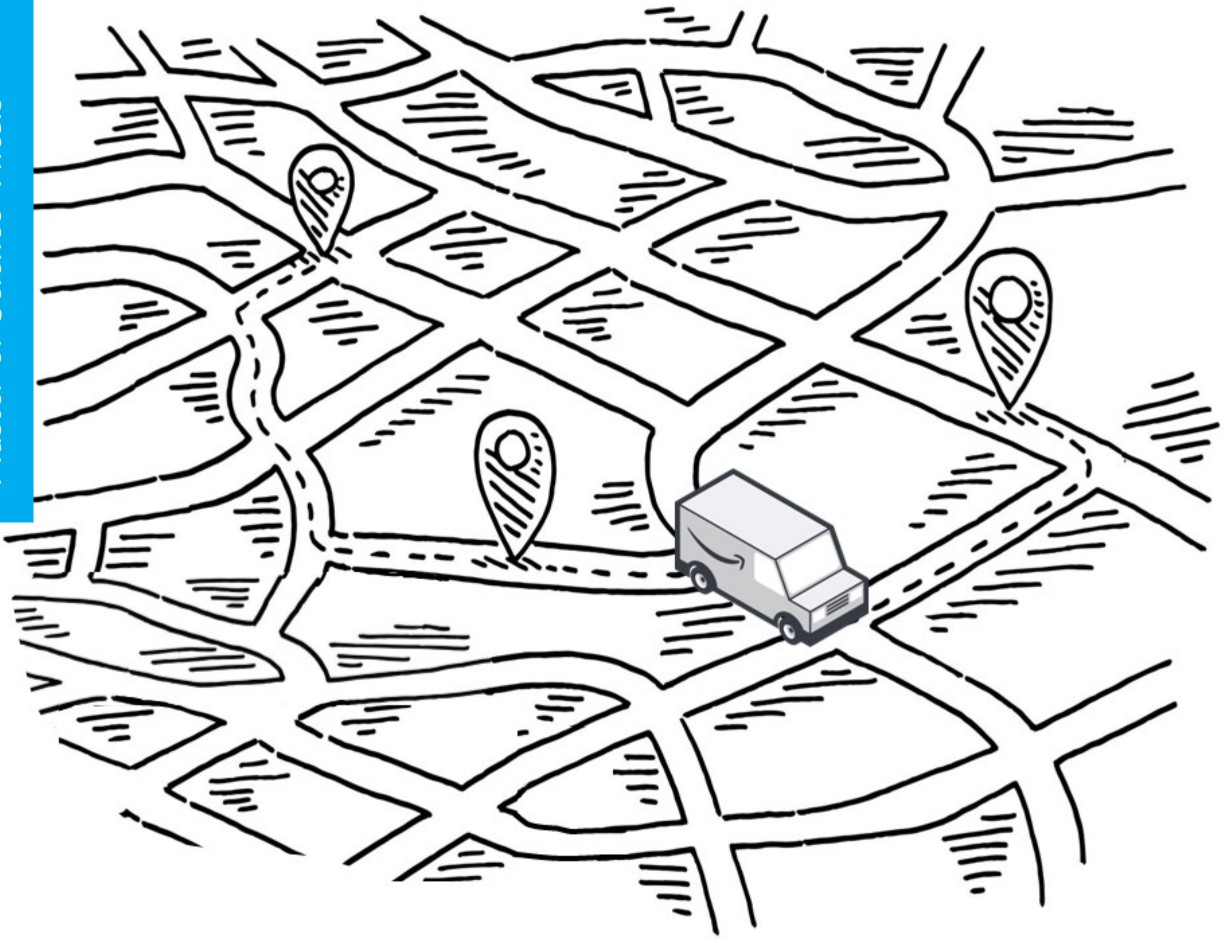
Learning Drivers' Preferences in Delivery Route Planning

an Inverse Optimization Approach

Piet van Beek

Master of Science Thesis



Delft University of Technology

Delft Center for Systems and Control

Learning Drivers' Preferences in Delivery Route Planning

an Inverse Optimization Approach

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Piet van Beek

November 2, 2022

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Copyright © Delft Center for Systems and Control (DCSC)
All rights reserved.

Abstract

Optimizing delivery routes is a well-researched topic, however, most of the classical approaches do not incorporate preferences of drivers, as those approaches focus on minimizing the time or distance of the routes. As a result, the actual driven route of an experienced driver often deviates from the proposed route since the drivers have tacit knowledge about the real-life conditions of the road network. Amazon proposed a challenge to learn a delivery route planning strategy from historically driven routes and thus incorporate this tacit knowledge. In this thesis, we will tackle the challenge using data-driven inverse optimization to learn the zone sequencing patterns of drivers. The zone sequences of expert drivers are assumed to be the solutions to a traveling salesman problem (TSP) in which the weights represent the preference of a driver to use a certain edge. The values of the weights will be learned through inverse optimization. Our final approach achieves a score that ranks 4th out of the 48 models that qualified for the final round of the challenge.

Keywords: *Amazon Last Mile Routing Challenge, Inverse Optimization, Traveling Salesman Problem, Weights Optimization*

Table of Contents

Acknowledgments

I would like to thank my supervisor Peyman Mohajerin Esfahani for his assistance during the writing of this thesis, his enthusiasm about the subject and his critical feedback. Furthermore, I would like to thank my daily supervisor Pedro Zattoni Scroccaro for his guidance, time and enjoyable discussions. His enthusiasm and knowledge about the subject greatly improved my interest and understanding of the topic.

Delft, University of Technology
November 2, 2022

Piet van Beek

Chapter 1

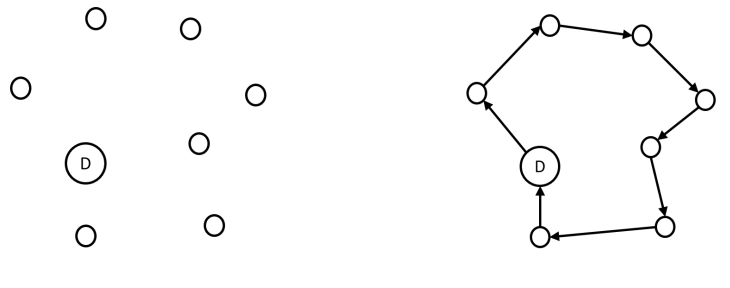
Introduction

Last-mile delivery refers to the last stage of post and package delivery in which drivers bring packages from depots to customers. A lot of research has been conducted into the optimization of delivery routes. Most of the existing approaches focus on minimizing the time or distance of a proposed route. In practice, however, the drivers' actual driven routes appear to diverge from the suggested routes. This is a result of the drivers' many preferences, such as finding good parking spaces and avoiding curvy or slow-moving streets. In traditional optimization strategies, it is hard to account for those preferences. Hence, the tacit knowledge of the experienced drivers, potentially resulting in more convenient routes in real-life operational conditions, is not taken into consideration. Amazon proposed a challenge that encouraged contestants to develop learning techniques that use the historical route data of experienced drivers to determine safer and more efficient routes. This challenge with the provided data set will be the focus of this thesis (https://routingchallenge.mit.edu/).

In most cases, last-mile delivery can be described by a vehicle routing problem (VRP). Dantzig et al. [1] introduced VRP by proposing a routing method for a fleet of trucks delivering gasoline from a depot to several service stations while attempting to minimize the total number of miles covered by the fleet. Due to the rapid growth of the logistics industry, a lot of research has been conducted in this field [2]. Efficient route planning could result in a significant reduction of costs for companies like Amazon.

The goal of a VRP is to optimize the route of drivers from one or more depots to multiple customers in different locations under certain constraints [2]. There are two important specifications of a VRP. First of all, each customer should be visited only once by a vehicle. Secondly, all vehicles start from a certain depot and return to the same depot after finishing their task.

A special case of the VRP is the traveling salesman problem (TSP). In the case of a TSP, a single driver leaves from and finishes at a single depot. Again, the driver has to visit all customers once and once only. For the Amazon challenge, the problem can be classified as a TSP since only one driver has to visit a predefined set of stops leaving from a certain depot in which it has to finish its route as well. A typical example of a solution diagram to a TSP is given in Figure 1-1.



(a) Example problem. **(b)** Example solution.

Figure 1-1: Example of a TSP diagram.

1-1 TSP State-of-the-Art Methods

The TSP is one of the most studied problems in combinatorial optimization. Although the problem is simple to state, the TSP is known to be an NP-hard problem. Throughout the years, several approaches have been proposed to solve the TSP, both exact solutions and heuristics. Laporte et al. [3] give a review of some of the most promising exact and heuristic methods.

Exact methods to solve a TSP are computationally expensive as they (implicitly) have to check all solutions to find the optimal solution [4]. Most exact approaches rely on the formulation of an integer linear program (ILP). Other exact approaches include dynamic programming, branch and bound algorithms and cutting plane methods [5]. On the other hand, heuristics can identify a close-to-optimal solution more efficiently but cannot ensure optimality. These techniques include genetic algorithms, tabu search, simulated annealing, greedy methods and ant colony algorithms [6].

1-2 Learning Drivers' Preferences

The state-of-the-art methods for solving a TSP mentioned in the previous section are able to find (close-to-)optimal solutions for a basic TSP, in which the time or distance of a proposed route is minimized. However, the actual driven routes of drivers often differ from the proposed routes, since the proposed routes do not account for preferences of drivers such as wide streets, good parking spots and support facilities along the route. Including the tacit knowledge of the experienced drivers may improve the efficiency of the routes, because the theoretically estimated costs of traveling may not represent the costs under real-life operational conditions due to stochasticity and time dependency. The knowledge of experienced drivers could account for those suboptimalities and result in safer and more efficient routes [7].

An alteration of the VRP capable of capturing those preferences is called rich VRP. This alteration includes multi-objective optimization programs and the addition of multiple constraints which relate to several real-life factors, such as inventory, scheduling and preferences of drivers or companies [8]. The TSP version of this alteration is called multi-objective TSP (MOTSP) [9]. Both alterations, however, increase computation times and heavily depend on human input as companies have to construct the constraints and objective functions manually. The information from historical routing data, in which the drivers drove according to their preferences, could provide a solution to this problem.

In literature, several methods aimed to learn from historical routing data in the planning of routes. Some of those methods use a discrete choice model in which the trajectories of the drivers are used to determine a transition probability matrix [10, 11]. Other methods use inverse reinforcement learning to determine a policy that matches the historical data [12, 13].

In this thesis, we will apply the concept of inverse optimization to incorporate the preferences of experienced drivers. In inverse optimization, a learner agent aims to imitate the behavior of an expert agent. The expert agent is assumed to compute an action given a certain signal by solving an optimization program with an unknown objective function. The goal of inverse optimization is for the learner agent to approximate this unknown objective function, given a sample set of signals and corresponding expert actions.

In literature, inverse optimization has been applied to transport problems. For example, several methods update the weights of the edges in a map with the use of inverse optimization. Burton et al. [14], András et al. [15] and Bärmann et al. [16] propose a method to find (a perturbation in) the original cost matrix of a shortest path problem such that the historic routes are the optimal solutions of the problem, whereas Chen et al. [17] aim to update the weights of a cost matrix of a VRP to incorporate preferences of drivers. Chung et al. [18] provide a general framework to find the minimum perturbation in a cost matrix, such that a TSP tour is the optimal solution to a TSP-solving algorithm on a given graph.

From those papers, it becomes clear that inverse optimization can be used for learning route planning strategies from historical routing data. Therefore, we will further investigate the use of inverse optimization applied to the TSP. Most of the approaches mentioned before mainly rely on the notion of inverse feasibility or duality, in which they assume there is a perfect model-data fit. In this thesis, we assume there is no perfect model-data fit. We will focus on data-driven inverse optimization in which loss functions are used to calculate the mismatch between the hypothesized objective function and the unknown expert objective function. The use of the loss functions proposed in this paper, to the best of the author's knowledge, has not been applied to real-life data sets of transport problems before.

The thesis is structured as follows. In Chapter 2 the Amazon Challenge and the corresponding data set will be introduced. Some preliminaries for the proposed method will be introduced in Chapter 3, after which in Chapter 4 the proposed method will be explained. The results will be presented in Chapter 5 and finally, the conclusions will be drawn in Chapter 6.

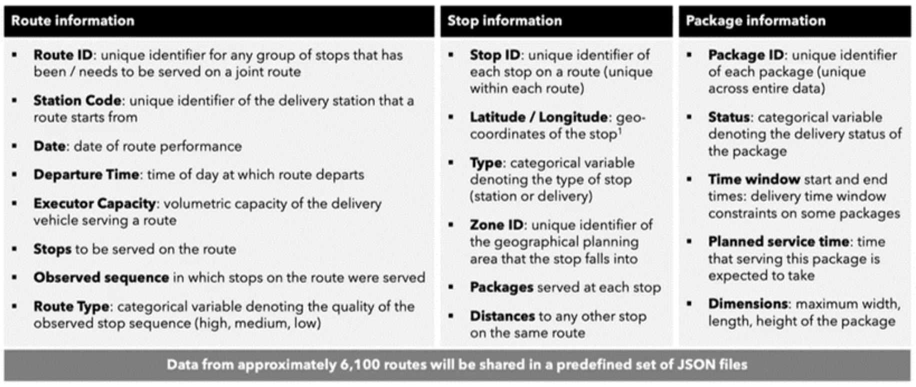
1-3 Notation

Throughout this thesis, we will use several notations. We denote a set of integers $\{1, \ldots, m\}$ with $[1, m]$. An inner product is defined as $\langle \cdot, \cdot \rangle$. A transpose of a matrix A is denoted as A'. In the case of elementwise multiplication between two vectors, we use the symbol $\odot$ and in the case of elementwise exponentiation, we use $\exp\{\}$. Samples of a data set will be denoted with the "hat" $(\hat{\cdot})$ notation. A set of indexed values is compactly denoted by $\{x_i\}_{i=1}^{N} := \{x_1, \ldots, x_N\}$.

Chapter 2

Amazon Challenge

In this chapter, the *Amazon Last Mile Routing Research Challenge* (from here on shortened to the Amazon Challenge) will be introduced. In 2021, Amazon.com, Inc. has published a data set of real-life delivery requests and the corresponding actual driven routes. Participants of the challenge were encouraged to develop learning techniques to build a model which mimics the routing behavior of the drivers as closely as possible using the provided data. The data set, the scoring criteria and the already proposed methods will be introduced in separate sections.



Route information	Stop information	Package information
▪ **Route ID**: unique identifier for any group of stops that has been / needs to be served on a joint route ▪ **Station Code**: unique identifier of the delivery station that a route starts from ▪ **Date**: date of route performance ▪ **Departure Time**: time of day at which route departs ▪ **Executor Capacity**: volumetric capacity of the delivery vehicle serving a route ▪ **Stops** to be served on the route ▪ **Observed sequence** in which stops on the route were served ▪ **Route Type**: categorical variable denoting the quality of the observed stop sequence (high, medium, low)	▪ **Stop ID**: unique identifier of each stop on a route (unique within each route) ▪ **Latitude / Longitude**: geo-coordinates of the stop[1] ▪ **Type**: categorical variable denoting the type of stop (station or delivery) ▪ **Zone ID**: unique identifier of the geographical planning area that the stop falls into ▪ **Packages** served at each stop ▪ **Distances** to any other stop on the same route	▪ **Package ID**: unique identifier of each package (unique across entire data) ▪ **Status**: categorical variable denoting the delivery status of the package ▪ **Time window** start and end times: delivery time window constraints on some packages ▪ **Planned service time**: time that serving this package is expected to take ▪ **Dimensions**: maximum width, length, height of the package

Data from approximately 6,100 routes will be shared in a predefined set of JSON files

Figure 2-1: Information provided by Amazon data set [19].

2-1 Amazon Data

The data set provided by Amazon is described in detail by Merchan et al. [20]. In this section, we will highlight its most important aspects. The data set contains information about 6112 route instances, in which, for every route instance, a set of stops (possibly with time windows) has to be visited by a single driver. For each route, information is provided regarding the stops, the packages and the vehicle. Furthermore, the actual driven route by the experienced driver is given. Lastly, Amazon scored every route based on the quality of the observed route as high, medium or low. Figure 2-1 gives an overview of which features are covered by the data set. Each of these routes originates at a depot and visits a collection of drop-off stops assigned to a single driver in advance. Thus, the expert human routes can be interpreted as the solutions to some kind of TSP (possibly with time windows).

	No. of historical routes			No. stops	No. zones	No. depots
	High	Medium	Low			
Los Angeles	1,393	1,438	57	414,440	12,410	6
Chicago	381	609	12	162,410	6,939	4
Seattle	542	521	16	155,781	4,677	3
Boston	306	610	13	140,622	8,361	3
Austin	96	114	4	31,274	3,198	1
Total	2,718	3,292	102			17

Figure 2-2: Information about the historical observed routes from the Amazon data categorized per city [7].

The stops are located in five cities in the United States: Los Angeles, Chicago, Seattle, Boston and Austin. Every city contains at least one depot and consists of multiple zones. Figure 2-2 shows information for each city in the data set. The zones are predefined clusters of stops. Every stop is given a zone ID which indicates to which zone a stop belongs. Every zone can just be visited from one depot. An example of all routes leaving from one certain depot is depicted in Figure 2-3, in which every color defines a certain route. Throughout the data set stops are generally not visited by more than one route in the data set, which can also be seen in Figure 2-3. This makes it difficult to learn any meaningful preferences of the drivers at stop level and is thus one of the main difficulties of this challenge.

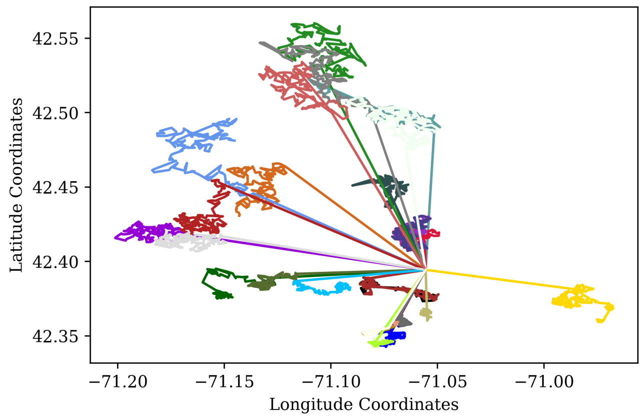


Figure 2-3: All actual driven routes leaving from depot DBO1 in Boston, every color defines a route.

2-2 Evaluation Criterium

As previously mentioned, the goal of the challenge is to incorporate the preferences of experienced drivers into the routing of last-mile delivery vehicles. Thus, rather than coming up with TSP strategies that minimize time or distance given a set of stops to be visited, the goal of the challenge is to learn from historical data how to route like the expert drivers. To this end, Amazon has also provided a data set of 3072 routes which all have a "high" quality, to evaluate the proposed approaches. Furthermore, Amazon provided a scoring metric that computes the similarity between the routes proposed by the approach of the participants and the actual driven routes [21]. The similarity score between a historically driven route A and an algorithm-produced route B is calculated by

$$\text{score} = \frac{\text{SD}(A, B) \cdot \text{ERP}_{\text{norm}}(A, B)}{\text{ERP}_{\text{e}}(A, B)} \tag{2-1}$$

in which SD denotes the sequence (route) deviation of B with respect to A. The other two terms are related to the measure *Edit Distance with Real Penalty* (ERP) [22]. ERP_{norm} denotes the ERP applied to routes A and B with normalized travel times, while ERP_{e} denotes the number of edits described by the ERP algorithm on sequence B with respect to A. If ERP prescribes 0 edits, the whole score becomes 0. The lower the score, the higher the similarity between the routes. For the final score, the average over the 3072 evaluation route instances is taken.

2-3 Technical Proceedings & Research Proposal

The already existing approaches proposed to tackle the Amazon challenge are summarized in a technical proceedings document [7]. Most methods rely on learning a certain pattern in the sequence of visiting zones. Some of them try to learn constraints based on the zone sequence and enforce them on a TSP heuristic (e.g., [23] and Article XIV of the proceedings). Other methods try to adjust the distance matrix based on the sequence of zones visited by the expert drivers (e.g., Article II, Article X, and Article VI of the proceedings). Inverse reinforcement learning was also used in order to approximate the routing policy of the expert drivers (Article IV and Article XVII).

Research proposal: In this thesis we will propose an inverse optimization strategy to learn the routing behavior of experienced drivers. In particular, given a graph and TSP tours over a subset of its nodes, we develop an inverse optimization approach that learns the edge weights that expert drivers are assumed to use while computing their node sequences. Our approach is based on the concept of minimizing inverse optimization loss functions [24], and we propose an efficient method to optimize the loss. Our algorithm is specially tailored to learning the weights of a graph (i.e., nonnegative cost functions). We aim to introduce a systematic and modular approach that is applicable to the Amazon Challenge but is also easily convertible to tackle other delivery route planning problems.

Chapter 3

Preliminaries

Before proposing a methodology to tackle the Amazon challenge we will introduce two main concepts of our approach, namely, inverse optimization and the traveling salesman problem.

3-1 Inverse Optimization

Traditional mathematical optimization programs aim to find an optimal decision or action that minimizes or maximizes a certain objective function given a signal. This process can also be called the forward model [25]. Inverse optimization reverses this process; the action and signal are given, while the objective function is unknown. A parameterized approximate version of the objective function is proposed and with inverse optimization the parameters of the proposed objective function are determined. One could also note that inverse optimization is very similar to imitation learning in which a learner agent tries to find a policy by mimicking the behavior of an expert agent [26]. In contradiction to imitation learning, in which the learner agent tries to find a direct mapping between the signal and action space, inverse optimization focuses on finding the objective function which the expert agent minimizes to find its policy.

Within the literature, a distinction exists between classical inverse optimization and data-driven inverse optimization. The difference between both methods is the notion of inverse feasibility [25]. In the case of classical inverse optimization, it is assumed that a perfect model-data fit exists. In other words, the objective function found by the inverse optimization program and the original objective function are optimal for the same actions. The focus in early literature was mainly on classical inverse optimization and how to reformulate the problem to make it solvable. However, more interest in the data-driven inverse optimization approach has arisen recently. In case of data-driven inverse optimization, there is no perfect model-data fit. The learner agent has imperfect information, resulting from one of the following reasons; the real objective function can not be found since it is not in the space of the set of hypothesis functions, the data is noisy or due to bounded rationality the expert agent finds a suboptimal solution [24]. To measure the mismatch between the model and data, loss functions are used. Those loss functions are the core of this approach and play an important role in the optimization of the parameters. In this thesis, we will focus on the data-driven inverse optimization approach.

3-1-1 Mathematical Formulation

For the mathematical notation of inverse optimization, we follow Akthar et al. [27]. The expert agent solves a forward optimization problem with the objective function F^* in which a certain signal $s \in \mathbb{S}$ is given and finds an optimal action $x^{\text{et}}(s)$ out of all possible actions $x \in \mathbb{X}(s)$ which minimizes the objective function F^*. $\mathbb{S}$ denotes the set of all admissible signals. The mathematical formulation of the statements above is given by

$$x^{\text{et}}(s) \in \arg\min_{x \in \mathbb{X}(s)} \quad F^*(s, x). \tag{3-1}$$

Next, in the inverse optimization program, the learner agent wants to find F_θ, the approximate of the objective function F^*. Therefore, a set of possible parameterized objective functions is proposed. This set of functions will be called the hypothesis space $\mathcal{F}$ and is given by

$$\mathcal{F} := \{F_\theta \ : \ \mathbb{S} \times \mathbb{X} \to \mathbb{R}, \ \ \theta \in \Theta\}, \tag{3-2}$$

in which $\theta \in \Theta$ define the parameters that have to be learned and $\mathbb{X} \supset \mathbb{X}(s)$ denotes the set of all admissible actions for all admissible signals $s \in \mathbb{S}$.

The learner agent can solve a forward optimization problem as well in which the optimal action

$$x^{\text{ln}}(s) \in \arg\min_{x \in \mathbb{X}(s)} \quad F_\theta(s, x) \tag{3-3}$$

can be found by minimizing the parameterized objective function F_θ while keeping the known constraints imposed on the action $(x(s) \in \mathbb{X}(s))$ in mind.

Since we deal with data-driven inverse optimization, we assume there is no perfect model-data fit resulting in a difference between the optimal action calculated by the expert agent and the optimal action calculated by the learner agent. To measure this difference, a loss function $\ell_\theta(x^{\text{et}}(s), x^{\text{ln}}(s))$ is used. This loss function is minimized with respect to θ to find the most suitable approximate of the objective function. If the learner agent has access to a data set of T instances of signals with corresponding expert actions $\{(\widehat{s}_t, \widehat{x}_t^{\text{et}})\}_{t<T}$, the optimization program

$$\min_{\theta \in \Theta} \sum_{t=1}^{T} \ell_\theta \left(\widehat{x}_t^{\text{et}}, x_\theta^{\text{ln}}(\widehat{s}_t) \right) \tag{3-4}$$

can be used to find the parameters θ.

3-1-2 Loss Functions

As stated before, the loss function is a measure to determine the mismatch between the objective function F_θ found by the learner agent and the real objective function F^* used by the expert agent. The most logical way would be to measure this by the *identifiability loss*

$$\ell_\theta^{\mathrm{i}}(s,x) := |F^*(s,x) - F_\theta(s,x)|^2, \tag{3-5}$$

since this loss function describes the mismatch ideally [24]. However, this loss function is not useful since the objective of inverse optimization is to learn the unknown objective function F^* of the expert agent. Since F^* is unknown, the identifiability loss function can not be used.

Another reasonable approach could be to use the squared Euclidean distance from the expert action x^{et} to learner action x^{ln} for the same signal s [28]. This is also called the *predictability loss* and the mathematical formulation is given by

$$\ell_\theta^{\mathrm{p}}(s, x^{\mathrm{et}}(s), x^{\mathrm{ln}}(s)) : = ||x^{\mathrm{et}}(s) - x^{\mathrm{ln}}(s)||_2^2. \tag{3-6}$$

The predictability loss function does capture the objective of inverse optimization very well, similar to the identifiability loss. However, Aswani et al. [28] show that the optimization turns into an NP-hard non-convex bilevel optimization problem. Therefore another loss function has to be introduced which is computationally more attractive, which is the *suboptimality loss*

$$\begin{aligned}\ell_\theta^{\mathrm{s}}(s, x^{\mathrm{et}}, x^{\mathrm{ln}}) : &= F_\theta(s, x^{\mathrm{et}}) - \min_{x \in \mathbb{X}(s)} F_\theta(s,x) \\ &= F_\theta(s, x^{\mathrm{et}}) - F_\theta(s, x^{\mathrm{ln}}).\end{aligned} \tag{3-7}$$

The suboptimality loss calculates the difference between the parameterized cost of the learner action and the expert action [24]. When the suboptimality loss is minimized, the idea behind the minimization is that it searches for an objective function F_θ which best explains the expert action x^{et} for a certain input s [27]. The suboptimality loss goes to zero only if the expert action x^{et} is the optimizer of the parameterized objective function F_θ. Figure 3-1 gives a graphical interpretation of the suboptimality and predictability loss.

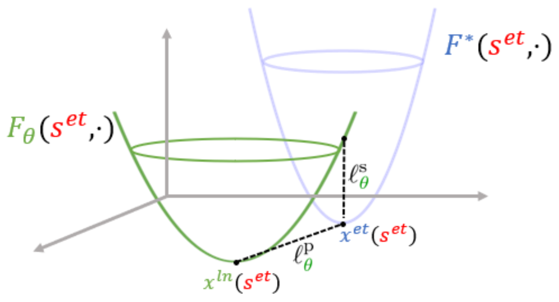

$F^*(s^{et}, \cdot)$
$F_\theta(s^{et}, \cdot)$
ℓ_θ^{s}
$x^{et}(s^{et})$
ℓ_θ^{p}
$x^{ln}(s^{et})$

Figure 3-1: Schematic interpretation of the predictability loss ℓ_θ^p and the suboptimality loss ℓ_θ^s.

The suboptimality loss is computationally attractive because the loss function is convex in θ when the hypothesis function F_θ is linear with respect to θ. As mentioned before, this is not the case for the predictability loss function. The convexity of the suboptimality loss function can be shown by rewriting the loss function from Equation 3-7 into the maximization problem [29]

$$\ell_\theta^{\mathrm{s}}(s, x^{\mathrm{et}}) = \max_{x \in \mathbb{X}(s)} F_\theta(s, x^{\mathrm{et}}) - F_\theta(s, x). \tag{3-8}$$

In this thesis, $\mathbb{X}(s)$ will be a finite integer set, which converts Equation 3-8 into a maximization over a finite set of convex functions, which preserves convexity [30]. Since F_θ will be chosen to be linear with respect to θ, the suboptimality loss will be a piece-wise linear function.

3-1-3 Hypothesis Functions

The parameterized objective function has to be chosen from the earlier defined hypothesis space $\mathcal{F}$. How to define the hypothesis space, depends on the forward expert optimization program. In general, on the one hand, the space $\mathcal{F}$ should be rich enough to capture the behavior of the expert agent's objective function F^*. On the other hand, it should not be too complex to ensure tractability of the inverse optimization program [24]. As mentioned in the previous section, a good choice for a class of hypothesis functions would be to make sure that F_θ is linear with respect to θ, which results in a convex inverse optimization program, when the suboptimality loss function is used.

A possible hypothesis class that incorporates the objectives mentioned above is the set of hypothesis functions that are linear with respect to the action x, given by

$$\mathcal{F}_l := \left\{ F_\theta(s, x) = \langle \theta, x \rangle \;\middle|\; \theta \in \Theta \right\}. \tag{3-9}$$

The choice of the set of admissible parameters Θ again depends on the problem but should definitely not contain $\theta = 0$ as this would result in a trivial constant objective function. Esfahani et al. [24] discuss some examples of how to choose Θ.

3-2 Traveling Salesman Problem

As mentioned earlier, a traveling salesman problem (TSP) is a combinatorial optimization program in which an agent has to visit a set of nodes once and once only leaving from and finishing at a predefined stop (depot). Each of the routes in the Amazon data set can be thought of as a distinct TSP. In this section, we will introduce the mathematical formulation of the TSP and some extensions.

3-2-1 Mathematical Formulation

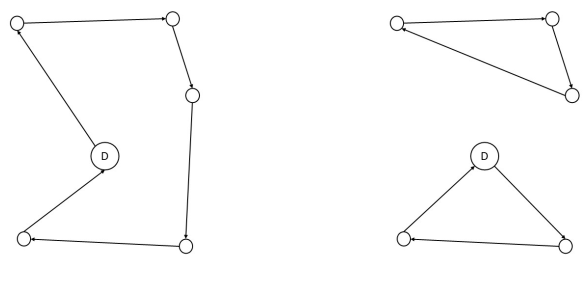
For the mathematical formulation of the basic TSP, we follow the ILP formulation proposed by Dantzig et al. [31]. First of all, let us define that the number of the depot is 1 and the number of the other nodes are $i = 2, 3, \dots, n$. All nodes including the depot are contained in the set $S = \{1, 2, 3, \dots, n\}$. Secondly, we denote the following:

$$x_{ij} := \begin{cases} 1, & \text{if node } j \text{ is directly visited after node } i \\ 0, & \text{otherwise.} \end{cases} \tag{3-10}$$

Lastly, we denote a cost matrix C which on every entry c_{ij} stores the cost to travel from node i to j. The basic TSP can then be mathematically formulated as

$$\begin{aligned} \min \quad & \sum_{i \in S} \sum_{j \in S} c_{ij} x_{ij} && \\ \text{s.t.} \quad & \sum_{j \in S} x_{ij} = 1 && 0 \le i \le n \\ & \sum_{i \in S} x_{ij} = 1 && 0 \le j \le n \\ & x_{ij} \in \{0, 1\} && 0 \le j \le n,\ 1 \le i \le n \\ & \sum_{i \in V} \sum_{j \in V} x_{ij} \le |V| - 1 && \forall V \subset S,\ V \neq \emptyset,\ \bar{V} \neq \emptyset, \end{aligned} \tag{3-11}$$

in which the optimization program minimizes the total cost of visiting all nodes once and once only. The first two constraints imposed on the optimization make sure that every node is visited only once. The third constraint imposes that the action space only consists of binary variables. Finally, the last constraint prohibits subtours; tours visiting less than n nodes (an example is given in Figure 3-2). V is any nonempty subset of S and $\bar{V}$ is the complement of V; a set that contains all nodes in S which are not in V. The constraint is very intuitive since if there would exist a subtour containing a subset of nodes $V \in S$, the left-hand side of the equation would be equal to $|V|$. This would violate the constraint of being smaller than or equal to $|V| - 1$ [4].



(a) Without subtours. **(b)** With subtours.

Figure 3-2: Example solutions to a TSP.

3-2-2 Asymmetrical TSP to Symmetrical TSP

In this and the following section, we will introduce two adjustments with respect to the basic TSP which will be used in our method. First of all, for some TSP solvers, we need to make sure that the cost matrix is symmetrical. This will not always be the case and thus we have to make a transformation of the cost matrix from asymmetrical to symmetrical. We will do this following the approach proposed by Jonker et al. [32]. In this approach for each node i, a dummy node d_i is added which is visited directly after the corresponding node is visited. The transformation works as follows: we define a weight matrix $\bar{C}$ which is equal to C except for the elements $\bar{c}_{ii}$. Those elements are set to $\bar{c}_{ii} = -M$ in which M is a sufficiently large number. Furthermore, we define the matrix U in which $u_{ij} = \infty \; \forall i, j \in S$. The new symmetrical TSP problem has the symmetrical weight matrix

$$\bar{C} = \begin{bmatrix} U & \bar{C}' \\ \bar{C} & U \end{bmatrix}.$$

If we would take an example in which the 3×3 cost matrix

$$C = \begin{array}{c|ccc|} & 1 & 2 & 3 \\ 1 & c_{11} & c_{12} & c_{13} \\ 2 & c_{21} & c_{22} & c_{23} \\ 3 & c_{31} & c_{32} & c_{33} \end{array} \tag{3-12}$$

would be transformed into

$$C = \begin{array}{c|cccccc|} & d_1 & d_2 & d_3 & 1 & 2 & 3 \\ d_1 & \infty & \infty & \infty & -M & c_{21} & c_{31} \\ d_2 & \infty & \infty & \infty & c_{12} & -M & c_{32} \\ d_3 & \infty & \infty & \infty & c_{13} & c_{23} & -M \\ 1 & -M & c_{12} & c_{13} & \infty & \infty & \infty \\ 2 & c_{21} & -M & c_{23} & \infty & \infty & \infty \\ 3 & c_{31} & c_{32} & -M & \infty & \infty & \infty \end{array}. \tag{3-13}$$

The transformation makes sure that a dummy node is visited directly after its corresponding node. Furthermore, it is impossible to visit a dummy node after a dummy node and impossible to visit a regular node after a regular node. A solution for example could be $1 \to d_1 \to 2 \to d_2 \to 3 \to d_3 \to 1$. This can easily be converted to the solution of the original asymmetric TSP by removing the dummy nodes.

3-2-3 Open TSP

In a basic TSP formulation, the agent has to finish its route at the depot it started from and thus a cycle is driven. In some cases, however, we would like to solve an open-loop TSP in which the agent does not have to come back to the depot. This can easily be achieved by setting the costs from all nodes to the depot equal to zero $c_{i1} = 0 \quad \forall i \in S$. We will refer to this as *Open TSP*.

It might be the case that we do not know beforehand at which stop we want to start an open loop TSP. In this case, there is no predefined depot but we want to optimize the route such that the location of the starting stop is part of the minimization of the costs. This can be achieved by adding a dummy node d to the cost matrix which will function as the depot. The costs from and to the dummy node will both be zero

$$\begin{aligned} c_{id} &= 0 \quad \forall i \in S \\ c_{dj} &= 0 \quad \forall j \in S \end{aligned} \tag{3-14}$$

resulting in an open loop TSP with an optimized starting location. From now on we will refer to this as *Open TSP with Unknown Depot.*

Chapter 4

Methodology

To tackle the Amazon challenge we will propose an approach in which we use inverse optimization to learn the weights between zones using historical routing data, after which we propose an algorithm to transform the zone sequence into a stop sequence. In this chapter, we will give a general overview of our method and discuss all sub-parts of the method in more detail.

4-1 Overview

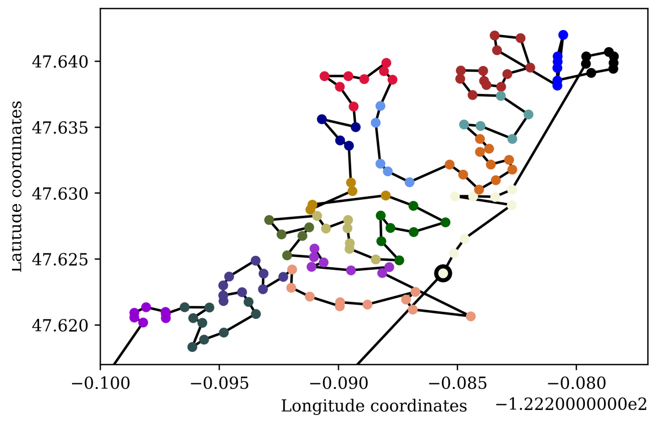


Figure 4-1: Expert route in which all stops are depicted by dots and every zone is represented by a color.

From analyzing the data of the Amazon challenge, it becomes clear that the data is not rich enough to learn patterns at stop level because in general every stop is just visited once throughout the complete data set. However, from the data analysis, it becomes clear that there is a pattern in the experienced driver routes. In most cases, drivers visit all stops in a zone before entering another zone. Figure 4-1 shows this pattern.[1] In the figure every color defines the stops belonging to a certain zone. The first stop in the sequence is indicated with a black circle around it. Clearly, the expert driver visits all stops in a certain zone before entering another zone. Since those zones, in contrast to stops, are visited multiple times throughout the data set, we will propose a method using inverse optimization to derive a policy that imitates the zone sequencing of experienced drivers. Subsequently, we propose a method to derive a stop sequence based on the zone sequences.

Our method consists of a learning phase and an evaluation phase. In the learning phase, the expert stop sequence is transformed into a zone sequence, after which we use inverse optimization to find weights that represent the preference of the driver to travel between a pair of zones. In the evaluation phase, a zone sequence can be estimated on an unseen data set using the weights found in the training phase. Lastly, we have developed an algorithm that computes the sequence of stops based on the zone sequence. Figure 4-2 gives an overview of the proposed method. In the following sections, we will explain the subparts of the method in more detail.

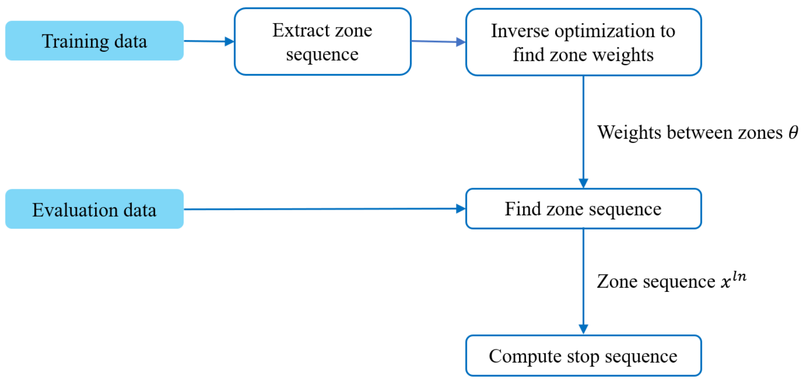


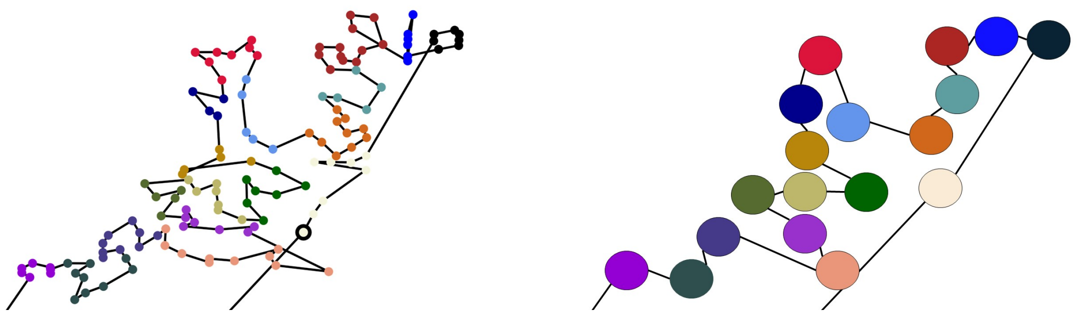
Figure 4-2: Overview of the proposed method.

4-2 Extract Expert Zone Sequence

As mentioned in the previous section, our goal is to learn the preferences of drivers at zone level. Therefore, we have to transform the expert stop sequences into expert zone sequences. Suppose we have a route as shown in Figure 4-1, we can transform this stop sequence into a zone sequence by replacing the cluster of stops with one node representing the zone, as shown in Figure 4-3.

In this example, all stops in a zone are visited before entering another zone, which is the case for most routes. However, in certain routes, it might be the case that a zone is visited, left and visited again by an experienced driver. In that case, we consider that a zone is visited at the time the most consecutive stops in that zone are visited. To illustrate this, consider the case when a driver visits 7 stops belonging to zones A, B, C, where the sequence of visited stops, in terms of zones, is $A \rightarrow B \rightarrow B \rightarrow A \rightarrow A \rightarrow C \rightarrow C$. In this case, we will consider the sequence of zones to be $B \rightarrow A \rightarrow C$.

[1]The depot is very far away from all other stops and thus cut off the figure.



(a) Expert stop sequence. **(b)** Expert zone sequence.

Figure 4-3: Example route showing the expert stop sequence and the corresponding zone sequence.

4-3 Inverse Optimization to Find Zone Weights

After a zone sequence is extracted from the data, we want to learn the zone sequencing behavior of the experienced driver. We therefore apply inverse optimization on the training data to receive a weights matrix θ between the different zones, which scales the weight according to the preference of the experienced drivers to use that certain edge. Since every zone can only be visited from one particular depot, we will consider every depot separately, and thus construct a weight matrix per depot. All routes leaving from that particular depot will form a learning data set for that certain depot's weight matrix. In this section, we will discuss the inverse optimization approach more thoroughly.

4-3-1 Inverse Optimization Applied to TSP Formulation

Formally, let $\mathcal{G}$ be a complete directed graph with m nodes and, consequently, m^2 directed edges, in which m defines the number of zones that can be visited from a certain depot. For every route instance in the training data set, only a subset of zones has to be visited. We thus define the signal $s \in \mathbb{R}^m$, where the i'th component of s equals 1 if zone i has to be visited, and 0 otherwise. The same information is encoded by the set S, where $i \in S$ if zone i has to be visited, and $i \notin S$ otherwise.

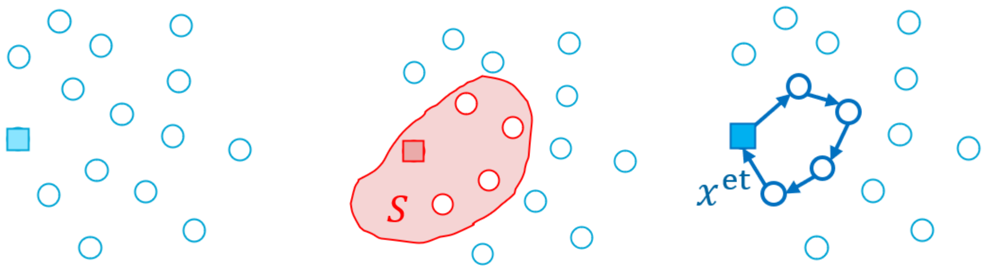
We assume that the experienced drivers are the expert agents and that they are solving a minimization

$$x^{\text{et}}(s) = \arg \min_{x \in \mathbb{X}(s)} F^*(s,x) \tag{4-1}$$

to determine the zone sequence $x^{\text{et}}(s)$. We assume that the objective function they are minimizing is equal to the TSP formulation and we thus formulate the hypothesis function as

$$F_\theta(s,x) = \sum_{i=1}^{m} \sum_{j=1}^{m} \theta_{ij} x_{ij} = \langle \theta, x \rangle. \tag{4-2}$$

The formulation is similar to the TSP formulation except for the fact that the weights are not based on time or distance. In this case, the weight θ_{ij} depends on the preference of the driver to use the edge connecting zone i to zone j and is the unknown parameter we aim to retrieve through inverse optimization. The parameter space $\Theta := \{\theta \geq 0\}$ imposes all weight vectors to be nonnegative. Similarly to the TSP formulation, x_{ij} is 1 if zone j is visited directly after zone i and 0 otherwise. A graphical interpretation depicting the nodes of the complete directed graph $\mathcal{G}$, the signal s and the action x^{et} is given in Figure 4-4. In this figure, the square depicts the depot and all other nodes are presented as circles.



(a) Nodes Graph $\mathcal{G}$. **(b)** Signal s. **(c)** Action x^{et}.

Figure 4-4: Graphical interpretation of the nodes of the graph $\mathcal{G}$, the signal s and the action x^{et}.

The constraint set $\mathbb{X}$

$$\mathbb{X}(s) = \left\{ x \in \mathbb{R}^{m^2} \;\middle|\; \begin{array}{ll} \sum_{j \in S} x_{ij} = 1, & \forall i \in S \\ \sum_{i \in S} x_{ij} = 1, & \forall j \in S \\ x_{ij} \in \{0,1\} & \forall i \in S, \forall j \in S \\ \sum_{i \in V} \sum_{j \in V} x_{ij} \leq |V| - 1, & \forall V \subset S, V \neq \emptyset, \bar{V} \neq \emptyset \\ x_{ij} = 0 & \forall i \notin S \\ x_{ij} = 0 & \forall j \notin S \end{array} \right\}, \tag{4-3}$$

is very similar to the constraint set of the formulation of the TSP and thus makes sure that routes are computed in which the driver visits every node once and once only and that no subtours occur. The last two constraints are added compared to the TSP constraint set. Those two constraints require to compute a tour over a subset of nodes S, instead of visiting all zones in the directed graph $\mathcal{G}$. $x \in \mathbb{R}^{m^2}$ denotes the vector containing all x_{ij}. Notice that x also contains information about edges from a node to the same node itself x_{ii}. The subtour elimination constraint, however, imposes that those edges always have a value of zero.

The suboptimality loss will be used to determine the mismatch between the learner action and expert action because of its computational benefits. Expressing the suboptimality loss with the hypothesis function results in

$$\begin{aligned}
\ell^{\mathrm{s}}_\theta &= F_\theta(s, x^{\mathrm{et}}) - \min_{x \in \mathbb{X}(s)} F_\theta(s, x) \\
\ell^{\mathrm{s}}_\theta &= \langle \theta, x^{\mathrm{et}} \rangle - \min_{x \in \mathbb{X}(s)} \langle \theta, x \rangle \\
\ell^{\mathrm{s}}_\theta &= \langle \theta, x^{\mathrm{et}} - x^{\mathrm{ln}} \rangle.
\end{aligned} \tag{4-4}$$

The provided training dataset $\{(\widehat{s}_t, \widehat{x}^{\mathrm{et}}_t)\}_{t<T}$ gives us a set of zones that has to be visited s and the corresponding expert zone sequence x^{et} for T route instances. The inverse optimization program which we want to solve is formulated as

$$\begin{aligned}
\min_{\theta \in \Theta} \sum_{t=1}^{T} \ell^{\mathrm{s}}_\theta(\hat{s}_t, \hat{x}^{\mathrm{et}}_t, x) &= \min_{\theta \in \Theta} \sum_{t=1}^{T} \left\{ \langle \theta, x^{\mathrm{et}}_t \rangle - \min_{x \in \mathbb{X}(\hat{s}_t)} \langle \theta, x \rangle \right\} \\
&= \min_{\theta \in \Theta} \sum_{t=1}^{T} \left\{ \langle \theta, x^{\mathrm{et}}_t - x^{\mathrm{ln}}_t \rangle \right\} \\
&:= \min_{\theta \in \Theta} \ell(\theta),
\end{aligned} \tag{4-5}$$

in which we want to minimize the loss for the set of route instances by optimizing the parameterized weights θ.

4-3-2 Solution Method

To solve Equation 4-5, we propose a subgradient descent method with a two-fold update step, containing an exact solution for the inner minimization and a subgradient descent update step for the outer minimization. In this section, this will be discussed in more detail.

The subgradient descent approach will update the θ values according to

$$\theta^{+} = \Pi(\theta - \eta \nabla \ell(\theta)), \tag{4-6}$$

in which a subgradient of the loss [33] is defined as

$$\nabla \ell(\theta) = \sum_{t=1}^{T} \left\{ x^{\mathrm{et}}_t - x^{\mathrm{ln}}_t \right\}. \tag{4-7}$$

The projection

$$\Pi(\theta) = \max(\theta_{ij}, 0) \quad \forall i, j \tag{4-8}$$

imposes that all weights keep a nonnegative value. The learner action

$$x^{\mathrm{ln}} = \arg \min_{x \in \mathbb{X}(s)} \langle \theta, x \rangle \tag{4-9}$$

has to be calculated every iteration again since the θ values are updated every iteration. To this end, we use a TSP solver in which the cost matrix is defined as the updated θ matrix.[2] Any TSP solver that finds an optimal solution can be used in this case. Since we are solving the problem on a directed graph, the updates of the θ values may result in an asymmetrical cost matrix. If a TSP solver is only able to solve symmetrical TSPs, it might be necessary to transform the asymmetrical TSP into a symmetrical TSP according to the approach discussed in Section 3-2-2. Furthermore, the zone of the depot is very far away from the other zones on a route. From the data analysis, it becomes clear that the first visited zone after the depot is chosen quite arbitrarily by drivers. We therefore want to exclude the depot from the routing problem and solve an *Open TSP with Unknown Depot* with the remaining zones as discussed in Section 3-2-3.

Although the subgradient descent method may yield sufficient results, we can achieve results more efficiently if we use a stochastic version of the subgradient descent method [34]. In this case, we take a randomized sample of the complete data set $k \sim \mathrm{U}[1, T]$ and calculate the subgradient considering that specific data point only

$$\widetilde{\nabla}\ell(\theta) = x_t^{\mathrm{et}} - x_t^{\mathrm{ln}}. \tag{4-10}$$

This is done iteratively for all data points $k \in [1, T]$. The amount of computational effort of T iterations of the stochastic subgradient descent approach is comparable to the computational effort of one iteration of the regular subgradient descent approach, because the number of times the TSP solver has to calculate the learner action, which takes the highest amount of computation effort, is equal. The performance, however, is higher for the stochastic version. Algorithm 1 gives an overview of the stochastic subgradient descent algorithm.

Algorithm 1 Stochastic Subgradient Descent Algorithm

```
1: Input: Step-size sequence $\{\eta_p\}_{p=1}^{P}$, initial weight matrix $\theta_1 \geq 0$ and dataset $\{(\hat{s}_t, \hat{x}_t^{\mathrm{et}})\}_{t<T}$.
2: for $p = 1, \ldots, P$ do
3:     Sample $k$ sequentially from $\{1, \ldots, T\}$
4:     Compute the learner action $x^{\mathrm{ln}} = \arg\min_{x \in \mathbb{X}(\hat{s}_k)} \langle \theta_p, x \rangle$
5:     Compute stochastic subgradient: $\widetilde{\nabla}\ell(\theta) = \hat{x}_k^{\mathrm{et}} - x^{\mathrm{ln}}$
6:     Stochastic Subgradient step: $\tilde{\theta}_{p+1} = \theta_p - \eta_p \nabla\ell(\theta)$
7:     Projection step: $\theta_{p+1} = \Pi(\tilde{\theta}_{p+1})$
8: end for
9: Output: $\theta_{P+1}$.
```

The value for η_p is determined as $\eta_p = \eta_1/\sqrt{p}$. The initial θ matrix is determined by the geographical distance between the zone centers. For every zone, we determine the location of a zone center by taking the mean of the longitudinal and lateral coordinates of all stops that belong to that particular zone. The initial value θ_{ij} is the Euclidean distance between the zone center of zone i and the zone center of zone j. Let us repeat, that Algorithm 1 is applied to every depot separately since every zone can only be visited from one depot. We will thus construct a separate weight matrix for every depot.

[2] Notice that regular TSP solvers do normally not solve problems with the last two constraints of the constraint set given in Equation 4-3. We simply respect those two constraints by ignoring the nodes from graph $\mathcal{G}$ that are not contained in the set s. As such the problem is transformed into a regular TSP.

4-4 Evaluation Phase

From the training phase, we retrieve the θ matrix providing us with weights that should describe the preference of an experienced driver to use that edge. In the evaluation phase, we want to compute routes using those weights on unseen evaluation data set $\{(\hat{s}_t)\}_{t<T}$. In this case, the actual driven routes are not presented to the algorithm but are later on used to evaluate how similar the actual driven route and the algorithm-computed route are.

First of all, for a new route, we will find the zone sequence $x^{\ln}$ by minimizing Equation 4-9 with the θ values we have found in the learning phase. Again, we can use any TSP solver that provides an optimal solution.

After that, the zone sequence should be transformed into a stop sequence. To determine the stop sequence, we propose to solve a TSP with a modified cost matrix C, in which c_{ij} defines the cost to travel from stop i to stop j. The travel times between the different stops are provided by Amazon. The base of the cost matrix C will be those travel times. In other words, $\tilde{c}_{ij}$ is defined as the time to travel from stop i to stop j. However, we will make two modifications

$$c_{ij} := \begin{cases} \tilde{c}_{ij}, & \text{if stops } i \text{ and } j \text{ are in the same zone} \\ \tilde{c}_{ij} + M/2, & \text{if the zone of stop } j \text{ should be visited directly after the zone of stop } i \\ \tilde{c}_{ij} + M, & \text{otherwise.} \end{cases} \tag{4-11}$$

The idea behind the cost matrix modification is that we add a penalty M if two stops are not in the same zone. If the zone of stop j has to be visited directly after the zone of stop i in the received zone sequence $x^{\ln}$ the penalty is lower, namely $\frac{1}{2}M$. If we choose the penalty M sufficiently big, this cost matrix imposes that a TSP solution will automatically visit all stops in the same zone before entering the next zone. Furthermore, the sequence of zones will not be violated.

A regular TSP solver can be used to solve a TSP with cost matrix C. Again the depot will be excluded from the problem and an open TSP will be solved. In this case, we know that the first stop has to be a stop from the first zone and thus in this case we solve the *Open TSP* introduced in Section 3-2-3. The begin stop and final stop of the TSP solution will be connected to the depot. The algorithm for computing the stop sequence is given by Algorithm 2.

Algorithm 2 Compute stop sequence for the evaluation data set

```
1: Input: $\theta$, $M$ and dataset $\{(\hat{s}_t)\}_{t<T}$.
2: for $p = 1, \dots, T$ do
3:     Compute zone sequence $x^{\ln} = \arg\min_{x \in \mathbb{X}(\hat{s}_p)} \langle \theta, x \rangle$
4:     Determine cost matrix $C$ as shown in Equation 4-11
5:     Solve TSP for stops with cost matrix $C$
6:     Connect TSP solution to the depot
7: end for
8: Output: Stop sequences for dataset $\{(\hat{s}_t)\}_{t<T}$.
```

Notice that in our approach we did not take time window constraints into consideration. Some participants of the Amazon challenge (e.g. articles VI and XXI of the proceedings [7]) observed that the constraints are often trivially satisfied and if they are ignored it has a minimal impact on the score. For that reason, we decided to ignore the constraints as well.

Chapter 5

Results

In this chapter, we will validate our method and compare the score we have retrieved with the scores on the Amazon leaderboard along with some other benchmark scores. Furthermore, we will show an algorithm-proposed route compared with the actual driven route.

5-1 Implementation

The proposed approach has been implemented in a python code and numerically evaluated on an Intel Xeon Gold 5218 CPU with a 2.30GHz clock speed and 32GB of RAM. In the code, two different TSP solvers have been used. The Gurobi ILP solver has been used to solve Equation 4-9 and calculate the learner action x^{ln} [35]. This solver has been chosen since it gives an optimal solution, which is necessary to guarantee convexity of the suboptimality loss. On the other hand, to determine the sequence of stops we used a TSP heuristic solver, which gives close-to-optimal solutions. The heuristic developed by Google OR tools is used [36] to find the stop sequence as this solver is computationally faster than the Gurobi ILP solver.

In the learning phase, we removed all routes qualified as "low" from the training data set since the evaluation data set only consists of routes that are qualified as "high". The percentage of "low" routes was much lower than the percentage of "medium" and "high" routes. Therefore, after removing the "low" routes, the data set was still sufficiently large to use for training.

Algorithm 1 iterates uniformly over all data points of the data set $\{(\hat{s}_t, \hat{x}_t^{\text{et}})\}_{t<T}$. We define an epoch as T (the length of the data set) iterations in which every data point has been used once. As explained previously, we apply our inverse optimization approach separately to each depot of the dataset, and each depot has a different number of examples. Thus, since we have to combine the results from all the depots for our final score, we decided to run the algorithm with a number of iterations proportional to the size of their respective datasets, and combine the result per epoch, instead of per iteration of Algorithm 1. The number of epochs is chosen to be 50. The other parameters have been chosen to be $\eta_1 = 0.01$ and $M = 1000$, as those gave the best score for a predefined number of epochs.

5-2 Validation of Approach

To validate the proposed approach in this thesis, we, first of all, want to check whether the total suboptimality loss[1] computed on the training data set decreases along with the number of epochs of the stochastic subgradient descent algorithm and converges to the optimal value for θ. Figure 5-1 demonstrates that the loss indeed decreases and seems to converge to an optimal value. We thus conclude that the stochastic subgradient descent algorithm works as intended.

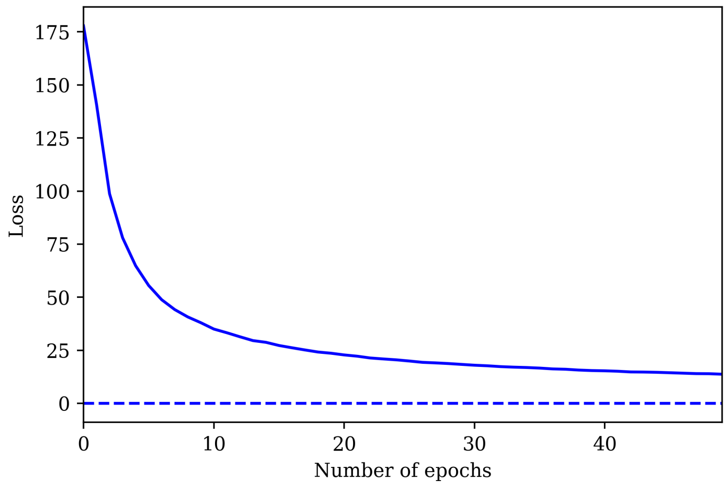


Figure 5-1: The sum of the loss per epoch.

Although the loss is minimized, we still want to check our hypothesis that by minimizing the loss, the similarity between routes proposed by the algorithm and the actual driven routes increases. To do so, after a certain number of epochs, we compute the routes with the weight matrix θ retrieved at that point and calculate the score. This is done for a predefined number of intervals. We plot the score along with the loss, to see if the score indeed decreases if the loss decreases. We also compute the score with the initial weight matrix. Figure 5-2 shows the in-sample results, that is, the results when we evaluate our method using the same dataset of 6,112 routes used for training. This result shows that optimizing the weights with the use of inverse optimization improves the similarity between the algorithm-proposed routes and the actual driven routes.

To see if our algorithm is able to capture the preferences of drivers on unseen delivery requests, we will be using the evaluation data set to compute the score. Figure 5-3 shows the out-of-sample results of our approach, that is, the results when we train our model using the 6,112 training routes but evaluate it using the testing data set of 3072 unseen routes. Again, we can see that if the weights are optimized in training, the score on the evaluation data set decreases, although naturally, not as low as the in-sample score. We can thus conclude that with the use

[1]The total loss shown in the figures is calculated slightly differently than stated in Equation 4-5 in order to decrease the computational effort. We approximate the loss by computing the loss after every iteration of the stochastic subgradient descent algorithm shown in Algorithm 1 for one data point only. The sum of the loss over all iterations in an epoch, and thus over all data points, is the one depicted in the graphs.

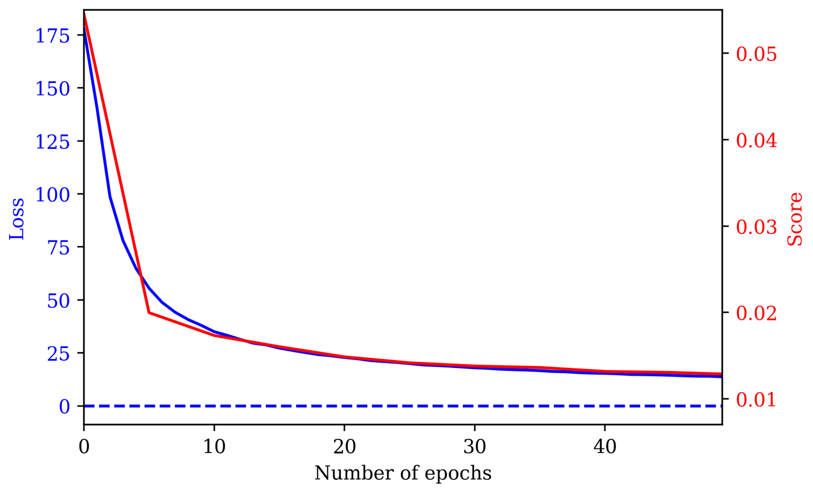


Figure 5-2: The sum of the loss per epoch and the in-sample Amazon score.

of inverse optimization we are able to increase the similarity between our algorithm-proposed routes and actual driven routes and we are thus able to capture (some of) the zone sequencing behavior of experienced drivers.

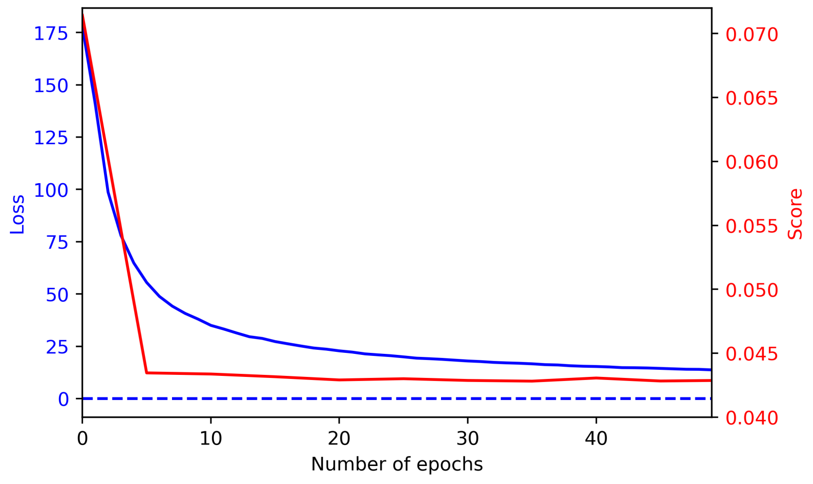


Figure 5-3: The sum of the loss per epoch and the out-of-sample Amazon score.

5-3 Amazon Score

The final score retrieved with our approach on the evaluation data set is 0.0429. To indicate whether this score is promising, we have a closer look at the scoring metric. As discussed in Section 2-2, the Amazon score indicates the similarity between an algorithm-proposed route and an actual driven route. Some benchmark scores give us an insight into the meaning of the score. First of all, for a completely random sequence of stops, the score is typically between 0.8 and 1.2. Secondly, Cook et al. [23] computed the score using a TSP solver without any extra information. The score retrieved in that case was 0.07030. Lastly, Amazon has published a leaderboard with the scores of all 48 participants of the Amazon Challenge. Our score in comparison with the top 10 contestants is shown in Figure 5-4.

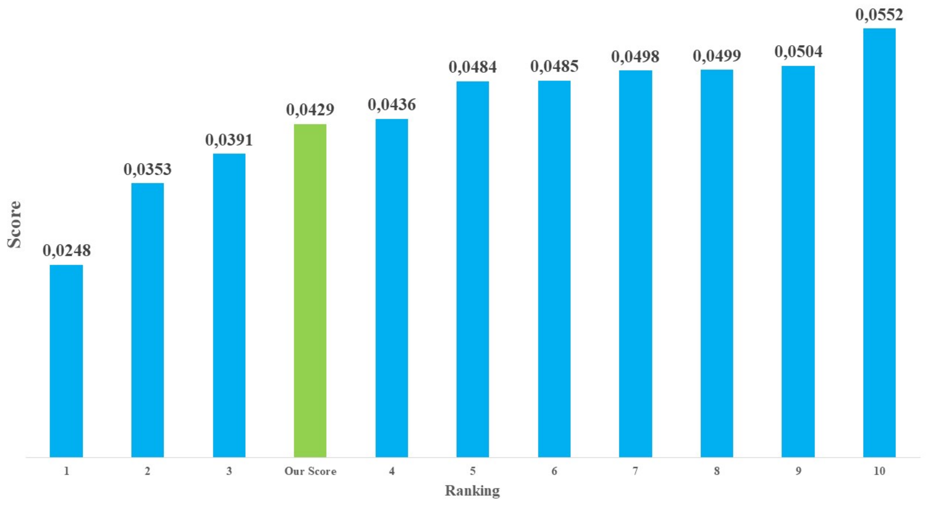


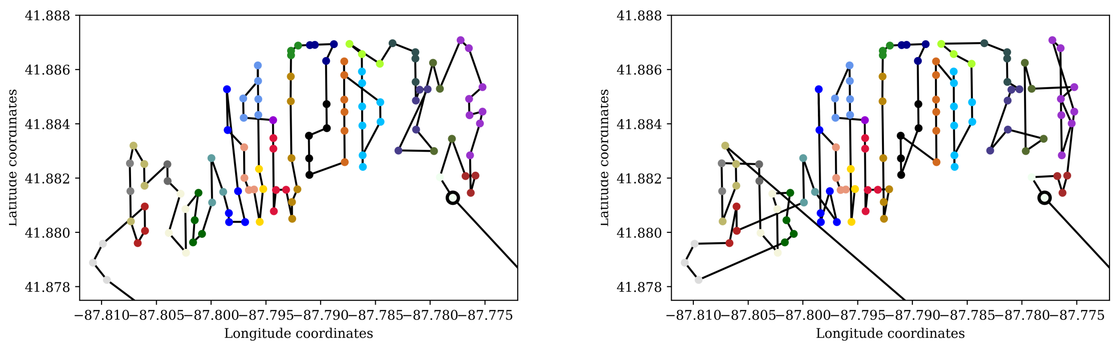
Figure 5-4: Our score compared to the Amazon Score Leaderboard of the top 10 contestants.

The first score in the leaderboard is derived by Cook et al. [23]. Their score is very impressive but our approach has some advantages compared to their method. Since their method consists of an in-depth analysis of the encoding of zones (the Zone ID), after which they derive constraints based on this encoding, their heuristic is very problem dependent. Our approach, however, is easily adaptable in two ways. First, the encoding of the zones has no influence on our approach, as such our method can also be applied to a similar problem with another encoding of the zones. If a data set is rich enough, we could even apply the zone sequencing strategy directly to stop sequencing. Secondly, if new data becomes available the weights θ could easily be updated. If there would be a constant supply of new data, the method could be transformed into an online inverse optimization learning method as proposed by Bärmann et al. [16].

The disadvantage of our implementation is that the learning time is high. This results from the structure of our method. For every iteration, an exact TSP solver has to compute the learner action x^{ln}, which approximately takes 0.5 seconds with the exact Gurobi ILP solver [35]. Since the number of iterations is high, due to the size of the data set, the learning phase might take a lot of computational effort.

5-4 Comparison of Routes

To give an interpretation of the routes proposed by our method, Figure 5-5 shows an expert route and an algorithm-proposed route from the evaluation data set.[2] From the comparison, we can remark that, in this route, the expert route does not always visit all stops in a zone before entering another zone. For the algorithm-proposed route, however, this will always be the case, since this is obliged by our approach. Considering the transformation of an expert route from a stop sequence to zone sequence discussed in Section 4-2, we can conclude that the zone sequences of both routes are almost similar.



(a) Expert route. **(b)** Algorithm proposed route.

Figure 5-5: Comparison between expert route and algorithm proposed route.

[2]The depot is very far away from all other stops and thus cut off the figure.

Chapter 6

Conclusions

In this thesis, we have shown that data-driven inverse optimization can be used to learn routing patterns from historical routing data. The zone sequences performed by experienced drivers have been assumed to be the solutions of a TSP, in which the weights represent the preference of the drivers to use a certain edge. Those weights were the unknown parameters in the inverse optimization program. Due to the way we formulated the inverse optimization program, using the suboptimality loss in combination with a hypothesis function linear with respect to θ, the convexity of the program allows us to use a stochastic subgradient descent method. Our results show that by minimizing the suboptimality loss, the similarity between the algorithm-proposed routes and the actual driven routes increases.

Regarding the Amazon Challenge, to which we applied our inverse optimization approach, we have been able to achieve a score of 0.0429, which scores 4th on the leaderboard of all 48 contestants. Furthermore, compared to the highest-scoring approach, our approach is systematic and does not depend on the encoding of the zones. This makes our approach applicable to other data sets. The inverse optimization approach, which learns the zone sequencing behavior of expert drivers, could even be applied directly at stop level, if the data is rich enough, meaning stops are visited multiple times throughout the data set. Furthermore, our approach is modular, in the sense that, any TSP solver, that gives an optimal solution, can be used in Algorithm 1. As such, if designers have preferences for certain solvers or developments occur in the field of TSP solution methods, it is easy to implement those solvers in our approach.

The inverse optimization approach proposed in this thesis is applied to a basic TSP. However, the tools described to use inverse optimization can easily be applied to other routing problems such as the VRP by adjusting the hypothesis function, shown in Equation 4-2, and the constraint set, shown in Equation 4-3, such that those are consistent with standard VRP modeling as an integer optimization problem [37]. Similar modifications could be used to handle other common scenarios, such as routing problems with time windows, backhauls, pickup and delivery, etc, which highlights the flexibility of our inverse optimization framework.

The main drawback of our implementation is its high computation time. Due to the structure of the approach, we have to solve a TSP for every iteration of the stochastic subgradient descent method in the learning phase. In our implementation, the Gurobi ILP solver takes approximately 0.1-0.5 seconds to solve a TSP instance. Due to the size of the data set, the computation time of the learning phase is high.

Even though we have already achieved promising results, there are several modifications to our approach that could lead to improvements in our final score.

- The use of an exponentiated gradient descent update step instead of a subgradient descent update step could improve the loss minimization. Preliminary results and a more detailed discussion on this topic can be found in Appendix A.
- To prevent overfitting to the training data, a generalized version of the suboptimality loss, in which we can include distance penalization, could be tested. Again, preliminary results and a more detailed discussion on this topic can be found in Appendix A.
- As explained in Section 4-1, human drivers tend to visit all stops within a zone before moving to another zone. It turns out that from the Zone ID provided by Amazon, we can define various levels of zones. That is, we have zones that comprise a set of stops; we have areas, which comprise a set of zones; and regions, which comprise a set of areas. Moreover, in the same way, drivers tend to visit all stops within a zone before moving to another zone, the same behavior is observed at the area and region levels. Thus, a promising idea is to modify our inverse optimization approach to use this assumption.
- The main drawback of our current implementation is the high computation time in the learning phase. The use of state-of-the-art TSP solvers, such as the Concorde solver [38], could decrease this.

Appendix A

Exponentiated Gradient Descent & Generalized Suboptimality Loss

In this appendix, we will introduce two concepts that could improve our method and show some preliminary results.

A-1 Generalized Suboptimality Loss

First of all, the generalized suboptimality loss function can be used instead of the suboptimality loss function, which in theory should prevent overfitting to the training data [39]. A generalized version of the suboptimality loss

$$\ell_\theta^{\mathrm{g}}(s, x^{\mathrm{et}}, x^{\mathrm{ln}}) := F_\theta(s, x^{\mathrm{et}}) - \min_{x \in \mathbb{X}(s)} \Big\{ F_\theta(s, x) - \gamma d(x^{\mathrm{et}}, x) \Big\} \tag{A-1}$$

includes a distance penalization term, where γ defines the penalization parameter and the function $d(x^{\mathrm{et}}, x)$ denotes a distance function between the expert action x^{et} and the optimizer x. If γ is chosen to be zero, the generalized suboptimality loss is again equal to the regular suboptimality loss.

Our choice for a distance function is the L1 norm

$$d(x^{\mathrm{et}}, x) = \|x - x^{\mathrm{et}}\|_1. \tag{A-2}$$

Since the action x in our problem only consists of binary variables, the distance function can be rewritten to

$$\begin{aligned}\|x - x^{\text{et}}\|_1 &= \sum_{k=1}^{m^2} (1 - x_k)x_k^{\text{et}} + (1 - x_k^{\text{et}})x_k \\ &= \sum_{k=1}^{m^2} x_k^{\text{et}} + (1 - 2x_k^{\text{et}})x_k.\end{aligned} \tag{A-3}$$

If this is plugged into the generalized suboptimality loss it will result in

$$\begin{aligned}\ell_\theta^{\text{g}} &= \langle \theta, x^{\text{et}} \rangle - \min_{x \in \mathbb{X}(s)} \left\{ \langle \theta, x \rangle - \gamma \left(\sum_{k=1}^{m^2} x_k^{\text{et}} + (1 - 2x_k^{\text{et}})x_k \right) \right\} \\ &= \langle \theta, x^{\text{et}} \rangle - \min_{x \in \mathbb{X}(s)} \left\{ \langle \theta, x \rangle - \sum_{k=1}^{m^2} \gamma x_k^{\text{et}} - \sum_{k=1}^{m^2} \gamma (1 - 2x_k^{\text{et}})x_k \right\} \\ &= \langle \theta, x^{\text{et}} \rangle + \sum_{k=1}^{m^2} \gamma x_k^{\text{et}} - \min_{x \in \mathbb{X}(s)} \left\{ \langle \theta, x \rangle - \gamma \langle \mathbf{1}_{m^2} - 2x^{\text{et}}, x \rangle \right\} \\ &= \langle (\theta + \gamma \mathbf{1}_{m^2}), x^{\text{et}} \rangle - \min_{x \in \mathbb{X}(s)} \langle (\theta - \gamma(\mathbf{1}_{m^2} - 2x^{\text{et}})), x \rangle \\ &= \langle (\theta + \gamma \mathbf{1}_{m^2}), x^{\text{et}} \rangle - \langle (\theta - \gamma(\mathbf{1}_{m^2} - 2x^{\text{et}})), x^{\text{ln}} \rangle.\end{aligned} \tag{A-4}$$

If a data set $\{(\hat{s}_t, \hat{x}_t^{\text{et}})\}_{t<T}$ is provided and the generalized suboptimality loss function is used, the inverse optimization program can be written as

$$\min_{\theta \in \Theta} \sum_{t=1}^{T} \ell_\theta^{\text{g}}(\hat{s}, \hat{x}^{\text{et}}, x^{\text{ln}}) = \min_{\theta \in \Theta} \sum_{t=1}^{T} \left\{ \langle (\theta + \gamma \mathbf{1}_{m^2}), x^{\text{et}} \rangle - \min_{x \in \mathbb{X}(s)} \langle (\theta - \gamma(\mathbf{1}_{m^2} - 2x^{\text{et}})), x \rangle \right\}. \tag{A-5}$$

This means we can still solve the inner minimization with an exact TSP solver, however, we have to adjust the weights with the term $\gamma(\mathbf{1}_{m^2} - 2x^{\text{et}})$. We can apply the stochastic subgradient descent algorithm in a similar fashion, since the gradient of this loss with respect to θ is still equal to the gradient for the suboptimality loss, as described in Equation 4-10. However, the learner action x^{ln} is the outcome of a different minimization program.

A-2 Exponentiated Gradient Descent

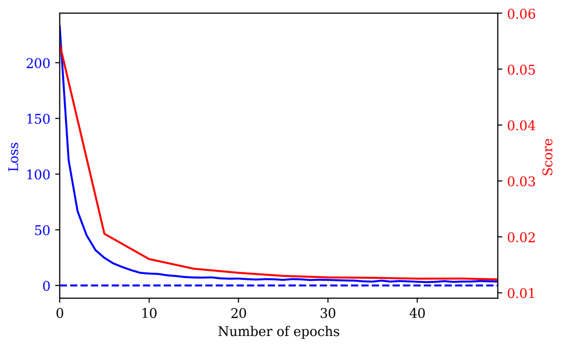
Secondly, the stochastic subgradient descent update step could be replaced by an exponentiated gradient descent update step

$$\begin{aligned}\tilde{\theta}^+ &= \theta \odot \exp\{-\eta \nabla \ell\} \\ \theta^+ &= \frac{\tilde{\theta}^+}{\|\tilde{\theta}^+\|_1},\end{aligned} \tag{A-6}$$

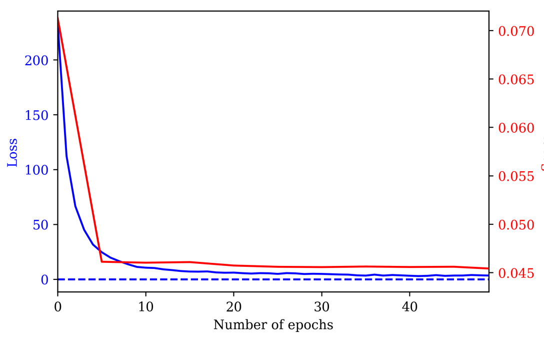
introduced by Kivinen et al. [40]. The advantage of this update step is that no projection is needed to make sure all θ_{ij} values are nonnegative since they stay positive by definition if the initial value of $\theta > 0$. This strategy is specially designed for problems in which the optimization variable is constrained to the simplex plane. As such, we define a parameter space $\Theta = \{\theta \geq 0, ||\theta||_1 = c\}$, in which c is an arbitrary constant, which guarantees that the trivial solution $\theta = 0$, will not be reachable. For the subgradient descent algorithm, we can not guarantee this. Although in practice the chance that the subgradient descent algorithm would reach the trivial solution is very small since in that case all m^2 θ values should become equal to zero, the advantage of the exponentiated version is that we can guarantee this will not happen.

A-3 Preliminary Results

Some preliminary results on the two introduced concepts have been retrieved. First, the exponentiated gradient descent method has been tested on the training data set and the evaluation data set. The results are shown in Figure A-1. We can see that the relative difference between the initial and final loss is bigger than the one we achieved using the subgradient descent approach. The final score using the training data set is also slightly lower than the one retrieved with the subgradient descent method. However, the score retrieved using the evaluation data set is slightly higher. A possible explanation could be that from a certain point, minimizing the loss for the training data set, results in overfitting to the training data set.



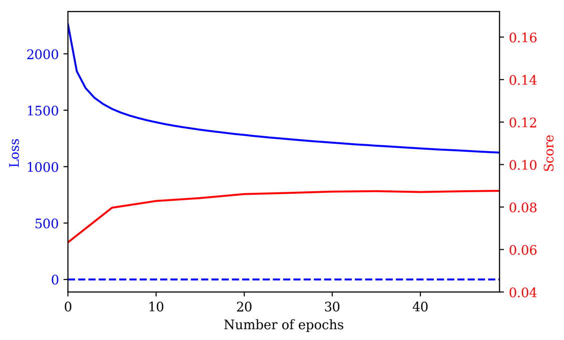
(a) In-sample results.



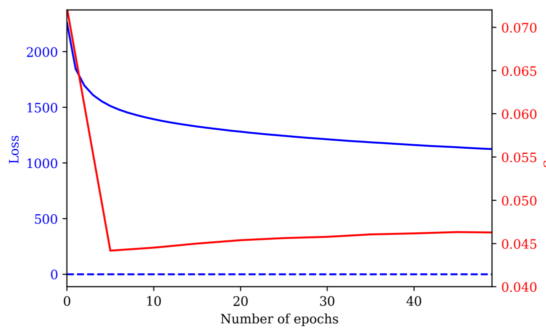
(b) Out-of-sample results.

Figure A-1: The loss and score of the exponentiated gradient descent method with $\gamma = 0$ and $\eta = 0.3$.

A possible way to prevent overfitting to the training data could be the use of the generalized suboptimality loss. In Figure A-2 the results are shown for the exponentiated gradient descent method including a distance penalty term $\gamma = 0.01$. From the results, it becomes clear, that indeed, the results using the training data set worsen while the results using the evaluation data set are not that much worse. In future research, the tuning of the distance penalty term γ could result in an improvement in results.



(a) In-sample results.



(b) Out-of-sample results.

Figure A-2: The loss and score of the exponentiated gradient descent method with $\gamma = 0.01$ and $\eta = 0.3$.

Bibliography

[1] G. B. Dantzig and J. H. Ramser, “The truck dispatching problem,” 1959.

[2] H. Zhang, H. Ge, J. Yang, and Y. Tong, “Review of vehicle routing problems: Models, classification and solving algorithms,” *Archives of Computational Methods in Engineering*, vol. 29, pp. 195–221, 1 2022.

[3] G. Laporte, “The traveling salesman problem: An overview of exact and approximate algorithms,” 1992.

[4] J.-Y. Potvin, “Genetic algorithms for the traveling salesman problem,” 1996.

[5] S. Goyal, “A survey on travelling salesman problem,” 2010.

[6] S. Sangwan and C. Dahiya, “Literature review on travelling salesman problem international journal of research literature review on travelling salesman problem,” 2018.

[7] M. Winkenbach, S. Parks, and J. Noszek, “Technical proceedings,” 2021.

[8] J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera, and A. A. Juan, “Rich vehicle routing problem: Survey,” *ACM Computing Surveys*, vol. 47, 12 2014.

[9] T. Lust and J. Teghem, “The multiobjective traveling salesman problem: A survey and a new approach,” 1970.

[10] R. Canoy and T. Guns, “Vehicle routing by learning from historical solutions,” vol. 11802 LNCS, pp. 54–70, Springer, 2019.

[11] M. Fosgerau, E. Frejinger, and A. Karlstrom, “A link based network route choice model with unrestricted choice set,” *Transportation Research Part B: Methodological*, vol. 56, pp. 70–80, 2013.

[12] S. Liu, H. Jiang, S. Chen, J. Ye, R. He, and Z. Sun, “Integrating dijkstra’s algorithm into deep inverse reinforcement learning for food delivery route planning,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 142, 10 2020.

[13] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner, "Large-scale cost function learning for path planning using deep inverse reinforcement learning," *International Journal of Robotics Research*, vol. 36, pp. 1073–1087, 9 2017.

[14] D. Burton and P. L. Toint, "On an instance of the inverse shortest paths problem," 1992.

[15] A. Faragó, Áron Szentesi, and B. Szviatovszki, "Inverse optimization in high-speed networks," 2003.

[16] A. Bärmann, S. Pokutta, and O. Schneider, "Emulating the expert: Inverse optimization through online learning," 2017.

[17] L. Chen, Y. Chen, and A. Langevin, "An inverse optimization approach for a capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 295, pp. 1087–1098, 12 2021.

[18] Y. Chung and M. Demange, "On inverse traveling salesman problems," *4OR*, vol. 10, pp. 193–209, 2012.

[19] M. Winkenbach, "Pre-competition informational webinar." https://routingchallenge.mit.edu/challenge-live-event-pre-competition-informational/, 2021. [Online; accessed 20-April-2022].

[20] D. Merchán, J. Arora, J. Pachon, K. Konduri, M. Winkenbach, S. Parks, and J. Noszek, "2021 amazon last mile routing research challenge: Data set," *Transportation Science*, 9 2022.

[21] "Amazon scoring metric." https://github.com/MIT-CAVE/rc-cli/tree/main/scoring, 2021. [Online; accessed 16-May-2022].

[22] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," 2004.

[23] W. Cook, S. Held, and K. Helsgaun, "Constrained local search for last-mile routing," 12 2021.

[24] P. M. Esfahani, S. Shafieezadeh-Abadeh, G. A. Hanasusanto, and D. Kuhn, "Data-driven inverse optimization with imperfect information," 12 2015.

[25] T. C. Y. Chan, R. Mahmood, and I. Y. Zhu, "Inverse optimization: Theory and applications," 9 2021.

[26] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Foundations and Trends in Robotics*, vol. 7, pp. 1–179, 2018.

[27] S. A. Akhtar, A. S. Kolarijani, and P. M. Esfahani, "Learning for control: An inverse optimization approach," 2021.

[28] A. Aswani, Z.-J. M. Shen, and A. Siddiq, "Inverse optimization with noisy data," 2015.

[29] M. Dankovic, "Learning parametric integer programming via inverse optimization," 2021.

[30] S. Boyd and L. Vandenberghe, *Convex optimization.* Cambridge university press, 2004.

[31] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[32] R. Jonker and T. Volgenant, "Transforming asymmetric into symmetric traveling salesman problems," 1983.

[33] D. P. Bertsekas, *Nonlinear programming.* Athena Scientific, 2008.

[34] S. Bubeck, *Convex Optimization: Algorithms and Complexity.* Foundations and Trends in Machine Learning, Now Publishers, 2015.

[35] "Traveling salesman problem, Gurobi ILP solver." https://gurobi.github.io/modeling-examples/traveling_salesman/tsp.html, 2021. [Online; accessed 28-October-2022].

[36] "Traveling salesperson problem | OR-tools | google developers." https://developers.google.com/optimization/routing/tsp, 2021. [Online; accessed 28-October-2022].

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

[38] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, *TSP Cuts Which Do Not Conform to the Template Paradigm*, pp. 261–303. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[39] P. Zattoni Scroccaro, B. Atasoy, and P. Mohajerin Esfahani, "Inverse optimization: Geometry, loss functions and algorithms," *preprint*, 2022.

[40] J. Kivinen and M. K. Warmutht, "Additive versus exponentiated gradient updates for linear prediction."