fixed Customer Locations

Figure 6.9 and Figure 6.10 show the development of the average routing cost during training for each model using a global and nodes policy training approach, respectively. After each training batch, the performance was evaluated by solving 200 different CVRP instances and calculating the average total routing cost.

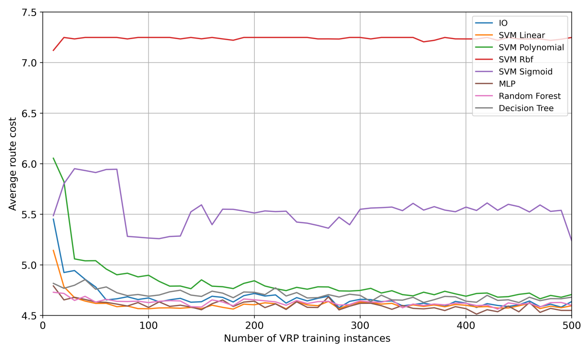

Figure 6.9: CVRP (fixed locations): Cost development during training using global policy training. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

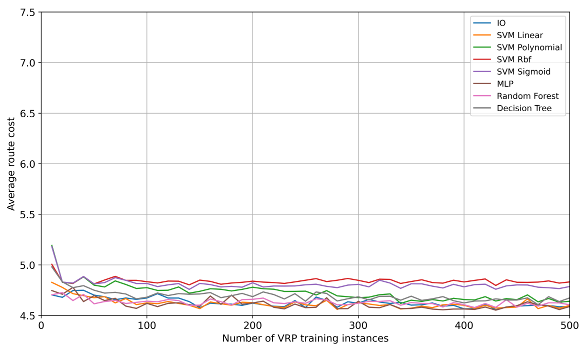

Figure 6.10: CVRP (fixed locations): Cost development during training using nodes policy training. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

In these figures, we can observe that almost all of the different models follow the same development of the average routing cost in both approaches. There is a decrease in the average cost for the initial batches, after which this decline slows down and stabilizes around a certain value, which is slightly lower for most models using the global policy training approach. However, we can see that the SVM Rbf and SVM Sigmoid models are two exceptions. For both models, the average routing cost converges to a much higher value during the global policy training approach. In addition, we see that for the SVM Polynomial model the average routing cost reduces more slowly using the global policy approach and the average value finally reached is a somewhat higher value compared to the majority of other models at the end of training.

The CVRP differs from the TSP in that multiple cars are used for routing, and each vehicle has a maximum capacity, as previously noted in the problem description. The consequence of this is that, depending on early routing decisions, it can be possible that the problem becomes unsolvable since none of the vehicles has enough capacity left to serve the remaining customer(s). As a result, the route and, therefore, the solution are incomplete. We increased the overall routing cost by double the distance between the depot and the remaining customer(s), which equates to an additional trip, as a penalty for this situation. Figure 6.11 and Figure 6.12 show how each model's incomplete routes, represented as a percentage of the total number of problem instances evaluated after each batch, develop during training for the global and nodes policy approach, respectively.

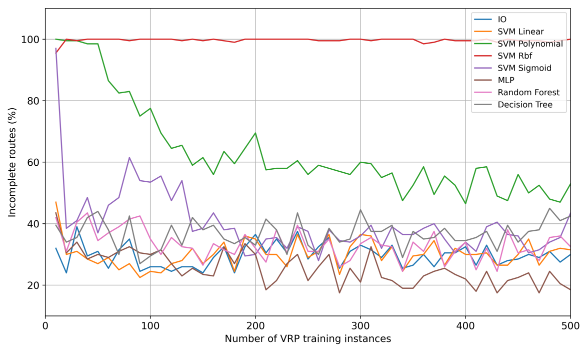

Figure 6.11: CVRP (fixed locations): Incomplete routes development during training using global policy training. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

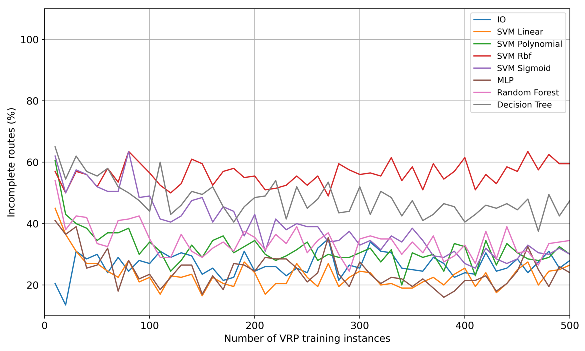

Figure 6.12: CVRP (fixed locations): Incomplete routes development during training using nodes policy training. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

Here we can see that, for most models, the development of the percentage of incomplete routes and the values are somewhat the same for both policy training approaches. Something that immediately stands out is that almost 100% of routes are incomplete for the SVM Rbf model using the global training approach throughout training. Upon further investigation of the routing decisions made by this model, we found that it almost always chooses to return to the depot after visiting a single customer for each vehicle. This is also reflected in the extremely high, almost constant average routing cost during the global policy approach training, as seen in Figure 6.9. Although this is not the case when using the nodes policy training approach, the SVM Rbf still performs than the other models, producing an incomplete route in more than half of the problem instances. Additionally, this model's performance does not improve as it continues to be trained.

Furthermore, we also see that the percentage of incomplete routes for the SVM Polynomial model is much higher during the global policy training approach. Note that the development of the percentage of incomplete routes for the model is similar to that of the average routing cost in Figure 6.9. It is also interesting to note that for the global policy training, the SVM Sigmoid model, which has a high average routing cost, does improve on its percentage of incomplete routes until it reaches the same level as the majority of other models at the end of training.

Finally, considering the nodes policy training approach, we see that the Decision Tree model has more incomplete routes than most other models. However, the average routing cost of the model during training is only slightly higher than that of other models, as seen in Figure 6.10.

After training the models, we also evaluated their performance on 200 new CVRP instances as we did for the TSP experiments. Comparing the results to the optimal solutions and the greedy approach. These results can be found in Table 6.5 as well as Figure 6.13 for the global policy training and Figure 6.14 for the nodes policy training approach.

Table 6.5: CVRP (fixed locations): Average routing cost per model for different policy training approaches. Trained over 500 TSP training instances and evaluated over 200 TSP test instances.

	Average Routing Cost						**Optimal** **Greedy**	4.39 5.61
Training Approach	**IO**	**SVM Linear**	**SVM Polynomial**	**SVM Rbf**	**SVM Sigmoid**	**MLP**	**Random Forest**	**Decision Tree**
Global	**4.56**	**4.54**	4.67	7.25	5.22	**4.52**	**4.54**	**4.58**
Nodes	4.61	4.55	**4.62**	**4.81**	**4.75**	4.56	4.57	4.61

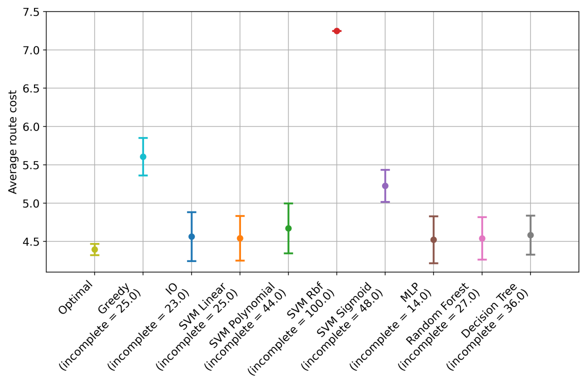

Figure 6.13: CVRP (fixed locations): Average routing costs with standard deviation per model after training over 500 CVRP instances using global policy training. Evaluated over 200 TSP test instances.

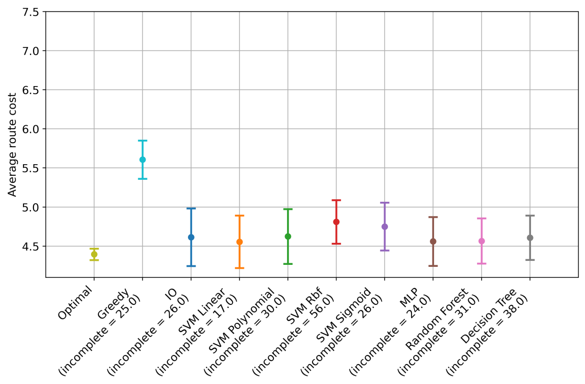

Figure 6.14: CVRP (fixed locations): Average routing costs with standard deviation per model after training over 500 CVRP instances using nodes policy training. Evaluated over 200 TSP test instances.

The results for both policy training approaches correspond to the models' performances at the end of training, as we would expect. For most models, the best performance is obtained using the global policy training approach, except for the SVM Polynomial, SVM Rbf and SVM Sigmoid models, which performed better using the nodes policy approach. The best overall performance is obtained by the MLP model having both the lowest average routing cost and percentage of incomplete routes for the global policy training approach.

When comparing the results with the optimal and greedy routing results, we can see that, with the exception of the SVM Rbf using the global policy, all models are able to obtain an average routing cost relatively close to that of the optimal solution and below the greedy routing strategy. However, focusing on the percentage of incomplete routing solutions, we find that for the global policy, only the IO and MLP models are able to obtain a lower result than the greedy strategy. For the nodes policy training approach, this is only the case for the SVM Linear and MLP models.

Random Customer Locations

For training the models on CVRP with random customer locations, we only use the global policy training approach, which is combined with both no feature vector transformation as well as feature vector reordering. Figure 6.15 and Figure 6.16 show the development of the average routing cost for each model using no transformation and reordering, respectively.

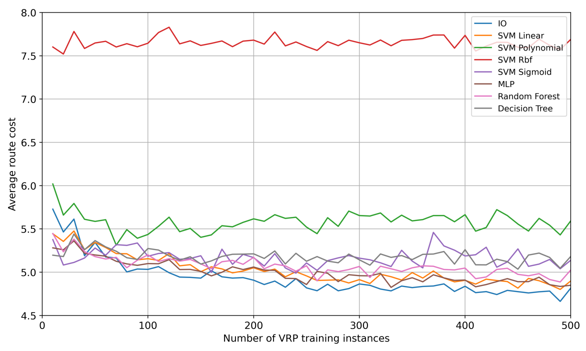

Figure 6.15: CVRP (random locations): Cost development during training using global policy training with no feature vector transformation. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

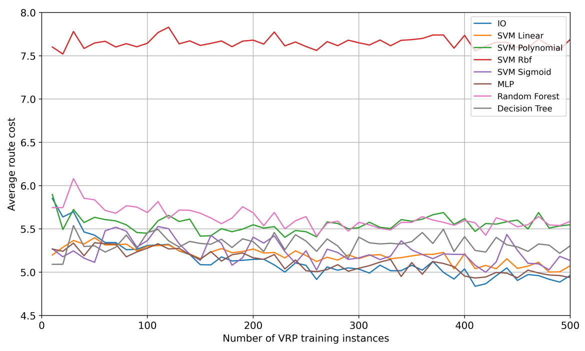

Figure 6.16: CVRP (random locations): Cost development during training using global policy training with feature vector reordering. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

In the figures above, we see that the average route cost development within each feature vector transformation approach is almost the same for most models. Furthermore, if we compare the two approaches, we see almost the same development for each model, with the value for the average route cost being slightly lower in the case of no feature vector transformation compared to the reordering approach.

In addition, we see that for both feature vector approaches the SVM Rbf gives a high average route cost compared to the other models that do not improve during training. The reason for this is the same as in the experiments with fixed customer locations. This is also reflected in the 100% incomplete routes during the training, as shown in Figure 6.17 for no transformations and Figure 6.18 for the reordering feature vector transformations.

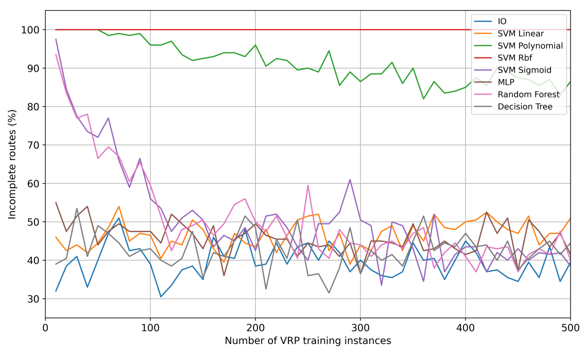

Figure 6.17: CVRP (random locations): Incomplete route development during training using global policy training with no feature vector transformation. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

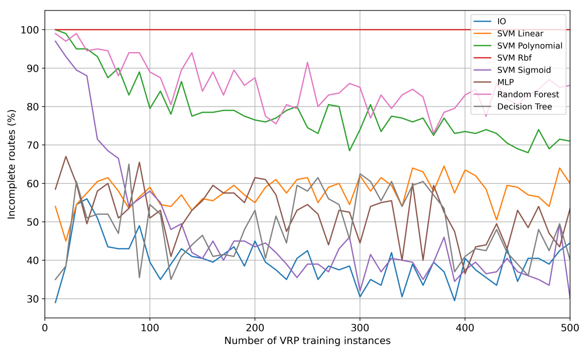

Figure 6.18: CVRP (random locations): Incomplete route development during training using global policy training with feature vector reordering. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

Considering the development of the percentage of incomplete routes for both the training approach with and without feature vector reordering, we see that for the IO, SVM Rbf and SVM Sigmoid models, this is relatively the same in both approaches. However, for the other models, we can observe differences between the two. First, the SVM Linear follows the same development, but the percentage of incomplete routes is higher when using feature vector reordering. For the SVM Polynomial, this is similar but with a lower percentage of incomplete routes using the reordering approach. Furthermore, using no feature vector transformations, we see that the Random Forest model starts with a high percentage of incomplete routes but is able to bring this down to the same level as most other models in the first 200 training instances. When we compare this to the experiment with feature vector reordering, we see that this is not the case. Here the percentage of incomplete routes also starts at a high value but decreases relatively slow during training and ends far above that of the other models.

Finally, we can observe that the percentage of incomplete routes is around the same value in the case of the MLP and Decision Tree models, both with and without feature vector reordering. However, we see that in the case of the reordering approach, this value shows large fluctuations between batches compared to the approach with no transformation.

After training, the performance of the models was tested on 200 new CVRP instances as we did before. Comparing the results to the optimal solutions and the greedy approach. The results of this can be found in Table 6.6 as well as Figure 6.19 and Figure 6.20, for the training approach without and with feature vector reordering, respectively.

Table 6.6: CVRP (random locations): Average routing cost per model using global policy training with different feature vector transformations. Trained over 500 TSP training instances and evaluated over 200 TSP test instances.

	Average Routing Cost						**Optimal**	3.90
							Greedy	4.76
State Feature Vector Transformation	**IO**	**SVM Linear**	**SVM Polynomial**	**SVM Rbf**	**SVM Sigmoid**	**MLP**	**Random Forest**	**Decision Tree**
None	**4.74**	**4.78**	**5.39**	**7.49**	4.96	**4.69**	**4.92**	**5.02**
Reorder	4.86	4.97	5.46	**7.49**	**4.95**	4.87	5.42	5.11

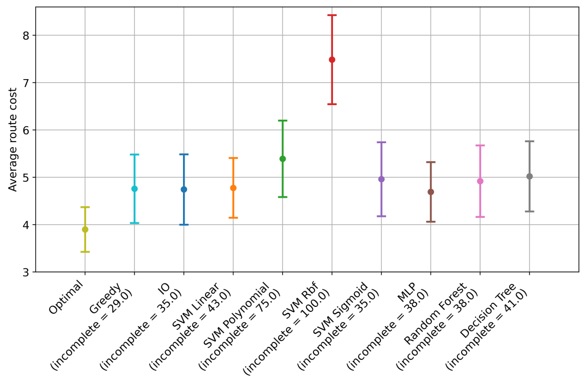

Figure 6.19: CVRP (random locations): Average routing costs with standard deviation per model after training over 500 CVRP instances using global policy training with no feature vector transformation. Evaluated over 200 TSP test instances.

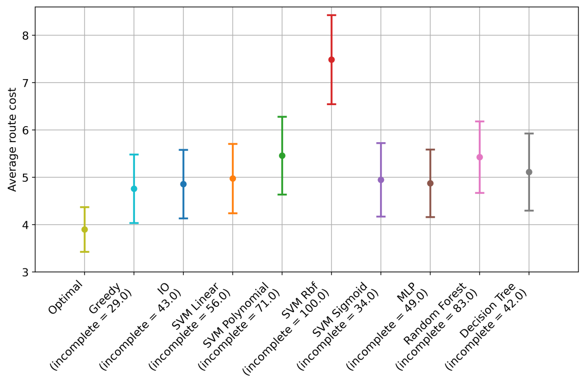

Figure 6.20: CVRP (random locations): Average routing costs with standard deviation per model after training over 500 CVRP instances using global policy training with feature vector reordering. Evaluated over 200 TSP test instances.

Analyzing these results, we find that for most models, better performance in average routing cost is obtained without using feature vector transformation. Only the SVM Sigmoid has a slightly lower average routing cost when using feature vector reordering. Furthermore, in terms of the percentage of incomplete routes, we see that most models perform better using no feature vector transformation. The MLP and IO models obtain the best performances using no feature vector reordering, with a lower average routing cost than the greedy strategy for both models. However, they still have a higher percentage of incomplete routes.

Influence of Distance Features

Just like we did with the TSP problems, we also evaluated the influence of including the distances as features, which are equal to the costs, on the performance of the models. To do this, we performed an additional experiment using global policy training without feature vector transformation, in which the distances between nodes are not included in the state representations. Figure 6.21 and Figure 6.22 show the development of the average routing cost and incomplete routes for each model without the distances included in the state feature vector, respectively.

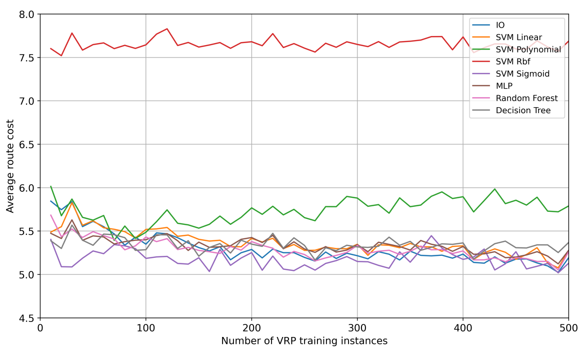

Figure 6.21: CVRP (random locations): Cost development during training using global policy training with no feature vector transformation. Distances not included in the state representation. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

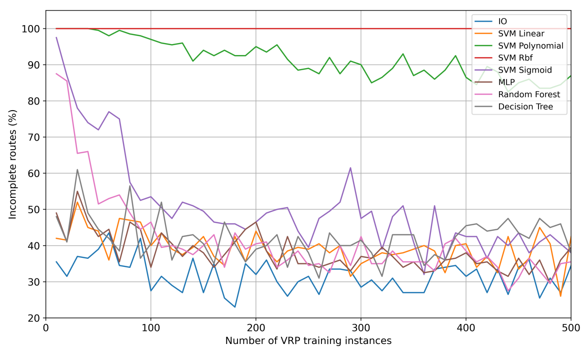

Figure 6.22: CVRP (random locations): Incomplete route development during training using global policy training with no feature vector transformation. Distances not included in the state representation. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

In these figures, we can see that both the development of the average costs and the percentage of incomplete routes, for all models, when the distances are not included in the feature vector, are very similar to the result we observed in Figures 6.15 and 6.17, where the distances were included. With only minor differences between the two experiments.

The results of the models' performances with and without the distances included in the feature vector in terms of average routing costs after training are provided in Table 6.7. Furthermore, Figure 6.23 shows the performance of the models without distances included in the feature vector compared to the optimal solution and greedy routing strategy.

Table 6.7: CVRP (random locations): Average routing cost per model using global policy training with no feature vector transformation. Trained over 500 TSP training instances and evaluated over 200 TSP test instances.

	Average Routing Cost						**Optimal**	3.90
							Greedy	4.76
Distances Included in Feature Vector	**IO**	**SVM Linear**	**SVM Polynomial**	**SVM Rbf**	**SVM Sigmoid**	**MLP**	**Random Forest**	**Decision Tree**
Yes	4.74	4.78	5.39	7.49	4.96	4.69	4.92	5.02
No	5.02	5.08	5.66	7.49	4.95	5.13	5.06	5.21

Analyzing these results, we can see that for almost all models, the average routing cost is higher when the distances are not provided in the feature vector. However, unlike what we saw in the TSP problem, no model has a major performance loss compared to the others. Interestingly, the SVM Sigmoid model improved its performance in terms of average routing cost, making it the best-performing model. Furthermore, we see that for most of the models, the percentage of incomplete routes has decreased when the distances are not included

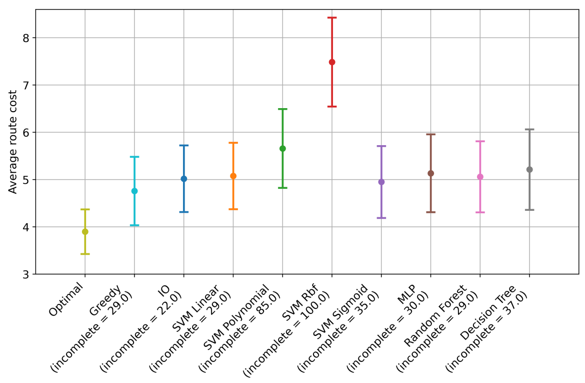

Figure 6.23: CVRP (random locations): Average routing costs with standard deviation per model after training over 500 CVRP instances using global policy training with no feature vector transformation. Distances not included in the state representation. Evaluated over 200 TSP test instances.

compared to the results in Figure 6.19, where distances were provided.

6.3. Discussion

The results above provide interesting insights regarding the different models and approaches for learning to make routing decisions from expert examples. When comparing the TSP and CVRP, we observe some similarities as well as differences in the results. For both problems, most models follow a similar learning curve when considering the development of the average routing cost. The models rapidly decrease their average routing costs in the first initial batches. This decrease then slows down and declines only very slowly for the remainder of the training instances. Furthermore, for both problems, we see that some models are unable to improve their performance during training or stop improving on a relatively high cost value. We mainly observed this with the SVM Sigmoid model for both problems and in the case of the CVRP with the SVM Rbf model when using a global policy training approach.

For both problems, we tested the model's ability to learn from expert examples when dealing with fixed as well as random customer locations. The objective of the models was to learn a policy resembling that of the expert, which was the optimal solver in this case. To do this, the models had to learn the costs of routing decisions (equal to the distances) depending on features and actions. Although the performance in terms of the average routing cost for some models is better than that of others, overall, we can observe that for both the fixed and random locations, most models were able to learn this decision-making up to a certain performance level. However, we also observed that there was a difference in this performance level between the problems with fixed and random locations in both problems.

In the case of the TSP with fixed locations, we found that all models were able to imitate the expert perfectly. This is not surprising as all problem instances and thus solutions are the same. However, for the CVRP with fixed locations, where there are differences between the problem instances, the average routing costs of almost all models get close to the optimal solution, which was used as expert examples. Additionally, all models perform considerably better than a greedy routing approach. For both problems with random customer locations, this is different. Here we observe that the gap between the average routing cost for the optimal solution and that of the different models is larger, with most models achieving average routing costs somewhat higher than the greedy strategy and only a few models obtaining a better result.

The difference in performance level can be explained by the large difference in the edges for the problems with fixed and random customer locations. Consider the CVRP, when using fixed locations, the customers are always in the same place for every problem instance. Therefore, the edges, which connect the customers, and their costs are constant for every problem instance. In our experiments, we used a single depot and ten customers. This means that there are only 110 different edges for which the model has to learn the cost. If we look at the problem with random customer locations, this is very different. Here the edges and thus also their costs vary in every problem instance. Making it considerably more complicated to learn the costs as there is an extremely high number of different possibilities. In this case, the model needs to interpret the new costs of a problem instance in some way using the features provided, which might be more challenging for some of the models, as we observed when comparing the performance using a feature vector with and without distances (and thus costs) included.

In these experiments, we found that, in the case of the TSP, the performances of the IO and SVM Linear models were highly dependent on whether the distances were included in the features. Both of these models use a linear relation between the features and weight in their hypothesis functions which, in the case of the random customer locations, when in every instance the edges are different, and the distances are not included in the features, is not a sophisticated enough for a hypothesis. At least not with the provided features. For the CVRP, we observed that although the average routing costs for most models increased by a small amount when the distances were not included in the feature vector, this was almost the same relative to each other, and there were no models for which the performance majorly decreased compared to the rest. In addition, we saw that the percentage of incomplete routes decreased for most models without the distance features. This indicates the importance of choosing the right features and that while in some cases, more features and thus information can benefit the performance in some aspects, at the same time, it can negatively impact others.

We notice a few things if we take a closer look at the different multiclass classification algorithms. In the case of the SVM, we used different kernel formulations to test their performance differences. The sigmoid kernel is not suitable for the problems we try to learn from and solve. For TSP, this function is barely able to improve its performance during training and also, in the CVRP, it is among the worst performing models of all. On the other hand, the polynomial and rbf kernel can both learn and perform well for the TSP with random locations. Even without the distances included in the state feature vector, both models are still able to reduce the average routing cost compared to other models. Taking into account that this is not the case for the linear formulated models. It shows that a higher-order non-linear hypothesis is needed for this formulation of the problem and state-actions. However, in the case of the CVRP, using the polynomial or rbf kernel results in some of the worst performances of all models. Interestingly, a linear approach is the best performing option for this problem.

The MLP classification model is overall the best performing model we tested, having one of the lowest average routing costs in both problems, even without the distances in the state feature vector for the problems with random locations. This confirms the advantage of the MLP model we mentioned in Chapter 5 and demonstrates the ability of a deep learning approach using neural networks to learn and adapt to different problems. However, the downside is that this model's training time is higher than the other classifiers.

Comparing the performance of the Random Forest and Decision Tree models, we can see that, as expected, the Random Forest model can obtain better results for almost all problems. This can be explained by the fact that the Decision Tree only uses a single tree to make its decisions while the Random forest model combines multiple. This results in lower average routing costs and fewer incomplete routes when considering the CVRP (without feature vector reordering).

Finally, as mentioned, the purpose of the models was to imitate the expert as well as possible based on examples. If we look at the performance of the different models compared to the optimal solution, in this case, the expert, we see that, in both problem formulations, most models can achieve this to some extent. However, note that there is a difference in how the optimal solver and the models determine their solutions. For an optimal solution, we have to solve the whole problem at once to determine the route, and if this problem changes, we have to do this again. For our trained models, this is not the case. These models are trained to make a decision based on the system's current state. Therefore, if the problem changes, this is just a different state. This would suggest that using this state-action formulation, the model's decision-making is faster than the approach of the optimal solver. Although this seems logical, we have not performed any specific testing to confirm this, so this would require further investigation.

7 Conclusion

In our research, we formulated state-action representations for the TSP and CVRP. We discussed how these routing problems could be interpreted as IO or multiclass classification problems evaluating the performance of different algorithms in imitating expert decision-making learned from examples. The characteristics of the problems mainly differed in terms of single vehicle routing and multi-vehicle routing with demands. Besides this, for both routing problems, we experimented with fixed and random customer locations between different problem instances. For the algorithms to learn to solve these routing problems, we propose multiple policy training approaches as well as state feature vector transformations that can be used based on the characteristics of the problems. These different training configurations are tested by an IO model and four multiclass classification models, some with multiple variations on training data sets consisting of state-action pairs for multiple different problem instances. The performance of the trained models is evaluated on new problem instances, for which we compare their performance relative to each other and to the optimal solution which acted as the expert and was used to create the training data. Because we used the distance between the nodes as a routing cost, we can also compare the performances to a greedy solution approach. Furthermore, we demonstrated how different features can influence the performance of the models and, in the case of the TSP formulation, showed the effect of using a larger training or problem size.

Considering our research question, we stated in the introduction of this report. We demonstrated that we can use IO and multiclass classification to learn from example data and imitate expert decision-making. However, we also showed a large variation in performance depending on the problem, state features, algorithm formulations and training configuration. Furthermore, the performance development of the models stabilizes at a certain level above that of the expert (in this case, the optimal), limiting perfect imitation.

For both the TSP and CVRP, our results show that the global policy training approach obtains the best results for most models. In the case of the TSP with random locations, this is true when combining this training approach with the feature vector reordering transformation. Furthermore, we found that overall the MLP model obtains the best performance for most problem setups. As we discussed, this model utilizes a neural network formulation, which is often used in other research on the application of machine learning for routing problems and has many advantages. However, these models are often complex and can require more data or time to train. Therefore, it is interesting to note that the MLP model's performance was closely matched in most experiments and, in some cases, even surpassed by some of the other models that use a simpler formulation, demonstrating their potential. However, the results also show that these simpler models are more susceptible to changes in problem formulation, features and training configurations, making them less versatile and flexible in use compared to the MLP model.

Focusing on the difference in performance considering problems with fixed or random customer locations, we can conclude that the best results are obtained when customer locations are fixed between problem instances. Because the locations are fixed, the edges and thus the costs are constant over all problem instances. As we discussed, this makes it easier for the models to learn the costs of certain routing decisions. Therefore, the models are better at imitating the expert by only learning from the examples. When the locations are random, the edges and thus also the costs of decisions vary for every instance. This creates a more complex problem, making it more challenging for the models to learn the expert's decision-making. In this research, we used the costs equal to the distances, so when we include the distances between locations in the feature vector, we also include the costs in the state representation. However, this can be challenging when more complex costs are used. Additional experiments not including the distances as features show that depending on the problem, it can become challenging for some models to imitate expert decision-making. Therefore, this shows that the hypothesis used by the model needs to be sophisticated enough to learn how the costs relate to the features presented when the costs vary in every problem instance and are not included in the features.

As discussed in Chapter 2, only a few research studies have been performed on learning to solve routing problems from expert examples. Therefore, our research is one of the early steps in investigating the application of known algorithms for different routing problem formulations. Based on our findings, we have some recommendations for further research.

In Chapter 3, we explained how using fixed, and random customer locations between problem instances outline two scenarios in which we can learn and apply the decision-making preferences of experts. The random locations represent a universal scenario for learning to make routing decisions based on general preferences, and the fixed locations represent a more area-specific application. The results of our research suggest that this second scenario might be more suitable for learning from expert examples. So for an area-specific application where the emphasis is on learning from experts and developing a policy for routing decisions in a specified region, one can consider designing a fixed graph that should represent an area layout, with only a few nodes representing consumers. The customer positions can then change inside the graph, but the overall graph, and so the edges and costs, remain unchanged. In this circumstance, the research could also consider more complex, area-specific routing costs that represent the real-world situation. Furthermore, the research might concentrate on other objectives of the routing problem or newly emerging customers during routing.

For the goal of learning from examples universally, we suggest concentrating research on state representation and model design to improve the model performance or make the performance less dependent on specific features. Focusing on a neural network approach could be a promising direction for this. Additionally, we already mentioned that we expect our state-action formulation and problem-solving approach, used by the different models, to be faster than the approach of an optimal solver. However, we performed no further experiments to show this specifically, so this could be confirmed in a new study.

A

Scientific Paper

Learning to Make Routing Decisions Based on Expert Examples

J. Schouten, P. Zattoni Scroccaro, B. Atasoy
Delft University of Technology
Delft, The Netherlands

***Abstract*—Logistics and mobility services play a major role in our society, and efficient routing is a crucial part of this. However, even though routing problems have been widely researched, the solutions provided by algorithms do not always match drivers' expectations. Routing costs used by these algorithms are often based on one or a few parameters, but in real-world operations, many factors and sometimes hard-to-define aspects are responsible for this. Drivers can consider these different aspects and some studies found that experienced drivers often plan better delivery routes than the optimization tools. In this research, we focus on using expert decision data as examples for learning the costs of routing and train a policy that can make decisions more in line with the expectations of the expert. We formulate state-action representations for the TSP and CVRP, which we use to interpret these routing problems as inverse optimization and multiclass classification problems. Additionally, we propose multiple policy training approaches as well as state feature vector transformations that can be used based on the characteristics of the routing problems. These different training configurations are utilized to train different existing algorithms with training data sets consisting of example state-action pairs. The performance of the trained models is compared to each other and the optimal solution. The optimal solution acted as expert example and was used to create the training data. We demonstrate that both inverse optimization and multiclass classification algorithms are able to imitate expert decision-making for new problem instances from example data. However, we also show a large variation in performance depending on the problem, state features, algorithm formulations and training configuration.**

***Index Terms*—Routing, Inverse optimization, Multiclass classification, Supervised learning**

I. Introduction

Logistics and mobility services play a major role in our current society. Ordering packages, groceries, or dinner is becoming more popular. Besides that, many people use mobility services, such as taxis. Vehicle routing is required to fulfil customer demand in all these cases. Finding the best route to visit a set of locations is often referred to as a routing problem. The two most common formulations of these routing problems are the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP). The goal is to determine a set of routes visiting all locations while minimizing the routing cost. However, even though the TSP and CVRP have been widely researched, the "theoretical optimal" solutions provided by routing algorithms do not always match the expectations of drivers.

This is because the costs this algorithm focuses on minimizing are often based on one or a few parameters, like distance or travel time. However, a good solution in real-world operations, according to drivers, can be influenced by many factors, some of which might even be difficult to define explicitly or may derive from the interdependencies among street segments. This complicates estimating real-world costs and incorporating them into an algorithm. Drivers, however, consider these different aspects, sometimes subconsciously, when determining a route. As a result, some studies found that experienced drivers often make better routing decisions than the optimization tools [1], [2].

In this research, we use these driver decisions as examples and utilize machine learning (ML) to learn the costs of specific routing actions and train a policy that can mimic the driver, which we refer to as the "expert". The benefits of this would be that this policy can make better decisions, which are more in line with the drivers' expectations, without having to define specific routing costs for the algorithm. Furthermore, it offers a way in which new drivers can use the knowledge of more experienced drivers.

In recent years, ML is increasingly being used for solving routing problems, with (deep) reinforcement learning (RL) being one of the most researched approaches [3], [4], [5]. RL, however, requires predefined costs, which, in our approach, are not available for the model. The idea of learning from (expert) example mainly falls under the domain of supervised or imitation learning, for which many different techniques exist [6], [7], [8].

We use two approaches for learning a policy to make routing decisions using expert examples. The first approach we use is inverse optimization (IO), which can be used to determine unspecified parameters of an optimization problem that make a given observed solution optimal [9], possibly in the presence of some state-action constraints. The approach is particularly suitable when the action is an optimal decision with respect to a certain unknown cost function [10].

In the second approach, we interpret routing as a multiclass classification problem. In multiclass classification, a model is trained to learn which label belongs to a particular sample so that when presented with a new sample, the algorithm can predict the correct label. In other words, the objective is to create a model of the distribution of class labels in terms of predictive features.

Our goal is to evaluate the use of IO and multiclass classification approaches to learn the costs of routing decisions and how to train a policy that can solve new routing problems imitating expert decision-making. We propose state-action representations for two separate routing problems with different characteristics and show how we can use these state-action formulations for our IO and classification approaches. Depending on the formulation of the problems, we provide multiple training strategies. Finally, we analyze and compare the performance of several existing solution algorithms for the problems in learning from routing examples and subsequently solving new routing problem instances.

The remainder of this report is organized as follows. Section II discusses relevant background literature. Next, we formulate different routing problems, corresponding training approaches in Section III and the IO and multiclass classification models in Section IV. Followed by the numerical experiments in V and conclusion of the findings in Section VI.

II. Background

The interest in IO among the mathematical programming community started with an IO approach for linear programming by investigating the inverse shortest path problem [9]. This was soon followed by a more generalised context based on optimality conditions and dual theory [11], [12]. IO models have further been researched in the context of integer programs [13], [14], network problems [9], [15], [11] and dealing with noisy and imperfect data [16], [17]. The applications of IO models can be found in a wide range of domains, like geoscience [9], [18], healthcare [19], [20] and energy [21], [22].

Considering routing problems, we found one study using an IO approach for solving the CVRP [1]. Another study proposed an urban freight modelling framework modelled as a custom VRP, where an IO approach was used to update coefficients of different objectives [23]. Furthermore, Xu et al. proposed an IO model to learn the parameters of heterogeneous travellers route behaviour to infer shared network state parameters in real-time [24]. Finally, a real-world example of using IO for routing consists of an online route recommendation system for long-haul truck drivers in the Swedish forest industry [25].

Now focusing on multiclass classification problems, various classification variations and approaches have been developed throughout the years [26], [8]. The applications of (multiclass) classification can be found in a wide range of domains, like ecology [27], food chemistry [28], and healthcare [29], [30], [31]. For the domain of transportation, multiclass classification is also used. However, most of these applications focus on categorising data or predicting already established routes [32], [33]. For the applications on vehicle routing, only one recent study was found using a form of classification for vehicle routing [34]. To the best of our knowledge, this is the only study that implements (multiclass) classification for solving routing problems.

III. Modeling Routing Problems

As routing problems we use a TSP and CVRP, which are defined on a graph $G = (V, E)$, with vertex set $V = \{v_0, ..., v_n\}$ and edge set $E = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$, where v_0 customers the depot, and the other vertices in V are the requests that need to be served. The goal of these routing problems is to determine, with a single or multiple vehicles, a set of routes such that each vertex except the depot is visited exactly once while solving an objective function and satisfying constraints.

The main difference between the problems is that in the TSP, only one vehicle is used to visit all the nodes. For the CVRP, multiple vehicles are used to visit the nodes. However, the vehicles only have a limited capacity, and each node has a specific demand that must be fulfilled. These demands remain the same but are redistributed across the nodes for each new problem instance. Both problem formulations will be evaluated with fixed customer locations, where the nodes are in the same place for each new problem instance and with random customer locations. The difference in characteristics regarding the fixed and random locations allows us to test two different scenarios in which we learn and apply the preferences of experts.

The random locations for each new problem instance ensure that the costs of different routing choices also vary according to the problem. Therefore, this scenario considers a universal use for learning to make routing decisions based on more general preferences. The fixed locations are similar to a more area-specific application. Because only a limited network is considered to learn and route inside, the edges, and the routing costs, stay constant.

As we mentioned before, for the algorithms we use to solve these problems, we will not provide the objective function or specify the routing costs used by the expert to determine the routes. Instead, we only provide a data set containing N routing examples, which is used to train the algorithms to make routing decisions. This data set consists of multiple state-action pairs $\{\hat{s}_i, \hat{u}_i\}_{i=1}^{N}$. To create these state-action pairs, we assume that a route, which is the solution to the routing problem, is formulated using a series of actions or decisions u. Each of the decisions being which node the vehicle will visit next. The state s represents the system state of the problem as it was when making the decision and is formulated as a feature vector.

A. Traveling Salesman Problem

In the TSP a single vehicle must visit a set all vertex $V = \{0, \ldots, N\}$ consisting of N nodes. The vehicle starts at the depot and must also return to the depot after visiting all the customers. The goal of the problem is to minimize the total routing cost. $C = \{c_{ij},\, i, j \in V\}$ is the cost matrix, in which c_{ij} is the travel cost from node i to node j, which is equal to the euclidean distance between the nodes.

To formulate a state s used in the data set, we selected the x-coordinates and y-coordinates of every node, the information if a node has already been visited or not and in case of the random customer locations also the distances of the node that the vehicle is located at that point to all the other nodes and the distances between all nodes and the depot as features. We

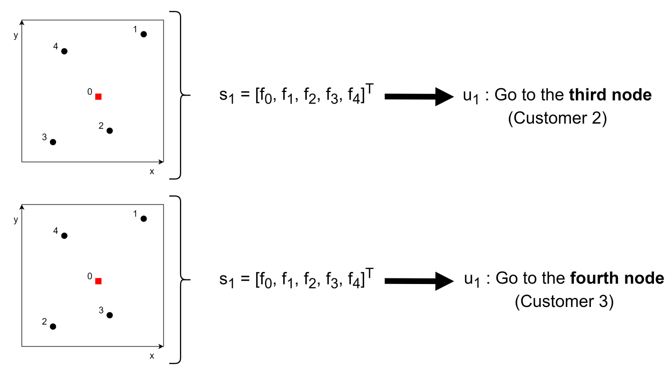
y
x
0
1
2
3
4
s1 = [f0, f1, f2, f3, f4]ᵀ
u1 : Go to the **third node**
(Customer 2)
s1 = [f0, f1, f2, f3, f4]ᵀ
u1 : Go to the **fourth node**
(Customer 3)

Fig. 1. Relation between node labels and decisions. For the same system different labeling outputs a different decision for the same routing location.

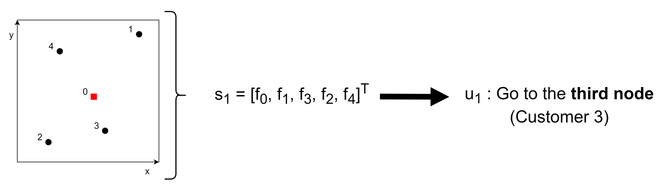

Fig. 2. Reordering of the feature vector results in different decision.

represent the information on if a node is already visited or not as an array of ones and zeros, where zero means a node is visited and one means the node is not visited and thus is still available.

Consider a system of 5 nodes, four customers and the depot. A vehicle started at the depot and followed a partial route visiting customer 2 and 3, respectively. Therefore, currently, the vehicle is located at customer 2. This results in the following features and corresponding feature vector, which are combined in a single state vector:

$$\begin{array}{rl} \text{x-coordinates:} & [x_0,\, x_1,\, x_2,\, x_3,\, x_4]^\top \\ \text{y-coordinates:} & [y_0,\, y_1,\, y_2,\, y_3,\, y_4]^\top \\ \text{distance to current node:} & [d_{20},\, d_{21},\, 0,\, d_{23},\, d_{24}]^\top \\ \text{distance to depot:} & [0,\, d_{01},\, d_{02},\, d_{03},\, d_{04}]^\top \\ \text{available nodes:} & [1,\, 1,\, 0,\, 0,\, 1]^\top \end{array}$$

The customers in a certain system are arbitrarily numbered, starting with the number 1. However, the numbering of the customers has no relation to the routing choices and only serves as a reference for interpreting the decisions made by the algorithm. This is illustrated by Figure 1.

In this example, we consider the same system, represented by a simplified feature vector s only consisting of a single feature f per node, with different labelling of the customers. As a result of the different labelling, the corresponding feature vector is different for the same order of the nodes, resulting in different actions for travelling to the same location. We can see that the labelling will not influence the location the vehicle will travel. This allows us to change the order of the second feature vector so that both systems states and actions are represented the same, as we demonstrate in Figure 2.

We use this information to reorder and even reduce the feature vector depending on the current system state and previous decisions so that the features of nodes that were already visited are moved to the end of their respective feature group or removed entirely. The result of this reordering is that the available nodes and their features will always be in front of the node order and their corresponding feature parts. Therefore, the decision will always be to go to one of the first nodes in the order, so depending on the number of available nodes, it will reduce the variety in the actions. Suppose we now consider the approach of training a different policy depending on the number of available nodes left. In that case, we can argue that we do not need to include the features of nodes that have already been visited before the current node. So we remove them, which reduces the length of the feature vector. Note that reduction of the feature vector can only be applied when training multiple decision-making policies because of the varying vector size.

B. *Capacitated Vehicle Routing Problem*

In the CVRP a number of vehicles K with a limited capacity Q is used to visit a set of vertex $V = \{0, \dots, N\}$ consisting of N nodes. All customers have a demand $q_i, i \in V\backslash\{0\}$ and may only be visited once. The goal of the problem is to minimize the total routing cost. $C = \{c_{ij},\, i, j \in V\}$ is the cost matrix, in which c_{ij} is the travel cost from node i to node j, which is equal to the euclidean distance between the nodes.

Like with the TSP, we formulate a vector using multiple features to represent the system's state. The features we use are; the vehicle for which we have to make the current decisions, its current capacity and location. The vehicle that has travelled the smallest distance is selected to move next. To indicate which vehicle this is, we use a one-hot encoding. This means that we use a vector with an equal length to the number of vehicles, where every element of the vector is equal to zero except for the considered vehicle, which is equal to one. For example, if there are three vehicles and we are considering the second vehicle, the one-hot encoding results in $[0,\, 1,\, 0]$. We use the same approach of one-hot encoding to represent the location of the current vehicle, where we use a vector with the length equal to the number of nodes of which the element corresponding with the current node is equal to one. The other features we use are; the current capacities of all vehicles, the current location of all vehicles again in a one-hot encoding, the initial demand of each customer and which customer the vehicle can visit depending on if the customer has already been visited and the vehicle still has enough capacity to serve the demand. Furthermore, in the case of the random customer locations, we include the x-coordinates and y-coordinates of every node, the distances of the node where the current vehicle is located at that point to all the other nodes and the distances between all nodes and the depot.

C. Data Set Creation

Both a training data set and a test data set are generated for each problem, containing a predetermined number of different problem instances. In the case of the training data set, these problem instances need to be transformed into state-action pairs. To do this, we determine the optimal route by solving each problem instance using Gurobi [35]. The optimal solution will be used as the expert routing examples we use for training. We split each optimal route into individual actions, with each new node in the route being an action. Then we determine the corresponding feature vectors representing the state for each action.

IV. Solution Methods

A. Inverse Optimization

In IO an agent aims to learn the behaviour of an expert who makes decisions based on an exogenous signal, which in this research is represented by the state of the routing problem. To do this, it is assumed that upon receiving a state $s \in \mathbb{S} \subseteq \mathbb{R}^n$, the expert optimizes a parametric optimization problem over a set of feasible actions $\mathbb{U}(s) \subseteq \mathbb{R}^m$, which also depend on the system state s. This optimization problem is formulated as

$$\min_{u \in \mathbb{U}(s)} F(s,u), \tag{1}$$

where $F : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$. In this optimization problem, we assume that for every state $s \in \mathbb{S}$, the set of minimizers $\mathbb{U}^*(s) := \arg\min_{u \in \mathbb{U}(s)} F(s,u)$ is non-empty. Furthermore, we assume that the agent has no prior knowledge of the objective function F representing the expert's preferences. Thus the agent is unable to predict the experts response u^{ex} to a particular system state s prior to training. The agent does however has access to a data set consisting on N states which are paired with corresponding expert actions $\{\hat{s}_i, \hat{u}_i^{\text{ex}}\}_{i=1}^N$, for which $\hat{u}_i^{\text{ex}}$ is a minimizer for $\hat{s}_i$.

We assume that the unknown objective function $F(s,u)$ can be represented by a parameterized function $F_\theta(s,u)$, which is part of a parametric hypothesis space $\mathcal{F} = \{F_\theta : \mathbb{S} \times \mathbb{U} \to \mathbb{R} \mid \theta \in \Theta\}$, where $\theta \in \Theta$ represents the parameters to be learnt. Using this we can formulate the optimization problem that the agent tries to solve as

$$\min_{u \in \mathbb{U}(s)} F_\theta(s,u). \tag{2}$$

We define the parameterized objective function $F_\theta(s,u)$ the agent uses to replicate the the expert behavior as

$$F_\theta(s,u) := s^\top Q u, \tag{3}$$

where we say $\theta = \text{vec}(Q)$, meaning that the parameter vector θ can be reshaped into a parameter matrix $Q \in \mathbb{R}^{n \times m}$.

Loss Function

Ideally, the agent would aim to identify a hypothesis closely resembling the behaviour of the expert. To do this, the available data set is used to training the agent so it can learn the parameters in the hypothesis. A loss function ℓ_θ is used to quantify the inaccuracy of the parameter. The goal during training would be to find the parameter θ which minimize the total loss.

In our research, we use the generalized suboptimality loss [36], and for a given data set, the training phase of the IO approach results in the optimization problem

$$\begin{aligned} \min_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \big(& F_\theta(\hat{s}_i, \hat{u}_i^{\text{ex}}) - \\ & \min_{u_i^{\text{ag}} \in \mathbb{U}(\hat{s}_i)} \left\{ F_\theta(\hat{s}_i, u_i^{\text{ag}}) - I(u^{\text{ex}}, u^{\text{ag}}) \right\}. \end{aligned} \tag{4}$$

with $I(u^{\text{ex}}, u^{\text{ag}}) = 0$ if $u^{\text{ex}} = u^{\text{ag}}$, else $I(u^{\text{ex}}, u^{\text{ag}}) = 1$.

For the implementation, we replicate the work of Zattoni et al. [36] to obtain the parameter vector θ for a certain training data set.

B. Multiclass Classification

To use classification algorithms for solving the TSP and VRP, we need to interpret these routing problems as classifications problem. The system's state is described as a feature vector in the routing problems. We consider this as the sample which needs to be classified. The next node on the route is determined depending on the system's state. Therefore, we can consider this node as the class belonging to the system state. Using this approach, the number of nodes determines the number of classes in the classification problem.

Important to note is that the classification algorithms do not necessarily consider the constraint that applies to the problems. To ensure that no constraints are violated, we validate the suggested action before it is output. If the classifier would suggest going to a node that violates any constraint and is infeasible, the closest feasible node will be output instead.

The following multiclass classification algorithms are used in this study to evaluate and compare their performance in solving the TSP and CVRP: Support Vector Machine (SVM), Multi-layer Perceptron (MLP), Random Forest, and Decision Tree. In the case of the SVM, we used four kernel functions: linear, polynomial, radial basis function (rbf), and sigmoid. The algorithms were implemented using Scikit-learn 1.1.1 [37], and the reader is referred to the literature for detailed formulations.

V. Numerical Experiments

As previously stated, the algorithms were evaluated on two different routing problems, the TSP and CVRP. To solve both problems, eight different algorithms were considered. All the algorithms were trained using the same data set. During testing, we compare the performance of the algorithms to each other and to the optimal solutions we want them to mimic. Furthermore, we also compare the findings to a greedy solution approach to evaluate performance.

Python 3.7 was used to implement all algorithms and conduct all experiments, where the IO model was implemented using Gurobi [35]. In the case of the SVM we used regularization parameter $C = 1$. Furthermore, in the MLP, 100 hidden layers were used, and the Random Forest consisted of 100 decision trees. All other parameters were left at their default settings in Scikit-learn 1.1.1.

A. Traveling Salesman Problem

For evaluating the different models, we considered a TSP consisting of 10 nodes V, a single depot and nine customers. The x-coordinates and y-coordinates of the customer locations are in the range $[0, 1]$. In every problem instance, the depot location was fixed, positioned at the point (0.5,0.5) in the middle of the feasible location region. Furthermore, the distances between nodes are equal to the euclidean distances. The models were trained in an offline batch manner, with each batch including 10 problem instances and the total training data set consisting of 500 problem instances. The performance of the models was evaluated by solving 200 different TSP instances and calculating the average routing cost after each batch and at the end of training.

There are multiple ways to approach the training of a decision-making model for solving the routing problem. The first and most straightforward way is training one decision-making policy for the entire statespace $\mathbb{S}$. Another option would be to train different policies depending on the number of available nodes left in the system, which we refer to as a decisions policy. To do this, we divide the data set of state-action pairs used for training in n_d individual data sets, which we use to learn a policy for every number of available nodes left.

For the TSP with fixed customer locations, we found that all models were able to imitate the expert which is the optimal solution exactly. Note that when we consider the TSP with fixed locations, every individual problem instance in training as well as in testing is the same. Therefore, this result is expected, showing that all models can learn how to solve a single TSP problem instance from an example.

Focusing on the TSP with random customer locations, we find more varying results. Table I provides an overview of the models' performances for each training configuration. The average optimal cost of these test instances, which the models are trying to imitate, is equal to 2.83, and the average routing cost using a greedy approach is equal to 3.08.

When analyzing these results, we find that for each training configuration, the best performance is obtained by the MLP model, with its overall best performance using the global policy training approach with feature vector reordering. This global policy training with feature vector reordering approach obtained almost all models' individual best performance, except for the IO, SVM Linear and SVM Sigmoid models.

Comparing the results of the global and decisions policy training approach, we can see that for most models, except for the IO and SVM Sigmoid models, the decisions policy training approach does not increase performance. Furthermore, considering the feature vector transformations, we observe that for the global as well as the decisions policy training approach, the performance of almost all models increases when reordering, and in the case of the decisions policy reducing, the state feature vector compared to using no transformation.

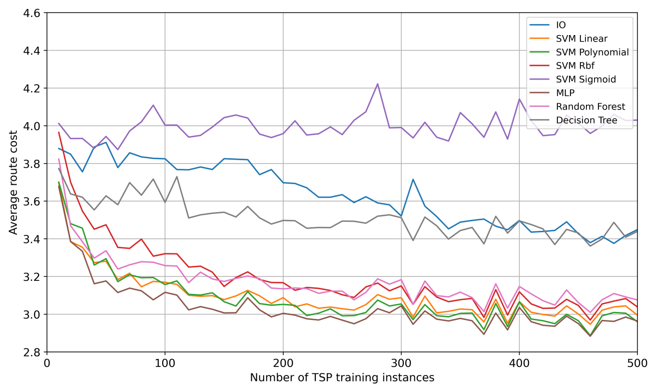

Fig. 3. TSP (random locations): Cost development during training using global policy training and feature vector reordering. Evaluated over 200 TSP test instances after each batch of 10 TSP training instances.

In the current results, we see that the SVM Linear, SVM Polynomial and MLP models all outperform the greedy approach using the global policy training with feature vector reordering, with the SVM Linear and MLP models also obtaining better results in some other training configurations.

Figure 3 shows the development of the average overall routing cost for each model during training using the global policy training with the feature vector reordering approach. We found that the average cost development of all training configurations is relatively similar. Therefore, we only focus on the global policy with the feature vector reordering approach, as this training approach obtained the best performances for most models.

Examining the data in Figure 3, we observe that most models follow a similar development curve. The average routing cost decreases rapidly during the first 200 TSP training instances, after which this decline slows down and almost stabilizes after about 400 TSP training instances. There are three models, however, for which the development of the average cost is different. For both the IO and Decision Tree models, average costs also decrease. However, the curves follow a slower and more steady rate than the other models. Furthermore, for these two models, the decline in the average routing cost does not seem to stabilize at the end of training, which indicates that they may reduce further when more training instances are available. Finally, the average routing cost of the SVM Sigmoid model shows no improvement during training.

B. Capacitated Vehicle Routing Problem

For the experiments, we consider a CVRP consisting of 11 nodes V, ten customers and the depot. We used the same coordinate range for the TSP experiments with the fixed depot location. Additionally, the number of vehicles $K = 3$ and each vehicle has a maximum capacity $Q = 20$. The demands of each customer $q_i \, \forall i \in V \backslash \{0\}$ changes each problem instance.

TABLE I
TSP (RANDOM LOCATIONS): AVERAGE ROUTING COST PER MODEL FOR DIFFERENT POLICY TRAINING CONFIGURATIONS. TRAINED OVER 500 TSP TRAINING INSTANCES AND EVALUATED OVER 200 TSP TEST INSTANCES.

		Average Routing Cost									
Training Approach	**State Feature Vector Transformation**	**Optimal**	**Greedy**	**IO**	**SVM Linear**	**SVM Polynomial**	**SVM Rbf**	**SVM Sigmoid**	**MLP**	**Random Forest**	**Decision Tree**
Global	None	2.83	3.08	3.36	**3.05**	3.10	3.19	4.13	3.00	3.28	3.55
Global	Reorder	2.83	3.08	3.38	3.06	**3.02**	**3.08**	3.90	**2.97**	**3.13**	**3.44**
Decisions	None	2.83	3.08	3.49	3.33	3.20	3.35	**3.75**	3.18	3.39	3.64
Decisions	Reorder	2.83	3.08	3.35	3.12	3.09	3.13	3.77	3.03	3.23	3.57
Decisions	Reduce	2.83	3.08	**3.33**	3.11	3.09	3.11	3.77	3.01	3.21	**3.44**

These values are sampled without replacement from a set of possible demands $d = [3, 4, 5, 5, 5, 6, 6, 6, 7, 8]$. Like with the TSP, we trained the models using an offline batch approach, with each batch including 10 different problem instances. A total of 500 problem instances were included in the training data set.

There are multiple ways we can approach the training of a decision-making model for solving the CVRP. We can again use the global policy training approach. However, the fixed node locations also allow us to take a node-orientated training approach. This means that we train and use different policies depending on the node where the vehicle is located. To do this, we divide the data set of state-action pairs used for training in n_n individual data sets.

Fixed Customer Locations

For the development of the average routing cost during training for each model using a global and nodes policy training, we found that almost all of the different models follow the same development of the average routing cost in both approaches, where there is a decrease in the average cost for the initial batches, after which this decline slows down and stabilizes around a certain value, which is slightly lower for most models using the global policy training approach. However, the SVM Rbf and SVM Sigmoid models are two exceptions. For both these models, the average routing cost converges to a much higher value during the global policy training approach.

The CVRP differs from the TSP in that multiple cars are used for routing, and each vehicle has a maximum capacity, as previously noted in the problem description. The consequence of this is that, depending on early routing decisions, it can be possible that the problem becomes unsolvable since none of the vehicles have enough capacity left to serve the remaining customer(s). As a result, the route and, therefore, the solution is incomplete. We increased the overall routing cost by double the distance between the depot and the remaining customer(s), which equates to an additional trip, as a penalty for this situation. Figure 4 and Figure 5 show how each model's incomplete routes, represented as a percentage of the total number of problem instances evaluated after each batch, develop during training for the global and nodes policy approach, respectively.

Here we can see that, for most models, the development of the percentage of incomplete routes and the values are somewhat the same for both policy training approaches. Something that immediately stands out is that throughout training, almost 100% of routes are incomplete for the SVM Rbf model using the global training approach. Upon further investigation of the routing decisions made by this model, we found that it almost always chooses to return to the depot after visiting a single customer for each vehicle. Although this is not the case when using the nodes policy training approach, the SVM Rbf performs worse than the other models, producing an incomplete route in more than half of the problem instances.

Furthermore, we also see that the percentage of incomplete routes for the SVM Polynomial model is much higher during the global policy training approach. It is also interesting to note that for the global policy training, the SVM Sigmoid model, which has a high average routing cost, does improve on its percentage of incomplete routes until it reaches the same level as the majority of other models at the end of training.

After training the models, we also evaluated their performance. Comparing the results to the optimal solutions and the greedy approach. The results of this can be found in II.

For most models, the best performance is obtained using the global policy training approach, except for the SVM Polynomial, SVM Rbf and SVM Sigmoid models, which performed better using the nodes policy approach. The best overall performance is obtained by the MLP model having both the lowest average routing cost and the percentage of incomplete routes for the global policy training approach.

When comparing the results with the optimal and greedy routing results, we can see that, with the exception of the SVM Rbf using the global policy, all models are able to obtain an average routing cost relatively close to that of the optimal solution and below the greedy routing strategy. However, focusing on the percentage of incomplete routing solutions, we find that for the global policy, only the IO and MLP models are able to obtain a lower result than the greedy strategy. For the nodes policy training approach, this is only the case for the SVM Linear and MLP models.

Random Customer Locations

For training the models on CVRP with random customer locations, we only use the global policy training approach, which is combined with both no feature vector transformation as well as feature vector reordering.

We found that for each feature vector transformation approach, the development of the average route cost is almost

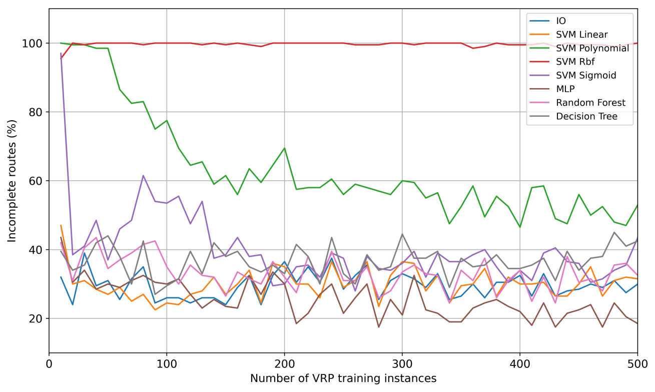

Fig. 4. CVRP (fixed locations): Incomplete routes development during training using global policy training. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

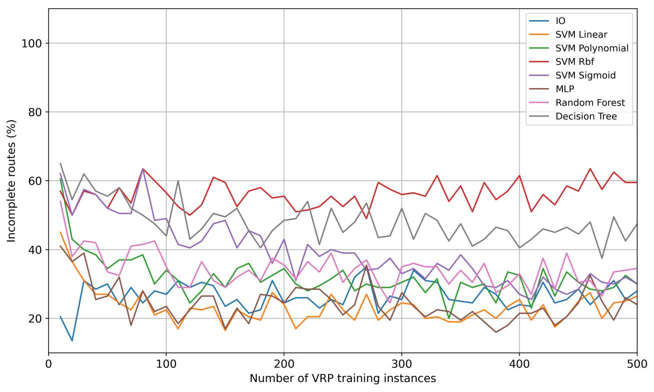

Fig. 5. CVRP (fixed locations): Incomplete routes development during training using nodes policy training. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

TABLE II
CVRP (FIXED LOCATIONS): AVERAGE ROUTING COST AND PERCENTAGE OF INCOMPLETE ROUTES PER MODEL FOR DIFFERENT POLICY TRAINING APPROACHES. TRAINED OVER 500 TSP TRAINING INSTANCES AND EVALUATED OVER 200 TSP TEST INSTANCES.

Training Approach	**Optimal**	**Greedy**	**IO**	**SVM Linear**	**SVM Polynomial**	**SVM Rbf**	**SVM Sigmoid**	**MLP**	**Random Forest**	**Decision Tree**
	Average Routing Cost									
Global	4.39	5.61	**4.56**	**4.54**	4.67	7.25	5.22	**4.52**	**4.54**	**4.58**
Nodes	4.39	5.61	4.61	4.55	**4.62**	**4.81**	**4.75**	4.56	4.57	4.61
	Percentage of Incomplete Routes									
Global	0%	25%	**23%**	25%	44%	100%	48%	**14%**	**27%**	**36%**
Nodes	0%	25%	26%	**17%**	**30%**	**56%**	**26%**	24%	31%	38%

the same for most models. Furthermore, if we compare the two approaches, we found almost the same development for each model, with the value for the average route cost being slightly lower in the case of the no feature vector transformation compared to the reordering approach. Furthermore, the SVM Rbf gives a high average route cost compared to the other models that do not improve during training. The reason for this is the same as in the experiments with fixed customer locations. This is also reflected in the 100% incomplete routes during the training, as shown in Figure 6 for no transformations and Figure 7 for the reordering feature vector transformations.

Considering the development of the percentage of incomplete routes for both the training approach with and without feature vector reordering, we see that for the IO, SVM Rbf and SVM Sigmoid models, this is relatively the same in both approaches. However, for the other models, we can observe differences between the two. First, the SVM Linear follows the same development, but the percentage of incomplete routes is higher when using feature vector reordering. For the SVM Polynomial, this is similar but with a lower percentage of incomplete routes using the reordering approach. Furthermore, using no feature vector transformations, we see that the Random Forest model starts with a high percentage of incomplete routes but is able to bring this down to the same level as most other models in the first 200 training instances. When we compare this to the experiment with feature vector reordering, we see that this is not the case. Here the percentage of incomplete routes also starts at a high value but decreases relatively slow during training and ends far above that of the other models.

Finally, we can observe that the percentage of incomplete routes is around the same value in the case of the MLP and Decision Tree models, both with and without feature vector reordering. However, we see that in the case of the reordering approach, this value shows large fluctuations between batches compared to the approach with no transformation.

After training, the performance of the models was tested on 200 new CVRP instances as we did before. Comparing the results to the optimal solutions and the greedy approach. The results of this can be found in Table III.

Analyzing these results, we find that for most models, better performance in average routing cost is obtained without using feature vector transformation. Only the SVM Sigmoid has a slightly lower average routing cost when using feature vector reordering. Furthermore, in terms of the percentage of incomplete routes, we see that most models perform better using no feature vector transformation. The MLP and IO models obtain the best performances using no feature vector reordering, with a lower average routing cost than the greedy strategy for both models. However, they still have a higher percentage of incomplete routes.

C. Discussion

The results above provide interesting insights regarding the different models and approaches for learning to make routing decisions from expert examples. For both of the problems,

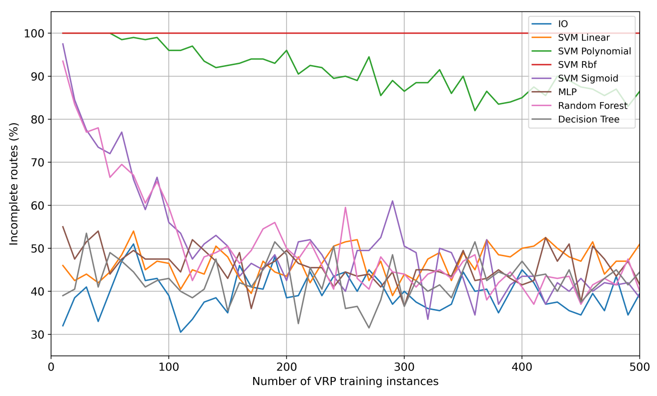

Fig. 6. CVRP (random locations): Incomplete route development during training using global policy training with no feature vector transformation. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

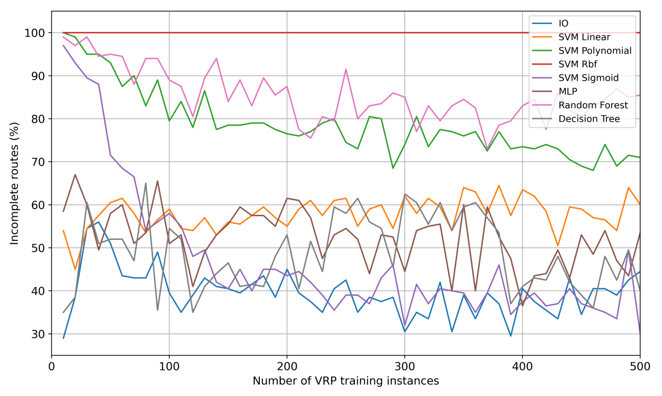

Fig. 7. CVRP (random locations): Incomplete route development during training using global policy training with feature vector reordering. Evaluated over 200 CVRP test instances after each batch of 10 CVRP training instances.

TABLE III
CVRP (random locations): Average routing cost and percentage of incomplete routes per model using global policy training with different feature vector transformations. Trained over 500 TSP training instances and evaluated over 200 TSP test instances.

State Feature Vector Transformation	**Optimal**	**Greedy**	**IO**	**SVM Linear**	**SVM Polynomial**	**SVM Rbf**	**SVM Sigmoid**	**MLP**	**Random Forest**	**Decision Tree**
	Average Routing Cost									
None	3.90	4.76	**4.74**	**4.78**	**5.39**	**7.49**	4.96	**4.69**	**4.92**	**5.02**
Reorder	3.90	4.76	4.86	4.97	5.46	**7.49**	**4.95**	4.87	5.42	5.11
	Percentage of Incomplete Routes									
None	0%	29%	**35%**	**43%**	75%	**100%**	48%	**35%**	**38%**	**41%**
Reorder	0%	29%	43%	56%	**71%**	**100%**	**34%**	49%	**38%**	42%

most of the models follow a similar learning curve when considering the development of the average routing cost, where in the first initial batches, the models rapidly decrease their average routing costs, after which this decline slows down and only lowers very slowly for the remainder of the training instances. Furthermore, for both problems, we see that some models cannot improve their performance during training or stop improving on a relatively high cost value. We mainly observed this with the SVM Sigmoid model for both problems and in the case of the CVRP with the SVM Rbf model when using a global policy training approach.

The objective of the models was to learn a policy resembling that of the expert, which was the optimal solver in this case. To do this, the models had to learn the costs of routing decisions (in this case, equal to the distances) depending on the features and actions. For both problems, we tested the model's ability to learn from expert examples when dealing with fixed as well as random customer locations. Although the performance in terms of the average routing cost for some models is better than that of others, overall, we can observe that for both the fixed and random locations, most models were able to learn this decision-making up to a certain performance level. However, we also observed that there was a difference in this performance level between the problems with fixed and random locations in both problems.

In the case of the TSP with fixed locations, we found that all models were able to imitate the expert perfectly. This is not surprising as all problem instances and thus solutions are the same. However, in the case of the CVRP with fixed locations, where there are differences between the problem instances, the average routing costs of almost all models get close to the optimal solution, which was used as expert examples. Additionally, all models perform considerably better than a greedy routing approach. For both problems with random customer locations, this is different. Here we observe that the gap between the average routing cost for the optimal solution and that of the different models is larger, with most models achieving average routing costs somewhat higher than the greedy strategy and only a few models obtaining a better result.

The difference in performance level can be explained by the large difference in the edges for the problems with fixed and random customer locations. Consider the CVRP, when using fixed locations, the customers are always in the same place for every problem instance. Therefore, the edges, which connect the customers, and their costs are constant for every problem instance. In our experiments, we used a single depot and ten customers. This means that there are only 110 different edges for which the model has to learn the cost. If we look at the problem with random customer locations, this is very different. Here the edges and thus also their costs vary in every problem instance. Making it considerably more complicated to learn the costs as there is an extremely high number of different possibilities. In this case, the model needs to be able to interpret the new costs of a problem instance using the features provided, which might be more challenging for some of the models.

We notice a few things if we take a closer look at the different multiclass classification algorithms. In the case of the SVM, it is clear that the sigmoid kernel is not a suitable formulation for the type of problems we are trying to learn from and solve. For TSP, this function is barely able to improve its performance during training and also, in the CVRP, it is among the worst performing models of all. On the other hand, the polynomial and rbf kernel can both learn and perform well for the TSP with random locations. However, in the case of the CVRP, using the polynomial or rbf kernel results in some of the worst performances of all models. For this problem, a linear approach is the best performing option.

The MLP classification model is the best performing model we tested, having one of the lowest average routing costs in both problems. This demonstrates the ability of a deep learning approach using neural networks to learn and adapt to a different problem. However, the downside is that this model's training time is much higher than the other classifiers.

Comparing the performance of the Random Forest and Decision Tree models, we can see that, as expected, the Random forest model can obtain better results for almost all problems. This can be explained by the fact that the Decision tree only uses a single tree to make its decisions while the Random forest model combines multiple.

Finally, there is a difference in the way that the optimal solver and the models determine their solutions. For optimal solution, we have to solve the whole problem at once to determine the route, and if this problem changes, we have to do this again. For our trained models, this is not the case. These models are trained to make a decision based on the system's current state. Therefore, if the problem changes, this is just a different state. This would suggest that using this state-action formulation, the model's decision-making is faster than the approach of the optimal solver. Although this seems logical, we have not performed any specific testing to confirm this, so this would require further investigation.

VI. Conclusion

In our research, we formulated a state-action representation for the TSP and CVRP. We discussed how these routing problems can be interpreted as IO or multiclass classification problems evaluating the performance of different algorithms in imitating expert decision-making learned from examples. The characteristics of the problems mainly differed in terms of single vehicle routing and multi-vehicle routing with demands. Besides this, for both routing problems, we experimented with fixed and random customer locations between different problem instances. For the algorithms to learn to solve these routing problems, we propose multiple policy training approaches as well as state feature vector transformations that can be used based on the characteristics of the problems. These different training configurations are tested by an IO model and four multiclass classification models, some with multiple variations on training data sets consisting of state-action pairs for multiple different problem instances. The performance of the trained models is evaluated on new problem instances, for which we compare their performance relative to each other and to the optimal solution, which acted as the expert and was used to create the training data.

We demonstrated that we can use IO and multiclass classification to learn from example data and imitate expert decision-making. However, we also showed a large variation in performance depending on the problem, state features, algorithm formulations and training configuration. Furthermore, the performance development of the models stabilizes at a certain level above that of the expert, limiting perfect imitation.

For both the TSP and CVRP, our results show that the global policy training approach obtains the best results for most models. In the case of the TSP with random locations, this is true when combining this training approach with the feature vector reordering transformation. Furthermore, we found that overall the MLP model obtains the best performance for most problem setups. As we discussed, this model utilizes a neural network formulation, which is often used in other research on the application of machine learning for routing problems and has many advantages. However, these models are often complex and can require more data or time to train. Therefore, it is interesting to note that the performance of the MLP model was matched by some of the other models that use a simpler formulation, demonstrating their potential. However, the results also show that these simpler models are more susceptible to changes in problem formulation, features and training configurations, making them less versatile and flexible in use compared to the MLP model.

Focusing on the difference in performance considering problems with fixed or random customer locations, we can conclude that the best results are obtained when customer locations are fixed between problem instances. When the locations are random, the edges and thus also the costs of decisions vary for every instance. This created a more complex problem, making it harder for the models to learn the expert's decision-making.

Only little research has been performed on learning to solve routing problems from expert examples. Therefore, our research acts as one of the early steps in investigating the application of known algorithms for different routing problem formulations. Based on our findings, we have some recommendations for further research.

We explained that the random locations represent a universal scenario for learning to make routing decisions based on general preferences, and the fixed locations represent a more area-specific application. The results of our research suggest that this second scenario might be more suitable for learning from expert examples. So for an area-specific application where the emphasis is on learning from experts and developing a policy for routing decisions in a specified region, one can consider designing a fixed graph that should represent an area layout, with only a few nodes representing consumers.

The customer positions can then change inside the graph, but the overall graph, and so the edges and costs, remain unchanged. In this circumstance, research could also consider more complex, area-specific routing costs, which are more representative of the real-world situation. Furthermore, the research might concentrate on other objectives of the routing problem or newly emerging customers during routing.

References

[1] L. Chen, Y. Chen, and A. Langevin, "An inverse optimization approach for a capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 295, no. 3, pp. 1087–1098, 12 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0377221721002575

[2] R. Canoy, V. Bucarey, J. Mandi, and T. Guns, "Learn-n-Route: Learning implicit preferences for vehicle routing," 1 2021.

[3] M. Nazari, A. Oroojlooy, L. V. Snyder, and M. Takáč, "Reinforcement Learning for Solving the Vehicle Routing Problem," 2 2018. [Online]. Available: http://arxiv.org/abs/1802.04240

[4] W. Kool, H. van Hoof, and M. Welling, "Attention, Learn to Solve Routing Problems!" 3 2018. [Online]. Available: http://arxiv.org/abs/1803.08475

[5] H. Lu, X. Zhang, and S. Yang, "A Learning-based Iterative Method for Solving Vehicle Routing Problems," 2020.

[6] E. Klein, M. Geist, B. Piot, and O. Pietquin, "Inverse Reinforcement Learning through Structured Classification," F. Pereira, C. J. C. Burges, and L. BottouK. Q. Weinberger, Eds. Curran Associates, Inc., 2012.

[7] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," *Twenty-first international conference on Machine learning - ICML '04*, p. 1, 2004.

[8] S. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques." *Informatica (Slovenia)*, vol. 31, pp. 249–268, 7 2007.

[9] D. Burton and P. L. Toint, "On an Instance of the Inverse Shortest Paths Problem," *Math. Program.*, vol. 53, no. 1–3, pp. 45–61, 1 1992.

[10] S. A. Akhtar, A. S. Kolarijani, and P. M. Esfahani, "Learning for Control: An Inverse Optimization Approach," *IEEE Control Systems Letters*, vol. 6, pp. 187–192, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9336679/

[11] Jianzhong Zhang and Zhenhong Liu, "Calculating some inverse linear programming problems," *Journal of Computational and Applied Mathematics*, vol. 72, no. 2, pp. 261–273, 8 1996.

[12] R. K. Ahuja and J. B. Orlin, "Inverse Optimization," *Operations Research*, vol. 49, no. 5, pp. 771–783, 10 2001.

[13] A. J. Schaefer, "Inverse integer programming," *Optimization Letters*, vol. 3, no. 4, pp. 483–489, 9 2009.

[14] L. Wang, "Cutting plane algorithms for the inverse mixed integer linear programming problem," *Operations Research Letters*, vol. 37, no. 2, pp. 114–116, 3 2009.

[15] D. S. Hochbaum, "Efficient Algorithms for the Inverse Spanning-Tree Problem," *Operations Research*, vol. 51, no. 5, pp. 785–797, 10 2003.

[16] P. M. Esfahani, S. Shafieezadeh-Abadeh, G. A. Hanasusanto, and D. Kuhn, "Data-driven Inverse Optimization with Imperfect Information," 12 2015. [Online]. Available: http://arxiv.org/abs/1512.05489

[17] A. Aswani, Z.-J. M. Shen, and A. Siddiq, "Inverse Optimization with Noisy Data," 7 2015. [Online]. Available: http://arxiv.org/abs/1507.03266

[18] G. Neumann-Denzau and J. Behrens, "Inversion of seismic data using tomographical reconstruction techniques for investigations of laterally inhomogeneous media," *Geophysical Journal International*, vol. 79, no. 1, pp. 305–315, 10 1984.

[19] Z. Erkin, M. D. Bailey, L. M. Maillart, A. J. Schaefer, and M. S. Roberts, "Eliciting Patients' Revealed Preferences: An Inverse Markov Decision Process Approach," *Decision Analysis*, vol. 7, no. 4, pp. 358–365, 12 2010.

[20] T. C. Y. Chan, T. Craig, T. Lee, and M. B. Sharpe, "Generalized Inverse Multiobjective Optimization with Application to Cancer Therapy," *Operations Research*, vol. 62, no. 3, pp. 680–695, 6 2014.

[21] L. J. Ratliff, R. Dong, H. Ohlsson, and S. S. Sastry, "Incentive Design and Utility Learning via Energy Disaggregation," 12 2013.

[22] J. Saez-Gallego, J. M. Morales, M. Zugno, and H. Madsen, "A Data-Driven Bidding Model for a Cluster of Price-Responsive Consumers of Electricity," *IEEE Transactions on Power Systems*, vol. 31, no. 6, pp. 5001–5011, 11 2016.

[23] S. I. You, J. Y. Chow, and S. G. Ritchie, "Inverse vehicle routing for activity-based urban freight forecast modeling and city logistics," *Transportmetrica A: Transport Science*, vol. 12, no. 7, pp. 650–673, 8 2016.

[24] S. J. Xu, M. Nourinejad, X. Lai, and J. Y. Chow, "Network learning via multiagent inverse transportation problems," *Transportation Science*, vol. 52, no. 6, pp. 1347–1364, 11 2018.

[25] M. Rönnqvist, G. Svenson, P. Flisberg, and L.-E. Jönsson, "Calibrated Route Finder: Improving the Safety, Environmental Consciousness, and Cost Effectiveness of Truck Routing in Sweden," *Interfaces*, vol. 47, no. 5, pp. 372–395, 10 2017.

[26] Chih-Wei Hsu and Chih-Jen Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 3 2002.

[27] M. Bourel and A. Segura, "Multiclass classification methods in ecology," *Ecological Indicators*, vol. 85, pp. 1012–1021, 2 2018.

[28] L. A. Berrueta, R. M. Alonso-Salces, and K. Héberger, "Supervised pattern recognition in food analysis," *Journal of Chromatography A*, vol. 1158, no. 1-2, pp. 196–214, 7 2007.

[29] J. Lu, G. Getz, E. A. Miska, E. Alvarez-Saavedra, J. Lamb, D. Peck, A. Sweet-Cordero, B. L. Ebert, R. H. Mak, A. A. Ferrando, J. R. Downing, T. Jacks, H. R. Horvitz, and T. R. Golub, "MicroRNA expression profiles classify human cancers," *Nature*, vol. 435, no. 7043, pp. 834–838, 6 2005.

[30] W. Akbar, W.-P. Wu, M. Faheem, S. Saleem, A. Javed, and M. A. Saleem, "Predictive Analytics Model Based on Multiclass Classification for Asthma Severity by Using Random Forest Algorithm," in *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*. IEEE, 6 2020, pp. 1–4.

[31] S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C.-H. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. P. Mesirov, T. Poggio, W. Gerald, M. Loda, E. S. Lander, and T. R. Golub, "Multiclass cancer diagnosis using tumor gene expression signatures," *Proceedings of the National Academy of Sciences*, vol. 98, no. 26, pp. 15 149–15 154, 12 2001.

[32] A. L. Duca, C. Bacciu, and A. Marchetti, "A K-nearest neighbor classifier for ship route prediction," in *OCEANS 2017 - Aberdeen*. IEEE, 6 2017, pp. 1–6.

[33] A. Lo Duca and A. Marchetti, "Exploiting multiclass classification algorithms for the prediction of ship routes: a study in the area of Malta," *Journal of Systems and Information Technology*, vol. 22, no. 3, pp. 289–307, 12 2020.

[34] J. Mandi, R. Canoy, V. Bucarey, and T. Guns, "Data Driven VRP: A Neural Network Model to Learn Hidden Preferences for VRP," 8 2021.

[35] Gurobi Optimization LLC, "Gurobi Optimizer Reference Manual," 2022.

[36] P. Zattoni Scroccaro, B. Atasoy, and P. Mohajerin Esfahani, "Data-driven Inverse Optimization: Loss Function, Algorithms and Sequential Decision-Making Problems," *[Unpublished paper] Delft University of Technology*, 2021.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in {P}ython," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

B
Binary Classification for Multiclass Classification Problems

Splitting the multiclass classification data set into multiple binary classification data sets and fitting a binary classification model to each is a way of employing binary classification algorithms for multiclassification problems. Two approaches are the One-vs-Rest (OvR) and One-vs-One methods (OvO).

In the OvR approach, a binary classifier is trained for each individual class. For each classifier, the class is fitted against all the other classes. Each of these classifiers aims to predict the probability that a certain sample belongs to its respective class. Then the class with the highest overall probability score will be predicted as the class which belongs to the sample.

Like the OvR approach, the OvO approach trains multiple classifiers. However, a binary classifier is trained for each pair of classes in this case. This means that for a problem with N_{classes} number of different classes, the number of individual classifiers is determined as

$$\text{number of classifiers} = \frac{N_{\text{classes}}(N_{\text{classes}} - 1)}{2}, \tag{B.1}$$

which is significantly more than the OvR approach. Each classifier will determine a class for a certain sample, and the overall class predicted the most will be returned as the corresponding class for the sample. When two classes get an equal number of votes, it chooses the class with the highest aggregate classification probability by averaging the pair-wise classification probability levels computed by the underlying binary classifiers (Bishop, 2006).

C Varying Training Times for The Inverse Optimization Model

As mentioned in Chapter 6.1, other activities were performed on the device used during the experiments. These activities influenced the training time of the different models. This is shown in Table C.1, which shows the same experiment with the exact same configurations performed on two different days while different activities took place. This makes it hard to draw any firm conclusions from these results concerning the difference in training time between the different training configurations.

Table C.1: Training times for the IO model of the exact same experiment on different points in time

Training Date	**Training Approach**	**State Feature Vector Transformation**	**Training Time IO Model (s)**
25-07-2022 11:00	Global	None	6854
27-07-2022 11:07	Global	None	4197

D

TSP Results for Fixed Customer Locations

Figure D.1 shows each model's average overall routing cost during training on the TSP problem with fixed customer locations, using the global policy training without feature vector transformation. After each training batch of 10 TSP training instances, the performance of the models was evaluated over 200 different TSP test instances. Furthermore, Figure D.2 visualizes the performance after the training along with the standard deviation of each model for this training configuration compared to the average optimal and greedy routing costs. Note that when we consider the TSP with fixed locations, every individual problem instance in training as well as in testing is the same. Therefore, this result is expected and shows that all models can learn how to solve a single TSP problem instance from an example.

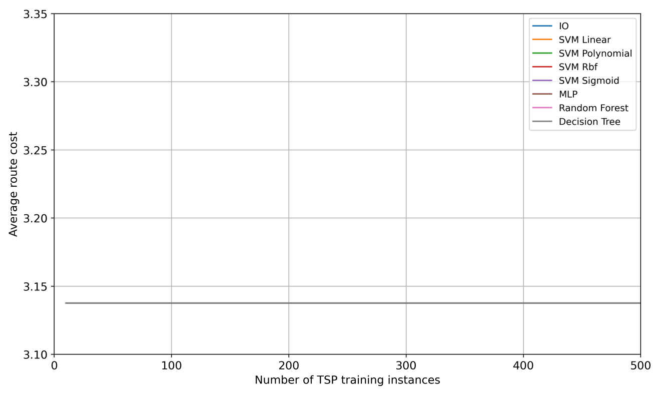

Figure D.1: TSP (fixed locations): Cost development during training using global policy training without feature vector transformation. Evaluated over 200 TSP test instances after each batch of 10 TSP training instances.

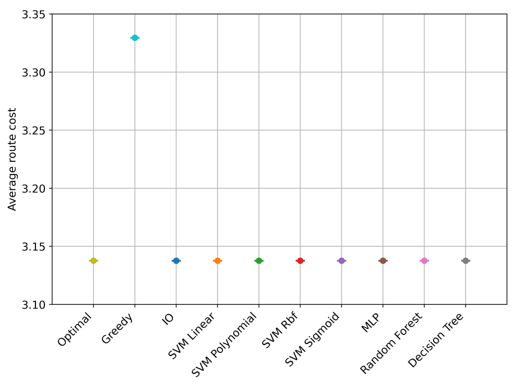

Figure D.2: TSP (fixed locations): Average routing costs with standard deviation per model after training over 500 TSP instances using global policy training without feature vector transformation. Evaluated over 200 TSP test instances.

E TSP Results for Random Customer Locations

This appendix shows the performances of the different models for each combination of different policy training approaches and feature vector transformations for the TSP formulation with random customer locations suggested in Chapter 3.1 (except the global policy training with feature vector reordering, as this is discussed in Chapter 6.1). All models were trained using an offline batch training approach on a total of 500 TSP training instances in batches of 10. The TSP consisted of 10 nodes, and the routing cost equals the distance.

Table E.1 provides the same overview of the performances per model tested on 200 TSP test instances for each different training configuration as Table 6.2.

Table E.1: TSP (random locations): Average routing cost per model for different policy training configurations. Trained over 500 TSP training instances and evaluated over 200 TSP test instances.

		Average Routing Cost						**Optimal** **Greedy**	2.83 3.08
Training Approach	**State Feature Vector Transformation**	**IO**	**SVM Linear**	**SVM Polynomial**	**SVM Rbf**	**SVM Sigmoid**	**MLP**	**Random Forest**	**Decision Tree**
Global	None	3.36	**3.05**	3.10	3.19	4.13	3.00	3.28	3.55
Global	Reorder	3.38	3.06	**3.02**	**3.08**	3.90	**2.97**	**3.13**	**3.44**
Decisions	None	3.49	3.33	3.20	3.35	**3.75**	3.18	3.39	3.64
Decisions	Reorder	3.35	3.12	3.09	3.13	3.77	3.03	3.23	3.57
Decisions	Reduce	**3.33**	3.11	3.09	3.11	3.77	3.01	3.21	**3.44**

The figures below show the development of the average routing cost for the different training configurations and the average routing cost and standard deviation tested after training. In these results, we can see that, for all training configurations, the development of the average routing cost during training follows a similar curve for most models, mainly differing in the rate of decline.

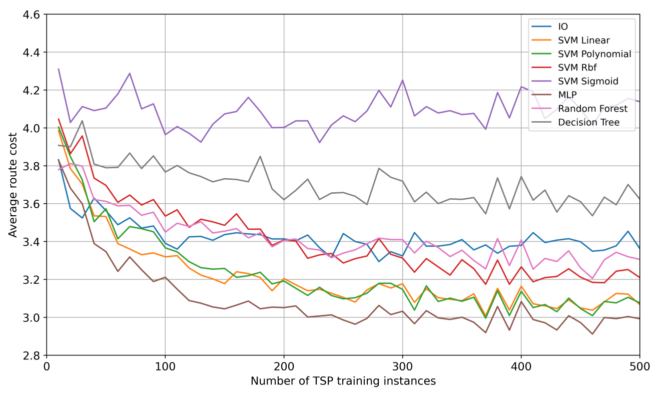

Figure E.1: TSP (random locations): Cost development during training using global policy training without feature vector transformation. Evaluated over 200 TSP test instances after each batch of 10 TSP training instances.

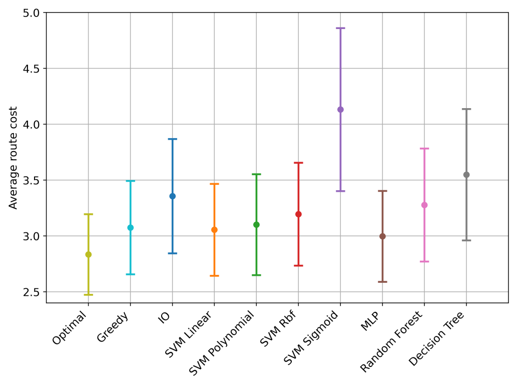

Figure E.2: TSP (random locations): Average routing costs with standard deviation per model after training over 500 TSP instances using global policy training without feature vector transformation. Evaluated over 200 TSP test instances.

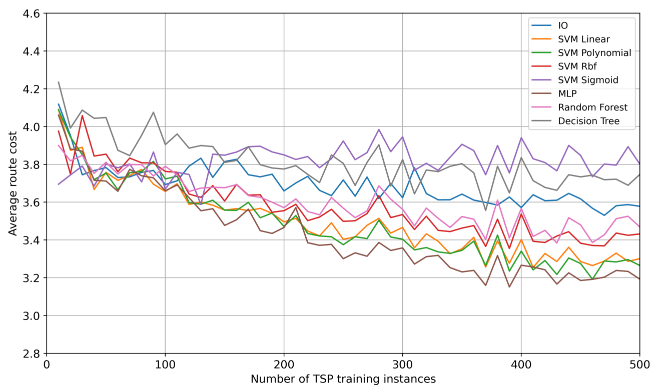

Figure E.3: TSP (random locations): Cost development during training using decisions policy training without feature vector transformation. Evaluated over 200 TSP test instances after each batch of 10 TSP training instances.

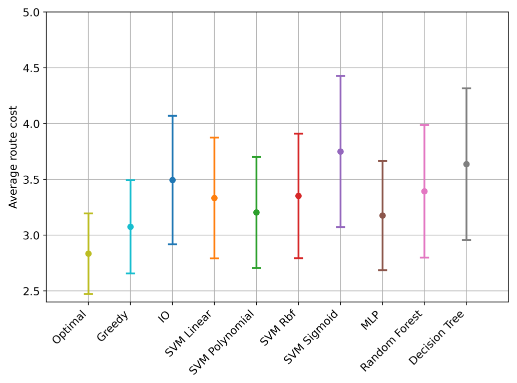

Figure E.4: TSP (random locations): Average routing costs with standard deviation per model after training over 500 TSP instances using decisions policy training without feature vector transformation. Evaluated over 200 TSP test instances.

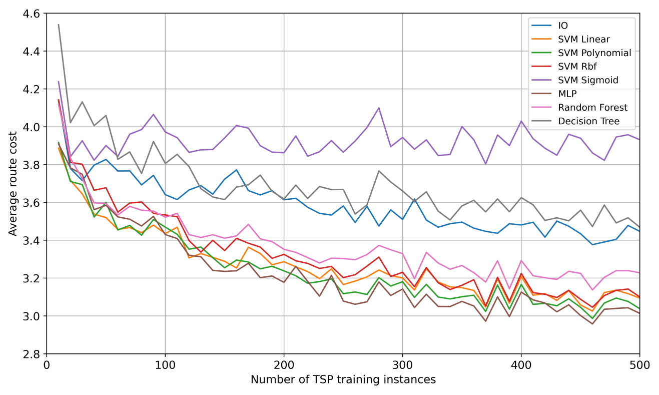

Figure E.5: TSP (random locations): Cost development during training using decisions policy training and feature vector reordering. Evaluated over 200 TSP test instances after each batch of 10 TSP training instances.

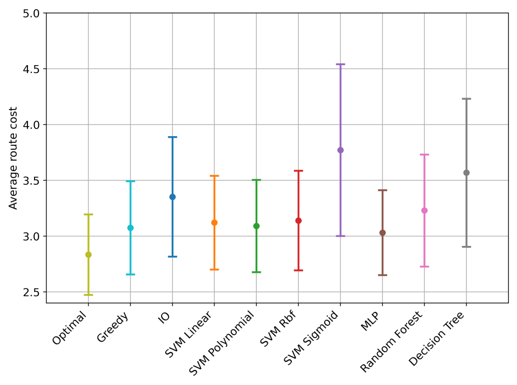

Figure E.6: TSP (random locations): Average routing costs with standard deviation per model after training over 500 TSP instances using decisions policy training and feature vector reordering. Evaluated over 200 TSP test instances.

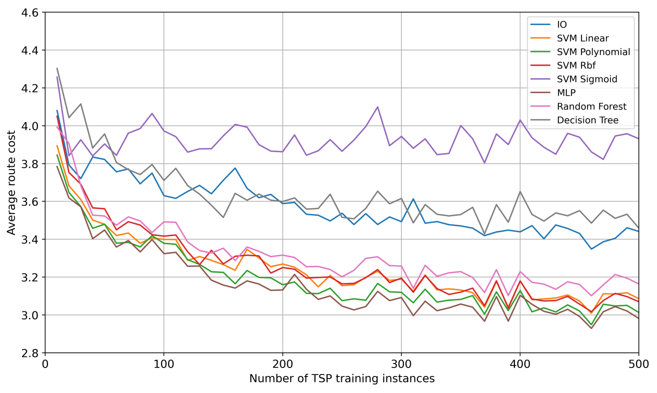

Figure E.7: TSP (random locations): Cost development during training using decisions policy training and feature vector reduction. Evaluated over 200 TSP test instances after each batch of 10 TSP training instances.

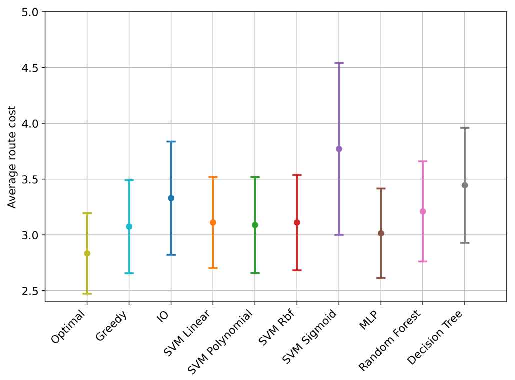

Figure E.8: TSP (random locations): Average routing costs with standard deviation per model after training over 500 TSP instances using decisions policy training and feature vector reduction. Evaluated over 200 TSP test instances.

Bibliography

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. *Twenty-first international conference on Machine learning - ICML '04*, 1.

Agarwal, R., & Ergun, Ö. (2010). Network Design and Allocation Mechanisms for Carrier Alliances in Liner Shipping. *Operations Research*, *58*(6), 1726–1742.

Ahuja, R. K., & Orlin, J. B. (2001). Inverse Optimization. *Operations Research*, *49*(5), 771–783.

Akbar, W., Wu, W.-P., Faheem, M., Saleem, S., Javed, A., & Saleem, M. A. (2020). Predictive Analytics Model Based on Multiclass Classification for Asthma Severity by Using Random Forest Algorithm. *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, 1–4.

Akhtar, S. A., Kolarijani, A. S., & Esfahani, P. M. (2022). Learning for Control: An Inverse Optimization Approach. *IEEE Control Systems Letters*, *6*, 187–192.

Aswani, A., Shen, Z.-J. M., & Siddiq, A. (2015). Inverse Optimization with Noisy Data.

Bärmann, A., Martin, A., Pokutta, S., & Schneider, O. (2018). An Online-Learning Approach to Inverse Optimization.

Bärmann, A., Pokutta, S., & Schneider, O. (2017). Emulating the Expert: Inverse Optimization through Online Learning. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (pp. 400–410). PMLR.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural Combinatorial Optimization with Reinforcement Learning.

Benediktsson, J., Swain, P., & Ersoy, O. (1990). Neural Network Approaches Versus Statistical Methods In Classification Of Multisource Remote Sensing Data. *IEEE Transactions on Geoscience and Remote Sensing*, *28*(4), 540–552.

Berrueta, L. A., Alonso-Salces, R. M., & Héberger, K. (2007). Supervised pattern recognition in food analysis. *Journal of Chromatography A*, *1158*(1-2), 196–214.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag.

Bourel, M., & Segura, A. (2018). Multiclass classification methods in ecology. *Ecological Indicators*, *85*, 1012–1021.

Boyd, S., & Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Breiman, L. (1998). Arcing classifier. *The Annals of Statistics*, *26*(3).

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (2017). *Classification And Regression Trees*. Routledge.

Burton, D., & Toint, P. L. (1992). On an Instance of the Inverse Shortest Paths Problem. *Math. Program.*, *53*(1–3), 45–61.

Canoy, R., Bucarey, V., Mandi, J., & Guns, T. (2021). Learn-n-Route: Learning implicit preferences for vehicle routing.

Chan, T. C. Y., Craig, T., Lee, T., & Sharpe, M. B. (2014). Generalized Inverse Multiobjective Optimization with Application to Cancer Therapy. *Operations Research*, *62*(3), 680–695.

Chen, L., Chen, Y., & Langevin, A. (2021). An inverse optimization approach for a capacitated vehicle routing problem. *European Journal of Operational Research*, *295*(3), 1087–1098.

Chih-Wei Hsu, & Chih-Jen Lin. (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, *13*(2), 415–425.

Chow, J. Y., & Recker, W. W. (2012). Inverse optimization with endogenous arrival time constraints to calibrate the household activity pattern problem. *Transportation Research Part B: Methodological*, *46*(3), 463–479.

Clarke, G., & Wright, J. W. (1964). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, *12*(4), 568–581.

Cook, W. J., Applegate, D. L., Bixby, R. E., & Chvátal, V. (2011). *The Traveling Salesman Problem*. Princeton University Press.

Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, *20*(3), 273–297.

Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America*, *2*(4), 393–410.

Dantzig, G. B., & Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, *6*(1), 80–91.

Duca, A. L., Bacciu, C., & Marchetti, A. (2017). A K-nearest neighbor classifier for ship route prediction. *OCEANS 2017 - Aberdeen*, 1–6.

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

Elshaer, R., & Awad, H. (2020). A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers & Industrial Engineering*, *140*, 106242.

Erkin, Z., Bailey, M. D., Maillart, L. M., Schaefer, A. J., & Roberts, M. S. (2010). Eliciting Patients' Revealed Preferences: An Inverse Markov Decision Process Approach. *Decision Analysis*, *7*(4), 358–365.

Esfahani, P. M., Shafieezadeh-Abadeh, S., Hanasusanto, G. A., & Kuhn, D. (2015). Data-driven Inverse Optimization with Imperfect Information.

Gardner, M., & Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, *32*(14-15), 2627–2636.

Gurobi Optimization LLC. (2022). Gurobi Optimizer Reference Manual.

Ho, S. C., Szeto, W. Y., Kuo, Y.-H., Leung, J. M. Y., Petering, M., & Tou, T. W. H. (2018). *A Survey of Dial-a-ride Problems: Literature Review and Recent Developments* (tech. rep.).

Ho, Y., & Wookey, S. (2020). The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling. *IEEE Access*, *8*, 4806–4813.

Hochbaum, D. S. (2003). Efficient Algorithms for the Inverse Spanning-Tree Problem. *Operations Research*, *51*(5), 785–797.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated.

Jianzhong Zhang, & Zhenhong Liu. (1996). Calculating some inverse linear programming problems. *Journal of Computational and Applied Mathematics*, *72*(2), 261–273.

Jünger, M., Reinelt, G., & Rinaldi, G. (1995). Chapter 4 The traveling salesman problem.

Klein, E., Geist, M., Piot, B., & Pietquin, O. (2012). Inverse Reinforcement Learning through Structured Classification. In F. Pereira, C. J. C. Burges, & L. BottouK. Q. Weinberger (Eds.). Curran Associates, Inc.

Konstantakopoulos, G. D., Gayialis, S. P., & Kechagias, E. P. (2020). Vehicle routing problem and related algorithms for logistics distribution: a literature review and classification. *Operational Research*.

Kool, W., van Hoof, H., & Welling, M. (2018). Attention, Learn to Solve Routing Problems!

Kotsiantis, S. (2007). Supervised Machine Learning: A Review of Classification Techniques. *Informatica (Slovenia)*, *31*, 249–268.

Lo Duca, A., & Marchetti, A. (2020). Exploiting multiclass classification algorithms for the prediction of ship routes: a study in the area of Malta. *Journal of Systems and Information Technology*, *22*(3), 289–307.

Lu, H., Zhang, X., & Yang, S. (2020). A Learning-based Iterative Method for Solving Vehicle Routing Problems.

Lu, J., Getz, G., Miska, E. A., Alvarez-Saavedra, E., Lamb, J., Peck, D., Sweet-Cordero, A., Ebert, B. L., Mak, R. H., Ferrando, A. A., Downing, J. R., Jacks, T., Horvitz, H. R., & Golub, T. R. (2005). MicroRNA expression profiles classify human cancers. *Nature*, *435*(7043), 834–838.

Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective Approaches to Attention-based Neural Machine Translation.

Mandi, J., Canoy, R., Bucarey, V., & Guns, T. (2021). Data Driven VRP: A Neural Network Model to Learn Hidden Preferences for VRP.

Nazari, M., Oroojlooy, A., Snyder, L. V., & Takáč, M. (2018). Reinforcement Learning for Solving the Vehicle Routing Problem.

Neumann-Denzau, G., & Behrens, J. (1984). Inversion of seismic data using tomographical reconstruction techniques for investigations of laterally inhomogeneous media. *Geophysical Journal International*, *79*(1), 305–315.

Oyola, J., Arntzen, H., & Woodruff, D. L. (2017a). The stochastic vehicle routing problem, a literature review, part I: models. *EURO Journal on Transportation and Logistics*, *7*(3), 193–221.

Oyola, J., Arntzen, H., & Woodruff, D. L. (2017b). The stochastic vehicle routing problem, a literature review, Part II: solution methods. *EURO Journal on Transportation and Logistics*, *6*(4), 349–388.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in {P}ython. *Journal of Machine Learning Research*, *12*, 2825–2830.

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

Ramaswamy, S., Tamayo, P., Rifkin, R., Mukherjee, S., Yeang, C.-H., Angelo, M., Ladd, C., Reich, M., Latulippe, E., Mesirov, J. P., Poggio, T., Gerald, W., Loda, M., Lander, E. S., & Golub, T. R. (2001). Multiclass cancer diagnosis using tumor gene expression signatures. *Proceedings of the National Academy of Sciences*, *98*(26), 15149–15154.

Ratliff, L. J., Dong, R., Ohlsson, H., & Sastry, S. S. (2013). Incentive Design and Utility Learning via Energy Disaggregation.

Recker, W. (1995). The household activity pattern problem: General formulation and solution. *Transportation Research Part B: Methodological*, *29*(1), 61–77.

Rönnqvist, M., Svenson, G., Flisberg, P., & Jönsson, L.-E. (2017). Calibrated Route Finder: Improving the Safety, Environmental Consciousness, and Cost Effectiveness of Truck Routing in Sweden. *Interfaces*, *47*(5), 372–395.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Rummukainen, H. (2021). Inverse Optimization for Warehouse Management.

Saez-Gallego, J., Morales, J. M., Zugno, M., & Madsen, H. (2016). A Data-Driven Bidding Model for a Cluster of Price-Responsive Consumers of Electricity. *IEEE Transactions on Power Systems*, *31*(6), 5001–5011.

Schaefer, A. J. (2009). Inverse integer programming. *Optimization Letters*, *3*(4), 483–489.

Sharma, S. K., Routroy, S., & Yadav, U. (2018). Vehicle routing problem: recent literature review of its variants. *International Journal of Operational Research*, *33*(1), 1–31.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks.

Toth, P., & Vigo, D. (2002). *The Vehicle Routing Problem* (P. Toth & D. Vigo, Eds.). Society for Industrial; Applied Mathematics.

Troutt, M. D., Pang, W.-K., & Hou, S.-H. (2006). Behavioral Estimation of Mathematical Programming Objective Function Coefficients. *Management Science*, *52*(3), 422–434.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer Networks.

Wang, L. (2009). Cutting plane algorithms for the inverse mixed integer linear programming problem. *Operations Research Letters*, *37*(2), 114–116.

Xu, S. J., Nourinejad, M., Lai, X., & Chow, J. Y. (2018). Network learning via multiagent inverse transportation problems. *Transportation Science*, *52*(6), 1347–1364.

You, S. I., Chow, J. Y., & Ritchie, S. G. (2016). Inverse vehicle routing for activity-based urban freight forecast modeling and city logistics. *Transportmetrica A: Transport Science*, *12*(7), 650–673.

Zattoni Scroccaro, P., Atasoy, B., & Mohajerin Esfahani, P. (2021). Data-driven Inverse Optimization: Loss Function, Algorithms and Sequential Decision-Making Problems. *[Unpublished paper] Delft University of Technology*.

Zhang, H., Ge, H., Yang, J., & Tong, Y. (2022). Review of Vehicle Routing Problems: Models, Classification and Solving Algorithms. *Archives of Computational Methods in Engineering*, *29*(1), 195–221.

Zheng, J., Gao, Z., Yang, D., & Sun, Z. (2015). Network Design and Capacity Exchange for Liner Alliances with Fixed and Variable Container Demands. *Transportation Science*, *49*(4), 886–899.