CIRRELT-2024-16

A Contextual Framework for Learning Routing Experiences in Last-mile Delivery

Huai Jun (Norina) Sun
Okan Arslan

June 2024

A Contextual Framework for Learning Routing Experiences in Last-mile Delivery

Huai Jun (Norina) Sun[1], Okan Arslan[1,2,*]

1. Department of Decision Sciences, HEC Montréal
2. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

Abstract. This paper presents a contextual framework for solving the data-driven traveling salesman problem in last-mile delivery. The objective of the framework is to generate routes similar to historic high-quality ones as classified by operational experts by considering the unstructured and complex features of the last-mile delivery operations. The framework involves learning a transition weight matrix and using it in a TSP solver to generate high quality routes. In order to learn this matrix, we use descriptive analytics to extract and select important features of the high-quality routes from the data. We present a rule-based method using such extracted features. We then introduce a factorization of the transition weight matrix by features, which reduces the dimensions of the information to be learned. In the predictive analytics stage, we develop (1) Score Guided Coordinate Search as a derivative-free optimization algorithm, and (2) label-guided methods inspired by supervised learning algorithms for learning the routing preferences from the data. Any hidden preferences that are not obtained in the descriptive analytics are captured at this stage. Our approach allows us to blend the advantages of different facets of data science in a single collaborative framework, which is effective in generating high-quality solutions for a last-mile delivery problem. We test the efficiency of the methods using a case study based on Amazon Last-Mile Routing Challenge organized in 2021. A preliminary version of our rule-based method received the third place and a $25,000 award in the challenge. In this paper, we improve the learning performance of our previous methods through predictive analytics, while ensuring that the methods are effective, interpretable and flexible. Our best performing algorithm improves the best score in the literature achieved on the available training dataset by 40.1%. It also enhances the performance of our rule-based method on an out-of-sample testing dataset by more than 23.1%.

Keywords: traveling salesman problem, learning, data-driven, derivative-free optimization, last-mile delivery

Acknowledgements. The authors gratefully acknowledge funding provided by the Natural Sciences and Engineering Research Council of Canada under grant RGPIN-2022-04979.

* Corresponding author: okan.arslan@hec.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2024

1. Introduction

Given a complete graph, the problem of finding a shortest tour that visits every node in the graph is defined as the traveling salesman problem (TSP), which is a well-known NP-hard problem (Applegate et al. 2011). In the TSP arising in last-mile delivery operations, the tour starts and ends at a designated node referred to as the depot and visits a set of customers. The intricate dynamics of the urban routing due to factors such as traffic conditions or parking availability complicate the route planning process and the goodness of a route cannot easily be expressed using a measure such as distance, time or cost, as in the TSP. The experienced drivers do not usually follow the shortest, fastest or least-cost routes, but use their tacit knowledge about the complex environment to execute efficient delivery operations. When the routes that are optimized with respect to a single measure are provided to drivers, they usually modify these routes. These modifications are often necessary in real-world operational settings, and require human intervention. The tacit knowledge that drives such modifications lack formalization. In this paper, we develop a data science framework to solve the data-driven TSP (DD-TSP). The aim is to generate tours *similar to* those that were historically executed by experienced drivers.

1.1. Literature Review

Consider a complete graph $G = (N, A)$, where N is the set of nodes and A is the set directed arcs, representing direct travel between nodes. We have $n := |N|$ and the cost of arc $(i, j) \in A$ is $d_{ij} \geq 0$. Node 0 represents the depot, and nodes $N \setminus \{0\}$ represent the customers. Let y_{ij} be a binary variable if and only if arc (i, j) is on the selected path, and 0 otherwise. The objective function of the TSP is linear in y_{ij} variables, which allows it to be modeled as a mixed integer linear program. Therefore, it is viable to obtain both lower and upper bounds on the optimal objective function value. Exact algorithms sequentially obtain such lower and upper bounds in order to reduce the optimality gap and prove optimality of a solution. The heuristics, on the other hand, obtain valid upper bounds on the objective function by finding feasible solutions. There is a multitude of both exact and approximate solution approaches for the TSP in the literature (Applegate et al. 2011). One of the best performing heuristics for the TSP is developed by Lin and Kernighan (1973) and an effective implementation is presented by Helsgaun (2000). For a review of heuristic algorithms developed for the TSP, we refer the reader to Laporte (1992). Several metaheuristics are also developed such as genetic algorithm (Potvin 1996), tabu search (Knox 1994), simulated annealing (Pepper, Golden, and Wasil 2002, Wang, Geng, and Shao 2009), variable neighborhood search (Carrabs, Cordeau, and Laporte 2007, Hore, Chatterjee, and Dewanji 2018), among others. Kotthoff et al. (2015) report that the Lin-Kernighan-Helsgaun (LKH) algorithm (Helsgaun 2000) and the genetic algorithm with a powerful edge assembly crossover operator (Nagata and Kobayashi 2013) are the two most powerful algorithms for inexact solution of the TSP.

Recent works on the TSP using machine learning are ample, and we categorize them into three classes related to our work: (1) those that generate routes directly and act as heuristics, (2) those that help improve the existing heuristics to improve their efficiency, and (3) those that are used for experience learning.

Stand-alone heuristics: Learning algorithms are used to generate feasible routes. In other words, they serve as *alternatives* to heuristics. Along this line, Bello et al. (2016) train a recurrent neural network to predict a distribution over different city permutations and generate a TSP route. Deudon et al. (2018) develop a neural network model for solving the TSP directly, and compare their results with OR-tools heuristic (Google 2022). Miki, Yamamoto, and Ebara (2018) develop a method for learning the optimal tour as an image using a convolutional neural network. Nazari et al. (2018) develops a reinforcement learning algorithm to generate routes for the capacitated VRP. Their approach outperforms classical heuristics and OR-Tools heuristics on medium-sized instances in solution quality with comparable computation time. Delarue, Anderson, and Tjandraatmadja (2020) uses deep reinforcement learning to generate routes. Their model achieves an average optimality gap of 1.7% on standard library instances of medium size. Bresson and Laurent (2021) uses transformer architecture to *learn* heuristics to potentially outperform human-developed heuristics. According to reported performances, the optimality gap is 0.39% for 100-node TSP instances. Note that, when the machine learning models are used as a stand-alone heuristic, their performance to this date is usually inferior to the performance of existing human-developed state-of-the-art heuristics.

Improving the heuristics: On the second stream of research, machine learning algorithms are used to improve the performance of the human-developed (meta)heuristics. Examples include the work by Zheng et al. (2021) that combine three reinforcement learning methods (Q-learning, Sarsa and Monte Carlo) with the LKH algorithm. Sultana et al. (2021) presents a learning-based approach for routing problems that uses a penalty term and reinforcement learning to guide the search efforts. Xin et al. (2021) improve the effectiveness of the LKH heuristics using deep learning. Hudson et al. (2021) present a framework for solving the TSP based on graph neural networks and guided local search. These aforementioned methods report performance improvements over the human-developed heuristics without machine learning support, but the improvements may be marginal. Varol, Özener, and Albey (2024) develop a TSP tour length estimator using neural networks as alternative to continuous approximation models. Their method involves an efficient method for obtaining valid lower bounds and partial solutions to the TSP. For a recent survey on machine learning algorithms designed for TSP, we refer the reader to Junior Mele, Maria Gambardella, and Montemanni (2021).

Contextual optimization and experience learning: The idea of learning input parameters of a problem to guide the classical optimization solvers is a recent idea that has been tested in various settings (Parmentier 2022) including vehicle routing (Canoy et al. 2023, 2024). Similarly, we approach the experience learning from a contextual optimization perspective, in which the available side information is leveraged to produce best empirical performance on a dataset. Sadana et al. (2024) classify the contextual optimization into three paradigms as 'decision rule optimization', 'sequential learning and optimization', and 'integrated learning and optimization'. The learning the routing preferences fall under the decision rule paradigm, in which a parameterized mapping is developed to the desired solution space.

In 2021, (Amazon 2021) organized the Amazon Last-mile Routing Challenge 2021 competition, which was based on the same idea of generating routes that are similar to those by executed by the drivers. We present further details about this competition in Section 4. The first-place holders (Cook, Held, and Helsgaun 2024) implement a penalty-based local search by considering constraints that are obtained through an analysis of historical routing data. Using the same data, the second place holders (Mo et al. 2023) develop a neural network for predicting the arc transition penalties. The third place holders (Arslan and Abay 2021) also implemented a penalty-based network transformation in which the penalties are obtained through an analysis of the data. We present this method in detail in Section 4.3. All three studies focus on learning the transition penalties (or weights). Using the same data, Özarık, da Costa, and Florio (2024) develop a pool-and-select framework, in which candidate sequences are pooled as a function of their features as seen in the dataset, and then their scores of predicted using a regression model. Scroccaro et al. (2023) propose an inverse optimization framework and develop a stochastic first-order algorithm for learning the preferences of the drivers. Their algorithm also achieves a high performance on the Amazon competition data. Ghosh et al. (2023) develop a method by first sequencing the customer zones, and then sequencing the customer stops in every zone. When compared to the aforementioned papers developing methods for solving the Amazon challenge, our best performing algorithm improves the best score reported by (Cook, Held, and Helsgaun 2024) on the Amazon *training data* by 40.1%. Even though it is generally uncommon to report training data performances, our interpretable and flexible method demonstrates the potential improvement scale that can be achieved on this data. We also improve on our previous results (Arslan and Abay 2021) by 23.1% on the Amazon *testing data* placing it the second in all the performances reported in the literature. Furthermore, our method involves a contextual framework, that can easily be adopted to any features beyond customer zones.

1.2. Contributions and Organization of the Paper

In this paper, we present a contextual framework to solve the data-driven traveling salesman problem (DD-TSP) arising in last-mile delivery operations. The objective is to generate routes that are similar to high quality ones as classified by operational experts. The framework involves learning a transition weight matrix and using a classical TSP solver to generate high quality routes. The contributions of this paper are as follows:

- We present a framework for modeling the DD-TSP, in which the aim is to generate a transition weight matrix for new problem instances.
- We introduce a factorization of the transition weight matrix by features, which reduces the dimensions of the information to be learned.
- We develop different methods for learning the transition weight matrix:

—Score Guided Coordinate Search (SGCS): a custom derivative-free optimization algorithm,

—Rule-based Feature Impact Matrix (RFIM): a rule-based method based on expert views,

—Label-guided Feature Impact Matrix (LFIM): a method for learning the transitions directly from the data, and

—Rule-based Label-guided Feature Impact Matrix (R+LFIM): combination of the two former methods.

Our methods are effective, interpretable and the flexible.

- We test the efficiency of these methods using a case study based on Amazon Last-Mile Routing Challenge. Our best performing algorithm improves the best score in the literature achieved on the available training dataset by 40.1%. We also improve on our previous methods, which achieved a third place in the Amazon challenge, by 23.1% on out-of-sample dataset.

We now present the problem and our methodological framework in the following section. Two algorithms are presented in Section 3. A case study is introduced in Section 4 and the results are presented in Section 5. Effectiveness, interpretability and flexibility of the methods are discussed in Section 6 and the study is concluded in Section 7.

2. Problem Definition and Methodological Framework

Consider a last-mile routing problem P, which aims to sequence a set of given customers, with an unknown objective function representing the preferences of the decision makers. Let s represent an instance of problem P including *features* such as distance matrix, travel times, number of parcels per customer, parcel sizes, customer zones, customer time windows, and weather information. Note that these features may be any type such as numerical, categorical, geospatial or binary. Let S be a set of instances of problem P. Consider a proxy problem $\widehat{P}$ with the same solution space as P and with a proxy objective function that *approximates* the unknown preferences of the decision makers.

These unknown preferences are generally hard to communicate and represent the tacit knowledge of the planners. A proxy problem $\widehat{P}$ is the classical TSP that minimizes one feature, which we refer to as T representing the distances or travel times between node pairs. Now, consider an optimization oracle $\eta : \mathbb{R}^{n \times n} \to \mathbb{R}$ that solves the approximate problem $\widehat{P}$ and generates an optimal solution $x_s^* := \eta(T)$ for an instance $s \in S$ using T as the only feature. Its proxy objective function leaves out the other features representing the side information. Generally, during the execution of this prescribed solution x_s^* in real-world operations, it is modified by the operational experts (by the planners before the execution or by the drivers during the execution) as $\hat{x}_s$ that reflect the unknown preferences of the planners due to the side information. The objective of this study is to learn these preferences so that the optimization oracle η produces routes similar to those that are historically executed reflecting the operational expertise. In this context, one needs to evaluate the differences between the solutions x_s^* and $\hat{x}_s$, which is achieved by a scoring function $\sigma(x_s^*, \hat{x}_s)$ that compares the two solutions x_s^* and $\hat{x}_s$, and evaluates the dissimilarity between them. Clearly, we have $\sigma(\hat{x}_s, \hat{x}_s) = 0$.

For solving problem P, the goal is to produce solutions similar to $\hat{x}_s$, which is deemed to be "superior" operationally than the solution x_s^* that is an "optimal solution" of the proxy problem $\widehat{P}$. We can measure the average *loss* of the solutions x_s^* generated by the optimization oracle with respect to the executed solutions $\hat{x}_s$ over $s \in S$ by

$$\frac{\sum_{s \in S} \sigma(x_s^*, \hat{x}_s)}{|S|}. \tag{1}$$

The problem of minimizing this error can be approached from two perspectives. One can modify the optimization oracle η so that the optimal solutions x_s^* generated by this oracle is more representative of the real-world operations. This can be done in a rule-based fashion, by discussing with operational experts and better representing their preferences in the objective function of the mathematical model. However, the planners' tacit knowledge may not always be easily articulated. In this paper, we approach this problem from a data-driven perspective, and *learn* these preferences using data. This learning is represented by transforming the instance s into matrix D^s so that the "optimal" solution generated by $\eta(D^s)$ generates solutions closer to historically preferred routes. Note that the side information available in instance s may be used to improve the sequencing decisions, and it is embedded into matrix D^s, which no longer represents the actual distances, but rather the *transition weight between node pairs*. In other words, the matrix D^s for $s \in S$ represents the experience and the preferences of the drivers. Each arc weight d_{ij} in D^s represents the penalty of including arc (i, j) in the solution. By modifying the weight, one can encourage or discourage this transition to apprear in the solution. For this reason, D^s is referred to as the *transition weight matrix*. The nomenclature is presented in Table 1.

Table 1 Nomenclature

Symbol	Definition
S	Instance set
D^s	Transition weight matrix for instance $s \in S$
η	Optimization oracle
σ	Scoring function
$\mathcal{F}$	Feature space
F_f	Feature impact matrix for feature $f \in \mathcal{F}$
γ^f	Weighted scoring matrix for feature $f \in \mathcal{F}$
I_f^s	Incidence matrix of nodes in instance $s \in S$ to feature $f \in \mathcal{F}$

2.1. Problem Definition

We now define the problem considered in this paper.

Definition 1. Given a set of instances $s \in S$ and executed routes $\hat{x}_s$ for $s \in S$, the data-driven traveling salesman problem (DD-TSP) is defined as finding the transition weight matrix D^s for $s \in S$, such that the average dissimilarity between the executed solutions $\hat{x}_s$, and the solutions generated by the optimization oracle $\eta(D^s)$ over the set of instances S is minimized.

DD-TSP can be expressed in compact form as follows:

$$\min_{D} \frac{\sum_{s \in S} \sigma(\eta(D^s), \hat{x}_s)}{|S|}. \tag{2}$$

Note that there are two nested optimization models in (2). While the outer optimization model minimizes the loss with respect to a scoring function $\sigma(\cdot)$, the inner optimization oracle $\eta(\cdot)$ solves a TSP using D^s, which is obtained by transforming the input data. The scoring function $\sigma(\cdot)$ has no particular structure, it can potentially be nonlinear or a nonsmooth function. The transition weight matrix is expected to incorporate the tacit knowledge by encoding the side information.

2.2. Factorization of the Transition Weight Matrix

Observe that even for a small problem instance with 100 nodes, model (2) requires learning 100^2 elements of the transition weight matrix D^s for a given instance $s \in S$. Furthermore, the set of nodes varies for every instance. Considering particularly the unstructured scoring function $\sigma(\cdot)$, learning the transition likelihood between every node pair may be computationally impractical. Therefore, inspired by the singular value decomposition and principal component analysis (Wall, Rechtsteiner, and Rocha 2003), we consider constructing the unknown full-dimensional D^s matrix by factoring out the features and investigating their impacts independently. Let $\mathcal{F}$ be the feature space, I_f^s be the incidence matrix of nodes in instance $s \in S$ to feature $f \in \mathcal{F}$, and F_f be the *feature impact matrix*, which describes the impact of feature f to the transition between nodes. Matrix F_f is square, and it measures the impacts of the transition between node pairs with particular features.

For $s \in S$, we include the side information using the following factorization of the transition weight matrix

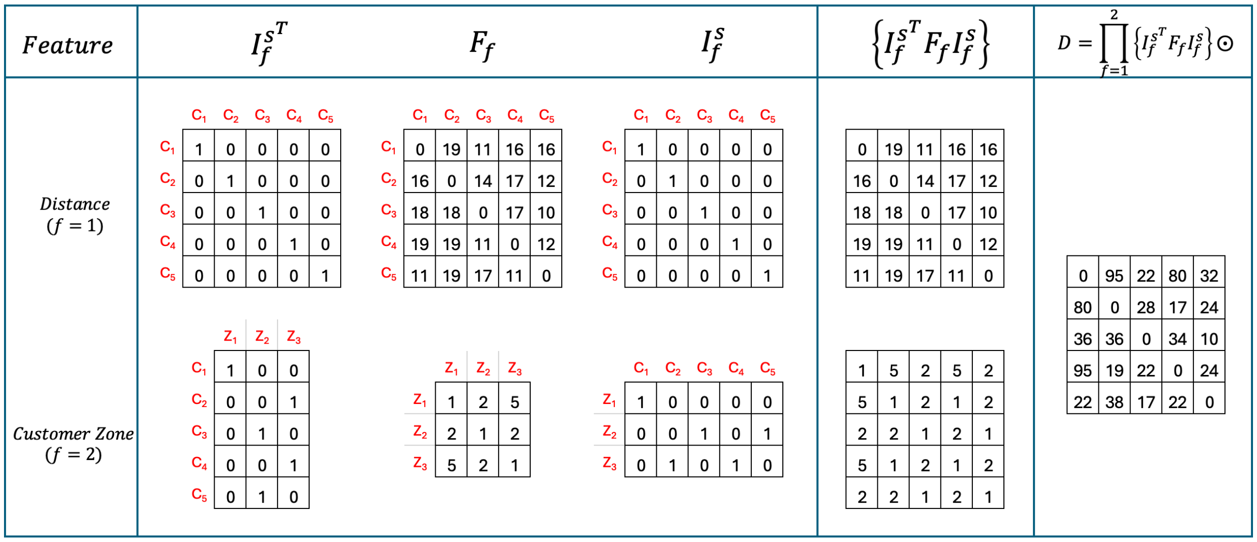
$$D^s = \prod_{f \in \mathcal{F}} \{I_f^{sT} F_f I_f^s\} \odot, \tag{3}$$

where $\odot$ represents the element-wise product of matrices. In other words, the matrix D^s is the element-wise product of the matrices $\{I_{f^T}^s F_f I_f^s\}_{n \times n}$ for $f \in \mathcal{F}$. Note that, by pre- and post-multiplying the feature impact matrix F_f by the instance matrix I_f^s, we implement a dimensionality expansion and obtain a weight per transition between node pairs corresponding to feature levels. Therefore, the problem is expressed as

$$\arg\min \frac{\sum_{s \in S} \sigma(\eta(\prod_{f \in \mathcal{F}} \{I_f^{sT} F_f I_f^s\} \odot, \hat{x}_s)}{|S|}. \tag{4}$$

In this model, the variables are the elements of the matrices F_f for $f \in \mathcal{F}$. Observe that, by this factorization, we benefit from dimensionality reduction similar to the principle component analysis. A problem instance s with five customer nodes $(C_1, \ldots, C_5)$ and two features is given in Figure 1. The first feature is the distance, and the second feature is the customer zone. The incidence matrix of the first feature is an identity matrix because distance between every node pair is unique. The second feature has three *levels* (or categories) as Z_1, Z_2 and Z_3 (emphasized in red color in the figure). The incidence matrix for the second feature represents the zone (i.e., the feature level) that customers reside in. In this example, the second feature impact matrix F_2 encourages delivering parcels within the same zone, and discourages transitions between zones. The transitions between Z_1 and Z_3 are the most penalized among other possible transitions. The matrix in the rightmost column of the table is the D^s matrix and it is obtained by the element-wise product of the two matrices in the fifth column. When this D^s matrix is used as input to the TSP, the optimization oracle produces a feasible solution, which is then evaluated in the scoring function (by comparing to the executed route in the real world). The objective in this paper is to *learn* the feature impact matrices F_f for $f \in \mathcal{F}$ corresponding to the side information (F_2 in the example in Figure 1) so that solving the TSP using the resulting D^s matrix yields "superior" solutions in practice that better represent the preferences of the decision makers.

Note that these features may be independent or dependent. The factorization may capture the independent relationships, and the feature matrices can be learned sequentially. On the other hand, the dependent information among factors may be lost in such a factorization. Nevertheless, learning capacity of algorithms such intricate relationships in large datasets is limited. Therefore, in this paper, we adapted factorization of the function in order to foster interpretability and flexibility of the methods. Similar to model building and variable selection strategies such as backward and forward methods in linear regression (Montgomery, Peck, and Vining 2021), our framework assumes learning features in a sequential manner.



Feature	$I_f^{s^T}$	F_f	I_f^s	$\{I_f^{s^T} F_f I_f^s\}$	$D = \prod_{f=1}^{2}\{I_f^{s^T} F_f I_f^s\} \odot$
Distance ($f = 1$)	$\begin{array}{cccccc} & C_1 & C_2 & C_3 & C_4 & C_5 \\ C_1 & 1 & 0 & 0 & 0 & 0 \\ C_2 & 0 & 1 & 0 & 0 & 0 \\ C_3 & 0 & 0 & 1 & 0 & 0 \\ C_4 & 0 & 0 & 0 & 1 & 0 \\ C_5 & 0 & 0 & 0 & 0 & 1 \end{array}$	$\begin{array}{cccccc} & C_1 & C_2 & C_3 & C_4 & C_5 \\ C_1 & 0 & 19 & 11 & 16 & 16 \\ C_2 & 16 & 0 & 14 & 17 & 12 \\ C_3 & 18 & 18 & 0 & 17 & 10 \\ C_4 & 19 & 19 & 11 & 0 & 12 \\ C_5 & 11 & 19 & 17 & 11 & 0 \end{array}$	$\begin{array}{cccccc} & C_1 & C_2 & C_3 & C_4 & C_5 \\ C_1 & 1 & 0 & 0 & 0 & 0 \\ C_2 & 0 & 1 & 0 & 0 & 0 \\ C_3 & 0 & 0 & 1 & 0 & 0 \\ C_4 & 0 & 0 & 0 & 1 & 0 \\ C_5 & 0 & 0 & 0 & 0 & 1 \end{array}$	$\begin{array}{ccccc} 0 & 19 & 11 & 16 & 16 \\ 16 & 0 & 14 & 17 & 12 \\ 18 & 18 & 0 & 17 & 10 \\ 19 & 19 & 11 & 0 & 12 \\ 11 & 19 & 17 & 11 & 0 \end{array}$	$\begin{array}{ccccc} 0 & 95 & 22 & 80 & 32 \\ 80 & 0 & 28 & 17 & 24 \\ 36 & 36 & 0 & 34 & 10 \\ 95 & 19 & 22 & 0 & 24 \\ 22 & 38 & 17 & 22 & 0 \end{array}$
Customer Zone ($f = 2$)	$\begin{array}{cccc} & Z_1 & Z_2 & Z_3 \\ C_1 & 1 & 0 & 0 \\ C_2 & 0 & 0 & 1 \\ C_3 & 0 & 1 & 0 \\ C_4 & 0 & 0 & 1 \\ C_5 & 0 & 1 & 0 \end{array}$	$\begin{array}{cccc} & Z_1 & Z_2 & Z_3 \\ Z_1 & 1 & 2 & 5 \\ Z_2 & 2 & 1 & 2 \\ Z_3 & 5 & 2 & 1 \end{array}$	$\begin{array}{cccccc} & C_1 & C_2 & C_3 & C_4 & C_5 \\ Z_1 & 1 & 0 & 0 & 0 & 0 \\ Z_2 & 0 & 0 & 1 & 0 & 1 \\ Z_3 & 0 & 1 & 0 & 1 & 0 \end{array}$	$\begin{array}{ccccc} 1 & 5 & 2 & 5 & 2 \\ 5 & 1 & 2 & 1 & 2 \\ 2 & 2 & 1 & 2 & 1 \\ 5 & 1 & 2 & 1 & 2 \\ 2 & 2 & 1 & 2 & 1 \end{array}$	

Figure 1 **An example for an instance with five nodes and two features (distance and parcel size) for calculating the transition weight matrix D^s.**

2.3. Decomposition of the Transition Weight Matrix

When certain levels of a feature only apply to a subset of the route instances, the transformation function can be decomposed based on these levels. In particular, a delivery operation originating from a depot in a city may lead to different tacit information that only applies to this depot. Therefore, the side information at this depot only applies to routes in their delivery region. Hence, the transformation function can then be decomposed. In the particular case of different depots, a feature impact matrix F can be learned per city.

3. Learning the Transformation by Predictive Analytics

The objective of DD-TSP is to solve (4) optimally and thereby learn the 'optimal' feature impact matrices F_f for $f \in \mathcal{F}$, which represents the preferences and the tacit information of the decision makers. Learning these preferences from the data can be achieved using descriptive or predictive methods. In this section, we focus our attention on predictive methods and present two methods for learning preferences from the data. In a case study in Section 4, we also show an application of learning by descriptive analytics. As discussed in Section 2.2, our framework assumes learning features independently in a sequential manner. Therefore, the focus in this section is learning one single feature only.

Recall that we assume no structure for the scoring function σ. Therefore, we do not have access to its gradient vector or its Hessian matrix. Within the realm of Derivative-Free Optimization (DFO), there exists several approaches ranging from simple search methods such as the Nelder-Mead Simplex Algorithm (Nelder and Mead 1965) to complex direct search methods such as Generalised Pattern Search and Mesh Adaptive Direct Search (Audet and Hare 2017). These methods are not scalable and are often used when optimizing approximately 50 or fewer variables (Audet and Hare 2017). Our preliminary experiments using DFO also confirmed this result. The size of a feature impact matrix, on the other hand, may reach up to a million variables to learn, which is beyond the size of the instances that can be handled using DFO algorithms. Our attempts to use metaheuristics including the genetic algorithm, the simulated annealing and the tabu search did not also yield any meaningful improvements in the overall score. We now present two algorithms that lead to improvements in terms of learning preferences. The first one is inspired by a DFO algorithm, the Coordinate Search, while the second one is inspired by supervised learning algorithms.

3.1. Score Guided Coordinate Search

Coordinate Search (CS) is an iterative algorithm that examines the coordinate directions (by varying one variable at a time) at decreasing step sizes (Audet and Hare 2017). If an improvement is detected, the incumbent is updated, and the search is terminated when certain conditions are met (such as time limit or iteration count). When solving the DD-TSP using the basic implementation of the CS algorithm, the matrix F_f is modified by changing one element at a time and all such directions are explored to improve the score of the routes generated by optimization oracle $\eta(D^s)$. Since no derivative information of the scoring function is available, the CS algorithm explores all directions at the incumbent solution in order to determine an improving direction. This is particularly very costly when the scoring function is computationally demanding. In this section, we present the Score Guided Coordinate Search (SGCS), which we develop by extending the basic CS algorithm. Instead of testing all coordinate directions, we guide the search by incorporating score information about potentially 'good' directions. Such good directions are provided by a proxy function as described in the next section.

3.1.1. A Proxy Gradient Function: Scoring function is used to evaluate the performance of the routes generated using a transition weight matrix D^s, which is obtained by feature impact matrices F_f for $f \in \mathcal{F}$. Observe that some features are common among instances such as customer zones and parcel sizes, whereas some are customer-specific such as the distances to the other nodes. For feature $f \in \mathcal{F}$ that is shared between instances, we define a *weighted scoring matrix* γ^f to measure the impact of every feature level. For $f \in \mathcal{F}$, let ℓ_1^f and ℓ_2^f be two feature levels, and $S_{\ell_1^f \ell_2^f} \subseteq S$ be the set of problem instances, in which particular transition from ℓ_1^f to ℓ_2^f is observed in the executed route $\hat{x}_s$. We then define

$$\gamma^f_{\ell_1^f \ell_2^f} = \frac{\sum\limits_{s \in S_{\ell_1^f \ell_2^f}} \sigma(\eta(D^s), \hat{x}_s)}{|S_{\ell_1^f \ell_2^f}|} \tag{5}$$

to be the *weighted score* of the particular transition from a node with feature ℓ_1^f to another node with feature ℓ_2^f if $|S_{\ell_1^f \ell_2^f}| \neq 0$, and $\gamma^f_{\ell_1^f \ell_2^f} = 0$ otherwise. Observe that this value allows us to measure the performance of a particular transition between features levels. It is the average score of those routes in which transition from ℓ_1^f to ℓ_2^f is observed. This measure is particularly valuable in guiding the CS algorithm. If this average value is poor, it indicates that the routes using this particular transition obtained worse scores than other routes that did not use this transition. Even though this is not the derivative of the scoring function σ, it acts as a proxy to approximate the behavior of the scoring function in a single coordinate direction. Observe that the dimensions of the weighted scoring matrix γ^f and the feature impact matrix F_f are the same, and we now have a performance measure for each element in F_f matrix. Clearly, if a particular transition is observed more on those routes with poor scores, then increasing the value of the corresponding element in F_f will discourage this particular transition from appearing in the routes. This is the key idea of the SGCS algorithm, which is presented next.

3.1.2. The SGCS Algorithm The SGCS algorithm explores the coordinate directions guided by the weighted scoring matrix γ^f to improve the performance of the poor-performing transitions. Considering the proxy gradient function and its output γ^f matrix, we increase the value of the element in the feature impact matrix F_f that corresponds to poor-performing elements in the γ^f matrix. By penalizing the transitions observed in poorly scoring routes, we guide the solver to explore alternative transitions. The algorithm is therefore guided to target the transitions which are universally unfavourable across routes, and encourage preferable alternatives. Additionally, this approach allows us to avoid exploring transitions that are unlikely to produce significant improvements. The SGCS is presented in Algorithm 1.

For a given feature $\hat{f}$, the algorithm takes an initial estimate of the feature impact matrix $F_{\hat{f}}^{input}$ as input and allocates it to a temporary working matrix Φ_{temp} (line 1). It will initially calculate the transition weight matrix D^s using Φ_{temp} and use optimization oracle $\eta(\cdot)$ to generate proposed routes for all $s \in S$ (lines 2–4). The minimum loss (Min_Loss) is calculated for these proposed routes (line 5). The objective of the algorithm is to improve the Min_Loss by modifying Φ_{temp}. Observe that different Φ_{temp} matrices yield different D^s matrices for $s \in S$, which in turn yield different routes by the optimization oracle, and the score is accordingly impacted. The outer loop (line 6) iterates $itCount$ times and starts by calculating the weighted score matrix $\gamma^{\hat{f}}$ at the current solution x_s^*. Recall that the dimensions of $\gamma^{\hat{f}}$ and Φ_{temp} matrices are the same. In the inner loop (line 8), we conduct trials with weight adjustments by modifying a batch of coordinates at a time. In particular, the algorithm penalizes ε percent of the worst-performing transitions in $\gamma^{\hat{f}}$ at every iteration. The penalty is Δ in the first round when $\ell = 1$, and it is equal to a

Algorithm 1: Score Guided Coordinate Search

Inputs: Set of instances S, executed routes $\hat{x}^s$, feature $\hat{f}$, initial solution ($F_{\hat{f}}^{input}$) for the feature impact matrix, iteration count ($itCount$), maximum number of trials at every iteration (Γ), number of unique elements changed (ε), step size (Δ)

Functions: $\eta(\cdot)$ is the optimization oracle and returns a route, $\sigma(\cdot)$ is the scoring function and returns a scalar, $\gamma(\cdot)$ is the weighted scoring function and returns a matrix, $find\_element\_order(\cdot)$ finds the ascending order of the value of an element among all elements in a given matrix

Output: $F_{\hat{f}}$

```
1  $\Phi_{temp} \leftarrow F_{\hat{f}}^{input}$
2  for $s \in S$ do
3      $D^s \leftarrow \{I^{s^T} \Phi_{temp} I^s\} \odot \prod_{f \in \mathcal{F} \setminus \hat{f}} \{I^{s^T} F_f I^s\} \odot$   /* Calculate the weight matrix */
4      $x_s^* \leftarrow \eta(D^s)$   /* Generate proposed routes using the $D^s$ matrix */
5  $Min\_Loss \leftarrow \sum_{s \in S} \frac{\sigma(x_s^*, \hat{x}_s)}{|S|}$
6  for $i = 1$ to $itCount$ do
7      Calculate $\gamma^{\hat{f}}$ using $\sigma(x_s^*, \hat{x}_s)$ for $s \in S$   /* Calculate the weighted score matrix */
8      for $j = 1$ to $\Gamma$ do
9          for $\ell = 1$ to $2$ do
10             if $\ell == 1$ then $penalty = \Delta$
11             else $penalty = 1000$
12             for every element $\gamma_{ij}^{\hat{f}}$ in $\gamma^{\hat{f}}$ matrix do
13                 $H_{ij} = \begin{cases} penalty & \text{if } 100\varepsilon(j-1) \leq find\_element\_order(\gamma_{ij}^{\hat{f}}) < 100\varepsilon j \\ 0 & \text{otherwise} \end{cases}$
14             $M_\ell \leftarrow \Phi_{temp} + H$
15             for $s \in S$ do
16                 $D_\ell^s \leftarrow \{I^{s^T} M_\ell I^s\} \odot \prod_{f \in \mathcal{F} \setminus \hat{f}} \{I^{s^T} F_f I^s\} \odot$
17                 $x_s^{**} \leftarrow \eta(D_\ell^s)$
18             if $\sum_{s \in S} \frac{\sigma(x_s^{**}, \hat{x}_s)}{|S|} < Min\_Loss$ then
19                 $\Phi_{temp} \leftarrow M_\ell$
20                 $x_s^* \leftarrow x_s^{**}$
21                 $Min\_Loss \leftarrow \sigma(x_s^{**}, \hat{x}_s))$
22                 break $j$ loop
23     $\Phi_{temp} \leftarrow M_\ell$   /* Use the last tested $M_\ell$ matrix as the incumbent */
24     $x_s^* \leftarrow x_s^{**}$
25 $F_{\hat{f}} \leftarrow \Phi_{temp}$
26 return $F_{\hat{f}}$
```

large value, 1000, when $\ell = 2$. At every inner iteration on j, the algorithm selects j^{th} ε percent of the worst performing scores. For example, if $\varepsilon = 2$, the algorithm first assigns a penalty for 2% of the worst performing transitions. In the second iteration on j, a penalty is assigned for 2−4% worst-performing transitions, and iterations continue until $j = \Gamma$. If at any iteration an improvement on *Min_Loss* is achieved, the incumbent solution is updated. If no improvement is made in Γ iterations, the algorithm takes the last tested matrix M_ℓ as the incumbent (line 23), which diversifies the search. In other words, the algorithm does a local search using coordinate search, and if no improvement is found, the weighted score matrix is recalculated in the outer loop using the new incumbent, and the search continues. The algorithm terminates when *itCount* iterations are completed.

3.2. Label-guided Feature Impact Matrix

Inspired by supervised learning, we also develop a non-iterative training method as a closed-form solution. Recall that we use the weighted scoring matrix γ^f as a proxy to guide the search algorithm. Each entry in this matrix represents how frequently a particular transition between feature levels is observed. Observe that the *executed routes* can also be evaluated using the same function, which expresses the transitions that are frequently seen in the testing dataset. For $f \in \mathcal{F}$, let ℓ_i^f and ℓ_j^f be two feature levels of two nodes i and j, and $L^f_{\ell_i^f \ell_j^f}$ be the number of transitions from every node i with a feature level ℓ_i^f to every other node j with feature level ℓ_j^f, which are observed in the executed routes $\hat{x}_s$ for $s \in S$. For $f \in \mathcal{F}$, we have

$$F_f^{ij} = \begin{cases} 1 & \text{if } L^f_{\ell_i^f \ell_j^f} = 0 \\ \frac{1}{L^f_{\ell_i^f \ell_j^f}} & \text{if } 1 \leq L^f_{\ell_i^f \ell_j^f} \leq \kappa \\ \frac{1}{\kappa} & \text{otherwise} \end{cases} \qquad (i,j) \in A, \tag{6}$$

where $\kappa \in \mathbb{N}$ represents the maximum number of transitions that can change the weight value. We refer to this approach as Label-guided Feature Impact Matrix (LFIM), which bares the same simplicity of RFIM, but takes a more granular approach making use of the data directly.

4. Description of the Case Study

We now present a real-world case study based on by Amazon Last-mile Routing Challenge 2021 competition (Amazon 2021). In this section, we first present the data and the competition in Section 4.1, and a scoring function in Section 4.2. We then extract and select features using descriptive analytics of the data in Section 4.3. Using the results of the descriptive analysis, we present a rule-based method to construct a feature impact matrix in Section 4.4. Finally, we combine the ideas of the rule-based and label-guided methods to devise a combined method and present it in Section 4.5.

Table 2 Depot Information

Depot	Number of Routes by Quality				Number of Zones
	High		Medium	Low	
	High-SV	High-MV			
DAU1	52	44	114	4	2786
DBO1	12	17	29	2	617
DBO2	67	48	178	3	2306
DBO3	73	89	403	8	3341
DCH1	30	81	85	0	1813
DCH2	18	48	88	0	1338
DCH3	78	18	167	8	1914
DCH4	72	36	269	4	2331
DLA3	30	128	89	7	1411
DLA4	19	59	116	3	1119
DLA5	28	22	103	2	1376
DLA7	164	283	673	13	4266
DLA8	53	227	156	12	3758
DLA9	185	195	301	20	4829
DSE2	24	74	26	1	563
DSE4	112	137	193	4	1809
DSE5	90	105	302	11	1255
Total	1107	1611	3292	102	35702

4.1. The Competition and the Data

This section briefly describes the dataset by Merchan et al. (2021), which includes 6112 routes from five major locations and their surrounding cities: Los Angeles, Seattle, Chicago, Boston, and Austin. These five major locations contain more than one servicing depot, and there are a total of 17 depots across all the cities. The number of routes per depot is presented in Table 2. The customer regions that are serviced by depots are non-overlapping. The routes are classified by operational experts into *high, medium,* and *low* qualities. Some high quality routes contain multiple visits to the same drop-off node due to failed a first attempt delivery and a subsequent re-attempt later on on the same route. We therefore further break down high quality routes into high-single-visit (*High-SV*) and high-multi-visit (*High-MV*) routes. Collectively, we refer to High-SV and High-MV routes as *High* routes. The aim is to generate routes similar to High-SV ones. The dataset contains the executed routes, and each route contains information on nodes that are visited and the visiting order, along with information on the packages delivered to each node.

4.2. Comparing Similarity of Two Routes: the Scoring Function

The scoring function introduced in Section 2 measures the dissimilarity between two routes. Recall that, in the DD-TSP, the objective is to imitate an experienced driver and to generate routes similar to historically executed ones. This requires expressing the difference between an estimated route and an executed one. Observe that, even though the two routes may have the same duration and length, they could still be significantly different in the order of customer visits, which would imply dissimilarity between the two routes. A scoring metric is developed in the context of Amazon

Last-mile Routing Challenge, which is a function of the *sequence deviation* and a variation of *edit distance* (Ristad and Yianilos 1998). The sequence deviation measures how different two routes are, and it captures differences in the ordering of nodes (regardless of the physical distance between the nodes). The edit distance, on the other hand, measures the number of single-element operations (insertions, deletions, and substitutions) required to transform one route to the other. Operations are also weighted by the physical distance of the nodes involved. More details about the scoring metric is available by Mo et al. (2023). The score produces a value between 0 and a maximum possible score (between 0.17 and 1.77, depending on the estimated route), which is determined by the operational experts. A perfect estimation of a historic route will result in a score of 0 and a random ordering of the customer nodes visited will result in a poor score close to the maximum possible score. In this case study, we adopt the scoring metric developed in this context; nevertheless, the methods are also valid for any nonconvex and nonsmooth functions.

4.3. Descriptive Statistics and Feature Selection

Model features are the inputs that models use during training and generation of solutions. Descriptive analyses of the data (presented in Appendix A) show that features that have a strong impact of the routes are (1) the travel times, (2) the first and last nodes on the route, and (3) the zone IDs of the customers. A zone ID, is formed of four tokens (such as *A1.1A*). In the executed routes in the dataset, two consecutive nodes on the routes 85.2% of the times share the same zone ID, that is, all 4 tokens are the same. If two consecutive nodes do not have the same zone IDs, the transition to another zone is likely to share 3 tokens 11.9% of the time. In the context of this case study, we mainly focus on the zone ID among the three features since it offers the most potential for performance improvements as also observed in different studies focusing the same case study (Cook, Held, and Helsgaun 2024, Mo et al. 2023, Özarık, da Costa, and Florio 2024, Accorsi, Lodi, and Vigo 2022). In the remainder of this paper, the goal is to learn the feature impact matrix of zone ID feature.

4.4. Rule-based Feature Impact Matrix

Taking the travel times and the zone IDs of the nodes into account, we build a feature impact matrix F_f in a rule-based fashion, which we refer to as rule-based feature impact matrix (RFIM). Let $x_{ij}^k = 1$ if the kth token of the zone ID of nodes i and j are different, and 0 otherwise. The

RFIM constructs the feature impact matrix F_f as follows:

$$F_f^{ij} = \begin{cases} b_1 & \text{if } i = 0 \\ b_2 & \text{if } j = 0 \\ a_1 & \text{if } i, j \neq 0, x_{ij}^1 = 1 \text{ and } \sum_{k=2}^{k=4} x_{ij}^k = 3 \\ a_2 & \text{if } i, j \neq 0, x_{ij}^1 = 1 \text{ and } \sum_{k=2}^{k=4} x_{ij}^k = 2 \\ a_3 & \text{otherwise} \end{cases} \qquad (i,j) \in A, \tag{7}$$

where $(b_1, b_2, a_1, a_2, a_3)$ are 'discouragement multipliers' with $a_1 < a_2 < a_3$. Hyperparameter a_1 is for the nodes in the same zone ID, which we set equal to 1. Hyperparameter a_2 is the multiplier between two nodes i and j whose zone IDs share three tokens including the first token. Hyperparameter a_3 is the discouragement multiplier between all other node pairs when neither of the two nodes is the depot. If the tail or head of the arc is the depot, the weight is b_1 or b_2, respectively. The RFIM method can be considered as expert views on the route generation process, because it is based on rules.

4.5. Combining the Label-guided and the Rule-based Feature Impact Matrices

The RFIM is general and can perform well on unforeseen data, for which the LFIM is limited. On the other hand, RFIM is generic and does not capture the nuances between similar feature levels, for which the LFIM performs well and captures the subtle differences between feature levels. We therefore propose a combined method, which we refer to as *R+LFIM*, by using the LFIM when a particular transition is seen in the dataset, and using RFIM otherwise. It can also be viewed as discarding any RFIM influence as long as the transition is seen in the training data for LFIM. Note that the RFIM can broadly be considered a method built on expert views, constructed in a rule-based fashion, while the LFIM is a data-driven method. The R+LFIM combines the best of both methods, utilizing their respective strengths.

5. Results and Discussion

We now present the implementation details in Section 5.1 and compare the performances of the methods in Section 5.2. We then test the performance of the best performing method under different settings and demonstrate its generalizability in an out-of-sample dataset in Section 5.3.

5.1. Implementation Details

We implemented our algorithms using Python 3.9, and all the experiments were conducted on the Cedar cluster of Compute Canada using single thread and 64GB of RAM on a Linux environment. We have used the LKH 2.0.10 solver (Keld Helsgaun 2022) as the optimization oracle unless otherwise stated. The time limit of LKH is set to 0.5 seconds. The learning is decomposed based on depots (as described in Section 2.3). We use the distance, the first and the last node, and the zone IDs as the features in the computational study in this paper, and focus on learning the feature impact matrix of the zone ID feature.

5.2. Comparison of Algorithm Performances

We now present computational results of the RFIM (Section 4.4), SGCS (Section 3.1), LFIM (Section 3.2) and R+LFIM (Section 4.5) methods. We use High-SV quality routes to compare the performances.

5.2.1. RFIM Performance: We have tested RFIM performance using our baseline hyperparameter settings, $(b_1, b_2, a_1, a_2, a_3) = (1, 0, 1, 10, 100)$ and have obtained an average score of 0.0367. We have also tested 494 randomly generated values for these hyperparameters, but none of these implementations achieved a better score than the baseline hyperparameter settings (see Appendix B.1 for details on tuning). We note that the RFIM method has achieved a third place in the Amazon competition, and we take it as our baseline when presenting the performance of the two other methods.

5.2.2. SGCS Performance The SGCS algorithm has four hyperparameters. We first tested a variety of settings and determined the best setting as ($itCount$, Γ, ε, Δ)= $(10, 5, 0.05, 50)$, as detailed in Appendix B.2. Furthermore, the initial solution for the feature impact matrix ($\Phi_{initial}$) is taken as the solution given by the RFIM. The experiments for the SGCS were carried out using High-SV routes and an 80% training and 20% testing split. The average scores of the SGCS algorithm per depot on training and testing datasets are presented in Table 3. The table also presents the results of the RFIM and compares the performance of the two methods. The first column is the depot name. The second and third columns report the average score per depot in training dataset for the RFIM and the SGCS algorithms, respectively. The average score is improved from 0.0362 to 0.0332 in the training dataset, an average of 8.2% improvement. The average scores of the two methods on the testing dataset are reported in the fourth and fifth columns, respectively. The average scores are 0.0386, and 0.0393 for the RFIM and the SGCS, deteriorating 1.5%. The score changes in percentage from the RFIM to the SGCS per depot on testing and training datasets are reported in sixth and seventh columns, respectively. The SGCS algorithm improves the average score on the training set, and we also observe improvements in the test data set for some depots such as DCH3 at -10.5% and DCH4 at -8.1%. Nevertheless, it fails to improve the performance of the RFIM method on the entire testing dataset and lacks the ability to generalise beyond seen routes.

Table 3 **Performance of RFIM and SGCS algorithms per depot on High-SV routes an 80% training and 20% testing split.**

Depot	RFIM Training score	SGCS Training score	RFIM Testing score	SGCS Testing score	Score change (%) on training dataset	Score change (%) on testing dataset
DAU1	0.0555	0.0521	0.0420	0.0611	−6.2	45.6
DBO1	0.0620	0.0454	0.0838	0.0838	−26.8	0.0
DBO2	0.0433	0.0405	0.0643	0.0646	−6.5	0.5
DBO3	0.0263	0.0246	0.0252	0.0241	−6.5	−4.2
DCH1	0.0532	0.0498	0.0446	0.0446	−6.4	0.0
DCH2	0.0744	0.0731	0.0611	0.0617	−1.7	1.0
DCH3	0.0342	0.0318	0.0434	0.0388	−7.1	−10.5
DCH4	0.0333	0.0300	0.0514	0.0472	−8.7	−8.1
DLA3	0.0291	0.0200	0.0278	0.0300	−31.2	7.8
DLA4	0.0369	0.0232	0.0155	0.0143	−37.2	−7.8
DLA5	0.0256	0.0224	0.0306	0.0338	−12.7	10.4
DLA7	0.0282	0.0249	0.0235	0.0221	−11.7	−5.9
DLA8	0.0345	0.0313	0.0402	0.0400	−9.3	−0.5
DLA9	0.0407	0.0401	0.0464	0.0455	−1.5	−1.8
DSE2	0.0612	0.0516	0.0658	0.0755	−15.7	14.7
DSE4	0.0259	0.0249	0.0218	0.0221	−3.7	1.1
DSE5	0.0327	0.0297	0.0377	0.0420	−9.4	11.5
Wt.Avg.	**0.0362**	**0.0332**	**0.0386**	**0.0392**	**−8.2**	**1.5**

5.2.3. LFIM Performance The only hyperparameter of the LFIM is the κ, which we set as 5 (see Annex B.3 for hyperparameter tuning details). The performance of the LFIM in comparison to the RFIM per depot on High-SV routes with an 80% training and 20% testing split is reported in Table 4. The column titles correspond to those in Table 3, indicating similar measures for comparing the performances of the RFIM and the LFIM methods. The weighted average score of the LFIM is 0.0374 and 0.0575 on training and testing datasets, respectively, which correspond to 3.4% and 48.8% deterioration with respect to the those of RFIM, respectively. Observe that the RFIM is more generic and can be implemented to any unseen data, whereas the LFIM is more dependent on the data seen in the training dataset. If a particular transition is never seen, no information is transferred, which leads to poor performance with respect to the RFIM method.

5.2.4. R+LFIM Performance: Combining the strong sides of both RFIM and LFIM methods, R+LFIM method is developed in Section 4.5. The performance of the R+LFIM with respect to the RFIM method is reported in Table 5 per depot on High-SV routes with an 80% training and 20% testing split. The column titles match those in Table 3, indicating similar measures for comparing the performances of the RFIM and the R+LFIM methods. The weighted average score of the R+LFIM is 0.0108 and 0.0299 on training and testing datasets, respectively, which correspond to 70.1% and 22.5% improvement with respect to the RFIM, respectively. This clear improvement

Table 4 **Performance of RFIM and LFIM algorithms per depot on High-SV routes with an 80% training and 20% testing split.**

Depot	RFIM Training score	LFIM Training score	RFIM Testing score	LFIM Testing score	Score change (%) on training dataset	Score change (%) on testing dataset
DAU1	0.0555	0.0666	0.0420	0.0708	19.9	68.7
DBO1	0.0620	0.0516	0.0838	0.1270	−16.8	51.6
DBO2	0.0433	0.0438	0.0643	0.0896	1.0	39.5
DBO3	0.0263	0.0284	0.0252	0.0440	8.0	74.6
DCH1	0.0532	0.0626	0.0446	0.0959	17.7	114.8
DCH2	0.0744	0.0892	0.0611	0.1295	19.9	112.0
DCH3	0.0342	0.0398	0.0434	0.0596	16.3	37.3
DCH4	0.0333	0.0342	0.0514	0.0678	2.8	32.1
DLA3	0.0291	0.0305	0.0278	0.0577	4.9	107.7
DLA4	0.0369	0.0476	0.0155	0.0188	28.9	21.2
DLA5	0.0256	0.0388	0.0306	0.0426	51.4	39.1
DLA7	0.0282	0.0235	0.0235	0.0370	−16.7	57.2
DLA8	0.0345	0.0462	0.0402	0.0731	34.0	81.9
DLA9	0.0407	0.0383	0.0464	0.0605	−6.0	30.3
DSE2	0.0612	0.0654	0.0658	0.0907	6.9	37.8
DSE4	0.0259	0.0251	0.0218	0.0330	−3.1	51.2
DSE5	0.0327	0.0289	0.0377	0.0465	−11.6	23.4
Wt.Avg.	**0.0362**	**0.0374**	**0.0386**	**0.0575**	**3.4**	**48.8**

is achieved by using the available data to the best extend possible by the LFIM, while consulting to the RFIM for unforeseen data due to its performance on unforeseen data. Overall, the method achieves the best result among all the other methods considered. We hereafter use the R+LFIM as our best performing method and test it under different settings in the next section.

5.3. Performance of the Best Performing Algorithm under Different Settings

The R+LFIM is the computationally demonstrated to be the best performing algorithm in terms of both the average score it achieves in testing dataset and its computational efficiency due to not requiring any iterations. We first test the impacts of the split ratio on the training and testing performances in Section 5.3.1. We then test the R+LFIM method on High routes in Section 5.3.2, on all routes in Section 5.3.3 and on an out-of-sample dataset, which was also used in the competition, in Section 5.3.4. Lastly, we test an implementation which takes into account the customer time windows in the optimization oracle in Section 5.3.5.

5.3.1. R+LFIM Performance on Various Training and Testing Splits The R+LFIM method trains on the available data and the size of training data is therefore crucial for the ability to generalize to unseen routes in the testing set. We now investigate the impacts of changing the size of the training and testing datasets between 10% and 90% of the High-SV routes. The results

Table 5 Performance of RFIM and R+LFIM algorithms per depot on High-SV routes with an 80% training and 20% testing split.

Depot	RFIM Training score	R+LFIM Training score	RFIM Testing score	R+LFIM Testing score	Score change (%) on training dataset	Score change (%) on testing dataset
DAU1	0.0555	0.0131	0.0420	0.0419	−76.3	−0.3
DBO1	0.0620	0.0188	0.0838	0.0838	−69.8	0.0
DBO2	0.0433	0.0111	0.0643	0.0516	−74.4	−19.6
DBO3	0.0263	0.0073	0.0252	0.0177	−72.1	−29.6
DCH1	0.0532	0.0181	0.0446	0.0550	−65.9	23.2
DCH2	0.0744	0.0259	0.0611	0.0621	−65.2	1.6
DCH3	0.0342	0.0112	0.0434	0.0318	−67.3	−26.8
DCH4	0.0333	0.0067	0.0514	0.0411	−79.8	−20.0
DLA3	0.0291	0.0071	0.0278	0.0279	−75.7	0.4
DLA4	0.0369	0.0122	0.0155	0.0138	−67.1	−11.2
DLA5	0.0256	0.0061	0.0306	0.0275	−76.2	−10.4
DLA7	0.0282	0.0084	0.0235	0.0161	−70.4	−31.7
DLA8	0.0345	0.0107	0.0402	0.0359	−68.9	−10.8
DLA9	0.0407	0.0108	0.0464	0.0357	−73.5	−23.1
DSE2	0.0612	0.0261	0.0658	0.0452	−57.4	−31.4
DSE4	0.0259	0.0107	0.0218	0.0149	−58.9	−31.6
DSE5	0.0327	0.0119	0.0377	0.0149	−63.5	−60.6
Wt.Avg.	**0.0362**	**0.0108**	**0.0387**	**0.0299**	**−70.1**	**−22.5**

are presented in Table 6. The first column shows the size of the training data as percentage of the High-SV routes, and the remaining routes are used for testing. A value of 10 implies that the training dataset contains 10% of the routes, and the remaining 90% of the routes are for used for testing. The following four columns in the table report the average training and testing scores for the RFIM and the R+LFIM methods. The improvements achieved by the R+LFIM over the RFIM are reported in the rightmost two columns.

The average training score has a direct relationship with the training split while the testing score has an inverse relationship. At 10%, the training score of the R+LFIM method is as low as 0.0081, while it increases to 0.0119 when the split is 100%. The reason is that less data is easier to learn, but the performance on the training set does not generalize well, as seen in the testing results. Note that the best performing algorithm in the training dataset reported so far is by Cook, Held, and Helsgaun (2024) and is equal to 0.01978. With our R+LFIM method, we achieved a score of 0.0119 on the complete dataset, which improves over the best known performance by 40.1%. The average score of R+LFIM improves from 0.0349 to 0.0298 in testing dataset as the split proportion increases from 10% to 90%. An average score of 0.0296 at an 90% training data split implies approximately 23.6% improvement over the RFIM results, which achieved a third place

Table 6 **R+LFIM performance with respect to RFIM on High-SV routes for varying training split proportions**

Training split (%)[†]	RFIM Training score	R+LFIM Training score	RFIM Testing score	R+LFIM Testing score	Score change (%) on training dataset	Score change (%) on testing dataset
10	0.0399	0.0081	0.0363	0.0349	−79.5	−3.9
20	0.0393	0.0090	0.0360	0.0338	−77.0	−6.2
30	0.0377	0.0089	0.0362	0.0329	−76.4	−9.0
40	0.0369	0.0096	0.0364	0.0320	−73.9	−12.2
50	0.0368	0.0099	0.0364	0.0306	−73.1	−16.1
60	0.0364	0.0101	0.0371	0.0304	−72.2	−18.1
70	0.0362	0.0106	0.0378	0.0306	−70.6	−19.0
80	0.0362	0.0108	0.0387	0.0299	−70.1	−22.5
90	0.0364	0.0113	0.0389	0.0296	−69.0	−23.6
100	0.0366	0.0119	-	-	−67.5	-

[†] The remaining data is used for testing.

in the Amazon competition. As we will demonstrate in Section 5.3.4, the method also achieves a 23.1% improvement on an out-of-sample dataset, which was used in the competition.

Figure 2 displays the score change reported in the rightmost two columns in Table 6. As more data is available, the testing performance increases while training performance decreases. Availability of more data clearly allows to generalize better. The figure additionally reports the improvements that the algorithm achieves on High routes, details of which are presented in the next section.

5.3.2. R+LFIM Performance on High Routes: We next conduct experiments using all 2718 High quality routes (including High-SV and High-MV). The results are reported in Table 7. The R+LFIM achieves a better average score of 0.0282 on High routes than on the High-SV routes. Observe that reattempted deliveries found in High-MV routes imply additional node visits unknown *a priori* to the route generation. These reattempts may lead the driver to take undesired transitions, which may eventually misguide the R+LFIM method. Despite this, the results continued to improve when new routes were added. This demonstrates the value of including additional data, which offsets any drawbacks from incorporating the High-MV routes into the learning process. Note that the new score achieved by the algorithm yields a 29.9% improvement with respect to RFIM baseline score. The score change with respect to varying training splits is also plotted in Figure 2. The R+LFIM performs better in every setting when new data is introduced to train.

5.3.3. R+LFIM Performance on All Routes: We have computationally demonstrated in the previous section that including greater number of High quality routes in testing data helped improve the results. This naturally raises the question whether medium and low quality routes would also lead to similar results. We now incorporate all 6112 routes including the medium and low quality ones. We use a training split of 0.5 on the High-SV routes, and use the other half for

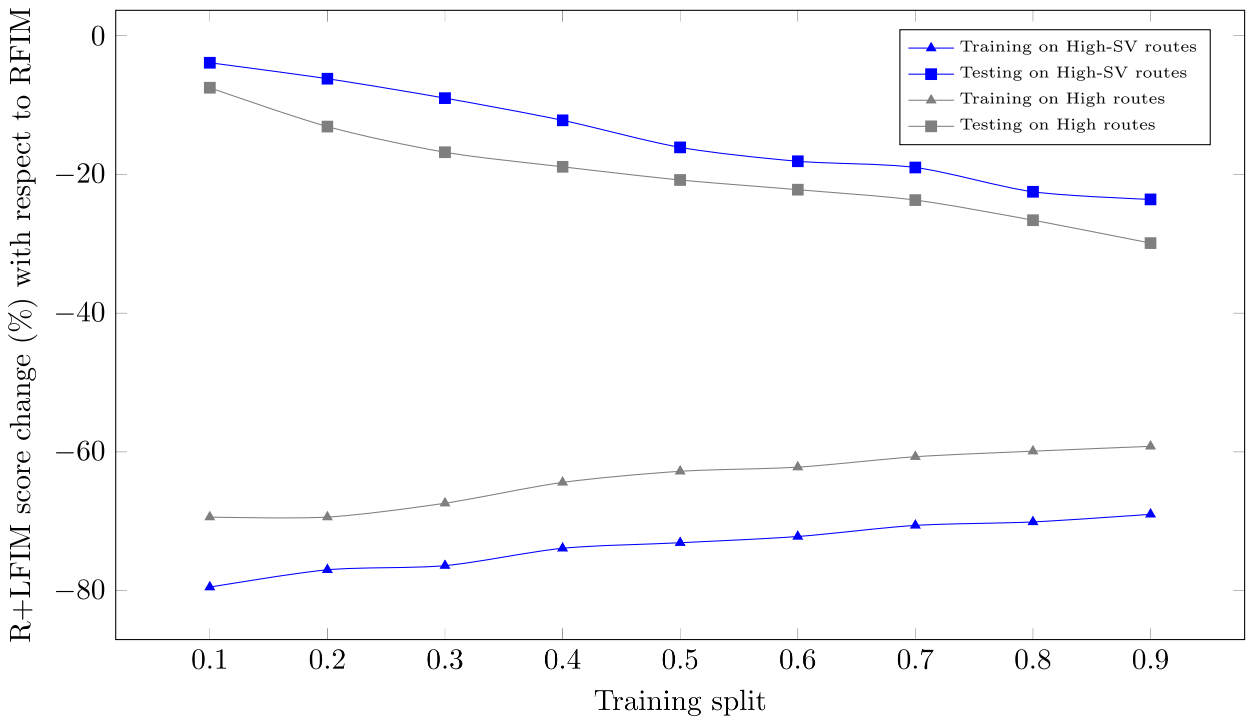


Figure 2 R+LFIM Scores with respect to RFIM on High-SV and High routes for different training and testing split ratios. Negative numbers on the vertical axis indicate the scale of improvement over RFIM.

Table 7 R+LFIM performance with respect to RFIM on High routes for varying training split proportions

Training split (%)[†]	RFIM Training score	R+LFIM Training score	RFIM Testing score	R+LFIM Testing score	Score change (%) on training dataset	Score change (%) on training dataset
10	0.0393	0.0121	0.0403	0.0373	−69.3	−7.5
20	0.0402	0.0118	0.0406	0.0353	−69.4	−13.1
30	0.0396	0.0129	0.0405	0.0337	−69.4	−16.8
40	0.0392	0.0140	0.0409	0.0331	−67.4	−18.9
50	0.0397	0.0148	0.0407	0.0323	−64.4	−20.8
60	0.0396	0.0150	0.0411	0.0319	−62.8	−22.2
70	0.0399	0.0157	0.0410	0.0312	−60.7	−23.7
80	0.0397	0.0159	0.0422	0.0310	−59.9	−26.6
90	0.0402	0.0164	0.0401	0.0282	−59.2	−29.9
100	0.0403	0.0168	-	-	−58.4	-

[†] The remaining data is used for testing.

our testing dataset. All the remaining routes also form part of the training dataset. The results are presented in Table 8 by depot. The weighted average scores of the RFIM and the R+LFIM on training dataset are 0.0427 and 0.198, respectively. Including the medium and low quality routes increased the training score with respect to those of High routes. The weighted average score of the RFIM and the R+LFIM on testing dataset are 0.0408 and 0.0283. We observe that the performance of the method on all routes remain on par with the previously reported results, demonstrating the robustness of the method.

Table 8 **R+LFIM performance per depot with respect to RFIM on All routes**

Depot	RFIM Training score	R+LFIM Training score	RFIM Testing score	R+LFIM Testing score	Score change (%) on training dataset	Score change (%) on testing dataset
DAU1	0.0626	0.0245	0.0656	0.0578	−60.9	−11.8
DBO1	0.0661	0.0396	0.1033	0.0871	−40.1	−15.7
DBO2	0.0562	0.0307	0.0509	0.0392	−45.3	−23.0
DBO3	0.0309	0.0124	0.0266	0.0140	−60.0	−47.5
DCH1	0.0601	0.0276	0.0534	0.0504	−54.1	−5.6
DCH2	0.0731	0.0390	0.0571	0.0665	−46.6	16.5
DCH3	0.0466	0.0240	0.0413	0.0342	−48.5	−17.2
DCH4	0.0417	0.0213	0.0430	0.0262	−48.9	−38.9
DLA3	0.0401	0.0162	0.0367	0.0237	−59.6	−35.6
DLA4	0.0431	0.0217	0.0411	0.0283	−49.6	−31.2
DLA5	0.0362	0.0130	0.0323	0.0250	−64.2	−22.6
DLA7	0.0362	0.0162	0.0335	0.0191	−55.3	−42.8
DLA8	0.0425	0.0186	0.0383	0.0302	−56.4	−21.2
DLA9	0.0467	0.0191	0.0444	0.0290	−59.2	−34.6
DSE2	0.0562	0.0362	0.0619	0.0489	−35.6	−20.9
DSE4	0.0329	0.0150	0.0283	0.0149	−54.5	−47.4
DSE5	0.0382	0.0193	0.0341	0.0158	−49.4	−53.6
Wt.Avg.	**0.0427**	**0.0198**	**0.0408**	**0.0283**	**−53.6**	**−30.8**

5.3.4. R+LFIM Performance on Out-of-sample Routes: In the Amazon competition, the algorithms were tested in an out-of-sample dataset, and the performance of the top three teams were 0.0248, 0.0353 and 0.0391, respectively. The first place winners, Cook, Held, and Helsgaun (2024), implemented a local search for learned constraints. Second place winners, Mo et al. (2023), implemented a hierarchical TSP solution with a custom cost matrix. Additional details for both approaches can be found in Winkenbach, Parks, and Noszek (2021). Lastly our initial RFIM method achieved a third place with a score of 0.0391. Our focus on affecting the distances between nodes based on zone differences is echoed in all of the top three performing methods, albeit utilized in different ways. We consider our method the most direct and the least complicated, as it does not include any modifications to a TSP solver directly and returns comparative results.

Table 9 presents the computational results. The RFIM score presented here achieves a score of 0.0382 whereas the same method achieved 0.0391 in the competition. The difference is considered to be due to random noise in the heuristics to generate routes. We then trained R+LFIM using all routes and tested on the out-of-sample dataset. The new method improved the testing score from 0.0382 to 0.294, an improvement of approximately 23.1%.

Table 9 R+LFIM performance per depot with respect to RFIM on Out of Sample Dataset

Depot	Number of Routes	RFIM Training score	R+LFIM Training score	RFIM Testing score	R+LFIM Testing score	Score change (%) on testing dataset
DAU1	121	0.0629	0.0293	0.0508	0.0410	−14.82
DBO1	59	0.0744	0.0375	0.0606	0.0562	−9.43
DBO2	152	0.0548	0.0310	0.0438	0.0374	−6.89
DBO3	202	0.0303	0.0123	0.0268	0.0126	−41.97
DCH1	123	0.0573	0.0307	0.0614	0.0522	−12.24
DCH2	71	0.0701	0.0420	0.0516	0.0492	−0.20
DCH3	161	0.0465	0.0242	0.0393	0.0314	−11.02
DCH4	113	0.0426	0.0215	0.0336	0.0273	−9.36
DLA3	129	0.0389	0.0170	0.0342	0.0271	−17.80
DLA4	157	0.0425	0.0233	0.0391	0.0336	−7.08
DLA5	94	0.0359	0.0136	0.0312	0.0310	5.04
DLA7	397	0.0360	0.0161	0.0324	0.0199	−32.25
DLA8	353	0.0414	0.0185	0.0370	0.0303	−16.61
DLA9	496	0.0460	0.0206	0.0385	0.0325	−11.52
DSE2	114	0.0596	0.0368	0.0513	0.0313	−39.14
DSE4	166	0.0315	0.0154	0.0258	0.0143	−39.88
DSE5	143	0.0373	0.0183	0.0335	0.0190	−30.30
DBO6	1	–	–	0.0752	0.0752	0.0
Wt.Avg.		**0.0422**	**0.0204**	**0.0382**	**0.0294**	**−23.1**

5.3.5. Optimization Oracle With or Without Time Windows: We have used the LKH-2 solver (Keld Helsgaun 2022) in all the computational tests conducted so far even if some customers have time windows in the dataset. This naturally raises the question of whether respecting the time windows could potentially improve the results. We then used LKH 3.0.9 (Helsgaun 2017), which supports modeling time windows when solving the TSP. Here we trained on the High routes, with an 80% training and 20% testing data split using R+LFIM. The results are presented in Table 10. The performance is insignificantly impacted when the optimization oracle respects the time windows on the available data.

6. Discussion on Effectiveness, Interpretability and Flexibility of the Methods

We have had three modeling strategies that governed the choice of methods. All our solution techniques presented in this paper are interpretable, flexible and they effectively capture sufficient information for competitive performances.

Effectiveness: The RFIM method provides a baseline score of 0.0367 on the High-SV dataset. Given the fact it is based on expertly engineered rules filtered through descriptive analytics, there was no training procedure. Our second algorithm is the SGCS, which have attained scores of 0.0332 and 0.0393 on training and testing datasets, respectively. Lastly, the training and testing scores of R+LFIM, using only a portion of the High-SV routes for testing and all the remaining routes as testing, yields a training score of 0.0198 and testing score of 0.0283. It also achieves a score of

Table 10 Comparing the impacts of optimization oracle with and without time windows on R+LFIM performance on High routes

Depot	Without TW Training Score	With TW Training Score	Without TW Testing Score	With TW Testing Score	Score change (%) on training dataset	Score change (%) on testing dataset
DAU1	0.0243	0.0240	0.0741	0.0669	−1.47	−9.8
DBO1	0.0444	0.0483	0.0353	0.0339	8.73	−4.0
DBO2	0.0190	0.0191	0.0471	0.0468	0.22	−0.7
DBO3	0.0096	0.0099	0.0164	0.0162	3.47	−0.9
DCH1	0.0240	0.0239	0.0497	0.0513	−0.37	3.3
DCH2	0.0247	0.0252	0.0686	0.0689	2.24	0.4
DCH3	0.0136	0.0141	0.0286	0.0288	4.02	0.7
DCH4	0.0087	0.0083	0.0344	0.0327	−4.36	−4.8
DLA3	0.0147	0.0146	0.0265	0.0266	−0.94	0.2
DLA4	0.0190	0.0191	0.0401	0.0403	0.83	0.4
DLA5	0.0086	0.0085	0.0300	0.0325	−0.90	8.4
DLA7	0.0122	0.0125	0.0177	0.0184	2.48	4.3
DLA8	0.0140	0.0142	0.0392	0.0398	0.96	1.4
DLA9	0.0169	0.0168	0.0328	0.0338	−0.70	3.1
DSE2	0.0313	0.0320	0.0447	0.0449	1.96	0.5
DSE4	0.0143	0.0143	0.0151	0.0151	−0.13	0.1
DSE5	0.0140	0.0140	0.0169	0.0165	−0.46	−2.3
Wt.Avg.	**0.0155**	**0.0156**	**0.0305**	**0.0304**	**0.68**	**−0.17**

0.0294 on an out-of-sample dataset. The key feature of the R+LFIM is that it does not require any iterative learning aspect, but it is a data transformation that can be executed in seconds. It works immediately in conjunction with established industrial solvers to provide effective and competitive results.

Interpretability: Our best performing method, the R+LFIM, solves a TSP on a modified graph, which has a low complexity in terms of interpretability. Our framing of the problem keeps the granularity to the features, which is easily interpretable, and intuitively understandable, because they represent the transition likelihood between the nodes. The penalty factors used in the transition matrix can even be changed manually to encourage or discourage particular visiting orders.

Flexibility: Our best performing method, the R+LFIM, is flexible due to ease of incorporating unseen data, and this requires no new training efforts. New nodes (customers), new features, or new feature levels for existing features can seamlessly be incorporated by explicitly interpreting their impacts. The RFIM method allows us to have an initial baseline of relations directly without any training. The R+LFIM benefits from the flexibility of the RFIM and additionally improves its performance.

7. Conclusion

We have built an effective, interpretable, and flexible framework for learning the experiences of the drivers. To this end, we have introduced, modeled and solved the Data-driven Traveling Salesman Problem (DD-TSP), which involves minimizing a potentially nonlinear and nonsmooth function. The aim of the first of the two stages is to modify the input weight matrix for the TSP, which corresponds to the transition discouragement levels according to driver experiences. In the context of our paper, we refer to this matrix as the *transition weight matrix*, which is then given to a TSP solver in the second stage to generate routes that are similar to the executed routes. We assume that this matrix is factorizable; that is, it can be expressed as an element-wise multiplication of different matrices that represent *features*. Each such factor matrix captures the impacts of a feature on the routes, and we leverage the side information in the data to learn about these matrices.

We have developed three methods for learning the transition weight matrix. The first one, Score Guided Coordinate Search (SGCS), is an extension of the coordinate search algorithm in derivative-free optimization. The second method is for learning the transitions directly from the data, which we have referred to as the Label-guided Feature Impact Matrix. The third method combines the benefits of a rule-based method with the data-driven approach, which is referred to as Rule-based Label-guided Feature Impact Matrix. We have tested the efficiency of these methods using a case study based on Amazon Last-Mile Routing Challenge. Results have shown that we have improved on our previous results, which achieved a third place in the competition, by 23.1% in an out-of-sample dataset. Furthermore, our best performing algorithm has improved the best score in the literature achieved on the available training dataset by 40.1%.

Acknowledgments

The authors gratefully acknowledge funding provided by the Natural Sciences and Engineering Research Council of Canada under grant RGPIN-2022-04979.

Appendix A: Descriptive Analytics

We now present a descriptive analysis of the data. We explore the travel times, the customer zones, and the first and last nodes on the routes in Sections A.1, A.2 and A.3, respectively, in order to extract and select important features.

A.1. Travel times

When the objective is defined as the minimization of the tour duration, the problem is a classical TSP. Using only the travel times, solving the corresponding TSP for every route and evaluating them in the scoring function presented in Section 4.2 gives a competitive score of 0.0795. Note that this score achieves the 16th best score among 45 finalist teams in the Amazon challenge studied in the case study. This brief analysis demonstrates that the drivers prefer shorter route durations, and that the travel time is an important measure in route selection.

A.2. Zones

Zoning is a critical factor in delivery operations. The service region is usually partitioned into several zones considering multiple factors such as the geographical and structural differences between locations. In the 6112 routes in the case study, there are an average of 21.04 different zones visited per route including the depot. The average number of times that the drivers change zones between two consecutive customer visits is only 23.61. In other words, the drivers generally visit the nodes in the same zone before switching to another one. In the actual routes, the zone IDs of two consecutive nodes are the same 85.2% of the times. A sample route is plotted in Figure 3. The pattern of visiting customers in the same zone before switching to the next one can clearly be observed in this example. The observation that the drivers generally visit the customers in the same zone before going into another one is an important observation. But the zone ID property contains more information. In particular, the more similar two zone names are, the closer they are geographically to each other, similar to Universal Transverse Mercator (UTM) grid reference system (Stott 1977). In the dataset, the '*zone ID*' property of a node is of the following format: $L\text{-}M.PR$, where L and R are letters and M and P are numbers. We refer to each of these four entries as a *token*. Each token breaks the service region into sub regions. Let $x_{ij}^k = 1$ if the kth token of the *zone ID* of nodes i and j are different, and 0 otherwise.

Definition 2. Given two nodes $i, j \in N$, we define $\eta_{ij} = \sum_{k=1}^{k=4} x_{ij}^k$ to be the 'token difference of nodes i and j', which is a parameter to measure the dissimilarity between the *zone IDs* of the two given nodes.

Having $\eta_{ij} = 0$ implies that nodes i and j share the same zone ID. Table 11 shows the average number of times that a token difference is observed on all 6112 routes. Two consecutively visited nodes are in the same region 124.38 times, on average. This corresponds to 85.2% of the average number of nodes. Note that the drivers very rarely change zones with $\eta_{ij} \geq 2$. Majority of these zone changes are between zones with a token difference of 1. For example, the vehicle visits a node in zone *A-1.2A* and continues with another node in zone *A-1.3A*. In other words, the consecutive visited zones share 3 tokens 17.38 times on average, which corresponds to 11.9% of the average number of nodes. This case is particularly important and we now further investigate such zone changes.

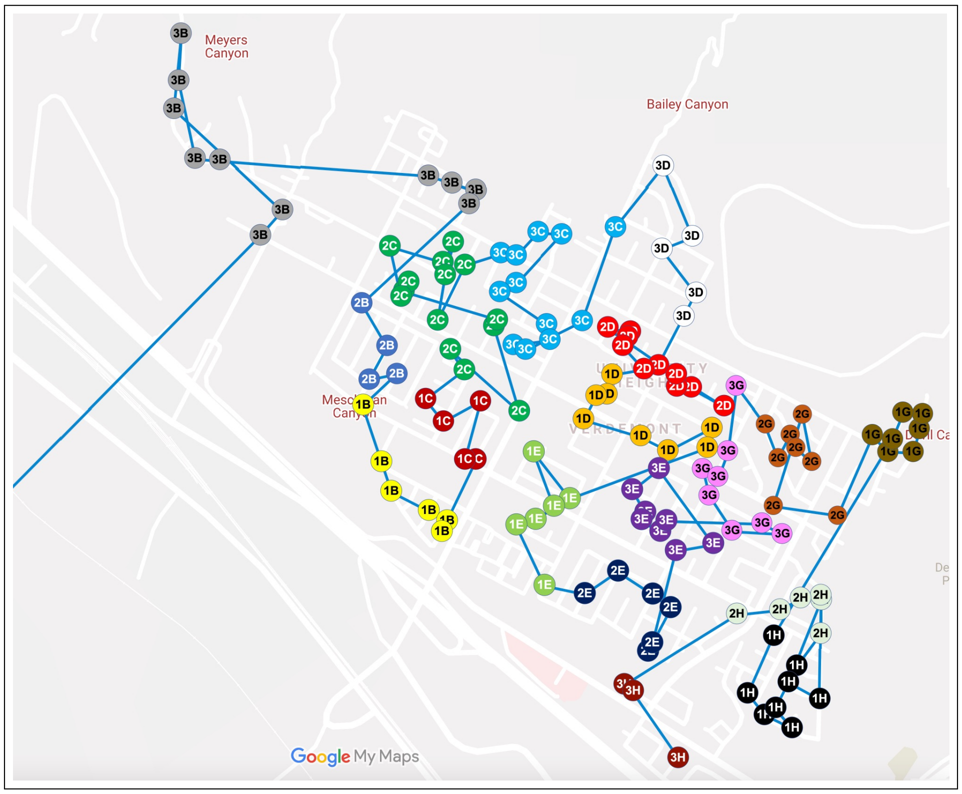


Figure 3 **Map of actual route 'RouteID_62955f5e-57a2-4425-bfe6-6989d8dae565'. Nodes are marked by their zone IDs and the route is shown in blue line. Note that only the last two tokens of zone IDs are plotted on the map for ease of display. The depot is not shown on this map. Google Map data ©2021**

Table 11 **Average number of times that a token difference appears between two consecutive nodes on actual routes**

Token difference (η_{ij})	Average number of times per route
0	124.38
1	17.38
2	1.88
3	0.34
4[†]	2.02
Total[†]	145.99

[†] Excluding the arcs from the depot

When the token difference equals 1 in Table 11, the two zone IDs are different only in i^{th} token for $i = 1, \ldots, 4$. Table 12 shows the average number of times that i^{th} token is different in consecutive nodes of the actual routes, for $i = 1, \ldots, 4$. Among the 17.38 zone changes with single token difference, the vehicle visits 11.33 times another zone that changes only in the 3rd token (Table 12). For example, the vehicle goes from zone A1.1A to A1.2A. They go to another zone that changes only in the 4^{th} token 5.22 times. Therefore, in the actual routes, the zone IDs of two consecutive nodes change only in the 3^{rd} or 4^{th} token 11.3% of the times.

Table 12 **Average number of times that two zone IDs are different only in i^{th} token for $i = 1, \ldots, 4$ (when the token difference equals 1 in Table 11).**

Token #	Average number
1^{st}	0.02
2^{nd}	0.81
3^{rd}	11.33
4^{th}	5.22
Total	17.38

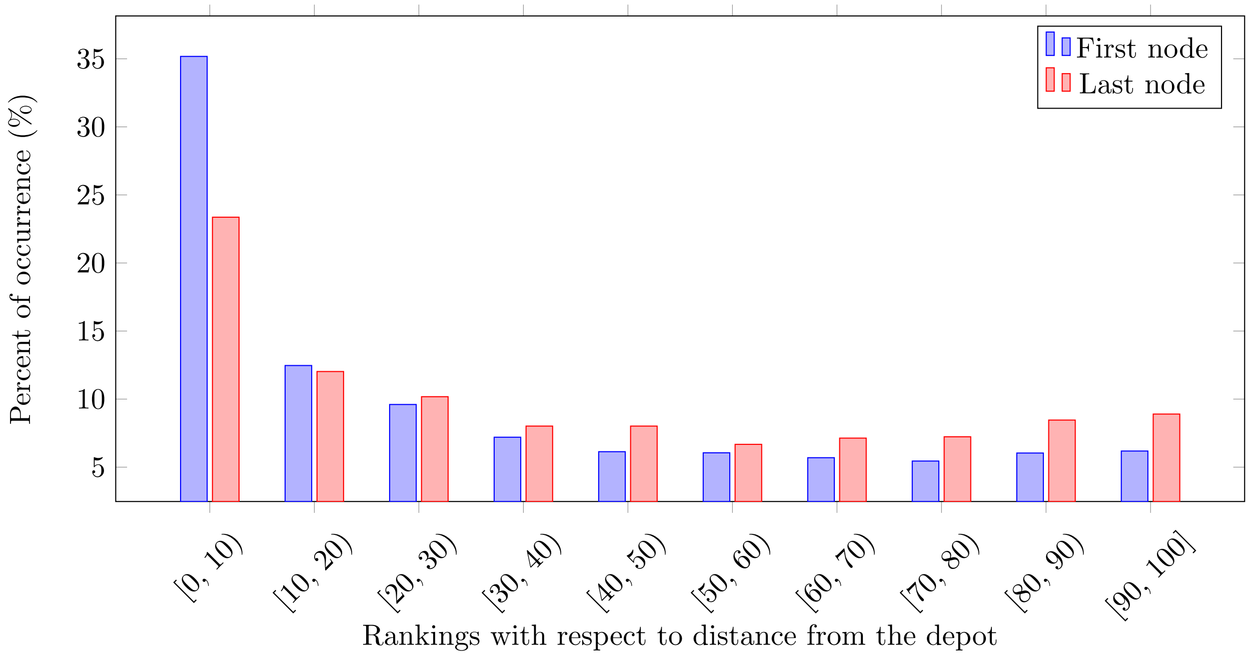


Figure 4 **The first node and the last nodes on the actual routes are not always the nearest two nodes to the depot. This plot shows the ranking distribution of the first and last nodes in terms of distance from the depot.**

A.3. The first and the last nodes on the routes

The average travel time from the depot to the first node is 1811.4 seconds, the travel time from the last node to the depot is 1857.7 seconds and the travel time travel time spent between customers is 8972.8 seconds. Therefore, 29% of the route time, the driver is enroute from and to the depot, which signifies the importance of these two trips. We therefore further investigate the first and the last nodes on the route from the depot. In particular, we sort the nodes with respect to their distance from the depot, and plot the distribution of the ranks of first and the last nodes in Figure 4. The first node on the route is in top 10% ranking in travel time from the depot in 35% of the routes. Similarly, the last node on the route is in top 10% ranking in travel time from the depot in 23% of the routes. As shown in the graph, there is a high probability that those nodes that are farther away from the depot can be selected as the first and the last nodes. Therefore, we conclude that, in the actual routes, the first and the last nodes are not necessarily close to the depot.

Appendix B: Tuning the Algorithm Hyperparameters

In this section, we present details on tuning the hyperparameters of the RFIM, SGCS and LFIM methods.

B.1. Tuning hyperparameters for RFIM

We randomly generated 494 (a_1, a_2, and a_3 values respecting $a_1 < a_2 < 10a_1$ and $a_2 < a_3 < 10a_2$ conditions and tested our algorithm. None of these implementations performed better than the baseline settings with $(a_1, a_2, a_3) = (1, 10, 100)$, in which we achieve a score of 0.0367. The average score of these 494 experiments is 0.0396 and the maximum score is 0.0543, demonstrating the robustness of RFIM method for different hyperparameter settings. The b_1 and b_2 hyperparameters, on the other hand, determine the importance of the trips from and to the depot. We tested $(0, 0)$, $(1, 0)$, $(0, 1)$, $(1, 1)$ and $(0.5, 0)$ settings with $(a_1, a_2, a_3) = (1, 10, 100)$. The average scores are 0.0381, 0.0367, 0.0378, 0.0374 and 0.0371, respectively. Therefore, we set $(b_1, b_2) = (1, 0)$ as in the baseline settings.

B.2. Tuning hyperparameters for SGCS

The hyperparameters for *SGCS* algorithm are the iteration count (*itCount*), the maximum number of trials at every iteration (Γ), the number of unique elements changed (ε), and the step size (Δ). We conducted experiments using depot DLA9, which is the largest depot in terms of the High-SV routes. The experimental design included all combinations of $itCount \in \{5, 10\}$, $\Gamma \in \{3, 5, 10\}$, $\varepsilon \in \{0.01, 0.05, 0.10\}$ and $\Delta \in \{10, 50, 100\}$. The setting that performed the best is $itCount = 10$, $\Gamma = 5$, $\varepsilon = 0.05$, and $\Delta = 50$ with a 1.52% improvement on the training dataset and a 1.84% improvement on the testing dataset. This setting is implemented in the reported results in this paper.

B.3. Tuning hyperparameters for R+LFIM

Recall that, when building the LFIM, $\kappa \in \mathbb{N}$ in equation 6 represents the maximum number of transitions that can change the weight value. We have tested $\kappa \in \{2, \dots, 10\}$ on the entire dataset. The impact on the score changes minimally between different κ values. We have selected and used $\kappa = 5$ for all the experiments carried out in this paper, which gave an overall score of 0.0403 on the training routes, and 0.0382 on the testing routes.

References

Accorsi L, Lodi A, Vigo D, 2022 *Guidelines for the computational testing of machine learning approaches to vehicle routing problems. Operations Research Letters* 50(2):229–234.

Amazon, 2021 *Amazon Last Mile Routing Research Challenge (Supported by the MIT Center for Transportation & Logistics).* URL `https://routingchallenge.mit.edu`, last accessed on Jun 18, 2021.

Applegate DL, Bixby RE, Chvátal V, Cook WJ, 2011 *The traveling salesman problem* (Princeton university press, New Jersey).

Arslan O, Abay R, 2021 *Data-driven Vehicle Routing in Last Mile Delivery* (Report CIRRELT-2021-30, Université de Montreal).

Audet C, Hare W, 2017 *Derivative-Free and Blackbox Optimization* (Springer Cham, Switzerland).

Bello I, Pham H, Le QV, Norouzi M, Bengio S, 2016 *Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940* .

Bresson X, Laurent T, 2021 *The transformer network for the traveling salesman problem. arXiv preprint arXiv:2103.03012* .

Canoy R, Bucarey V, Mandi J, Guns T, 2023 *Learn and route: learning implicit preferences for vehicle routing. Constraints* 28(3):363–396.

Canoy R, Bucarey V, Mandi J, Mulamba M, Molenbruch Y, Guns T, 2024 *Probability estimation and structured output prediction for learning preferences in last mile delivery. Computers & Industrial Engineering* 189:109932.

Carrabs F, Cordeau JF, Laporte G, 2007 *Variable neighborhood search for the pickup and delivery traveling salesman problem with lifo loading. INFORMS Journal on Computing* 19(4):618–632.

Cook W, Held S, Helsgaun K, 2024 *Constrained local search for last-mile routing. Transportation Science* 58(1):12–26.

Delarue A, Anderson R, Tjandraatmadja C, 2020 *Reinforcement learning with combinatorial actions: An application to vehicle routing. Advances in Neural Information Processing Systems* 33:609–620.

Deudon M, Cournut P, Lacoste A, Adulyasak Y, Rousseau LM, 2018 *Learning heuristics for the TSP by policy gradient. International conference on the integration of constraint programming, artificial intelligence, and operations research*, 170–181 (Springer).

Ghosh M, Kuiper A, Mahes R, Maragno D, 2023 *Learn global and optimize local: A data-driven methodology for last-mile routing. Computers & Operations Research* 159:106312.

Google, 2022 *Or-tools.* `https://github.com/google/or-tools`.

Helsgaun K, 2000 *An effective implementation of the Lin–Kernighan traveling salesman heuristic. European Journal of Operational Research* 126(1):106–130.

Helsgaun K, 2017 *An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. Roskilde: Roskilde University* 12:966–980.

Hore S, Chatterjee A, Dewanji A, 2018 *Improving variable neighborhood search to solve the traveling salesman problem. Applied Soft Computing* 68:83–91.

Hudson B, Li Q, Malencia M, Prorok A, 2021 *Graph neural network guided local search for the traveling salesperson problem. arXiv preprint arXiv:2110.05291* .

Junior Mele U, Maria Gambardella L, Montemanni R, 2021 *Machine learning approaches for the traveling salesman problem: A survey. 2021 The 8th International Conference on Industrial Engineering and Applications (Europe)*, 182–186.

Keld Helsgaun, 2022 *LKH solver.* `http://webhotel4.ruc.dk/~keld/research/LKH/`.

Knox J, 1994 *Tabu search performance on the symmetric traveling salesman problem. Computers & Operations Research* 21(8):867–876.

Kotthoff L, Kerschke P, Hoos H, Trautmann H, 2015 *Improving the state of the art in inexact tsp solving using per-instance algorithm selection.* Dhaenens C, Jourdan L, Marmion ME, eds., *Learning and Intelligent Optimization*, 202–217 (Springer Cham, Heidelberg).

Laporte G, 1992 *The traveling salesman problem: An overview of exact and approximate algorithms. European Journal of Operational Research* 59(2):231–247.

Lin S, Kernighan BW, 1973 *An effective heuristic algorithm for the traveling-salesman problem. Operations Research* 21(2):498–516.

Merchan D, Pachon J, Arora J, Konduri K, Winkenbach M, Parks S, Noszek J, 2021 *Amazon last mile routing research challenge dataset.* URL `https://registry.opendata.aws/amazon-last-mile-challenges`, accessed January 6, 2022.

Miki S, Yamamoto D, Ebara H, 2018 *Applying deep learning and reinforcement learning to traveling salesman problem. 2018 international conference on computing, electronics & communications engineering (ICCECE)*, 65–70 (IEEE).

Mo B, Wang Q, Guo X, Winkenbach M, Zhao J, 2023 *Predicting drivers' route trajectories in last-mile delivery using a pair-wise attention-based pointer neural network. Transportation Research Part E: Logistics and Transportation Review* 175:103168.

Montgomery DC, Peck EA, Vining GG, 2021 *Introduction to linear regression analysis* (John Wiley & Sons, Hoboken, USA).

Nagata Y, Kobayashi S, 2013 *A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS Journal on Computing* 25(2):346–363.

Nazari M, Oroojlooy A, Snyder L, Takác M, 2018 *Reinforcement learning for solving the vehicle routing problem. Advances in neural information processing systems* 31.

Nelder JA, Mead R, 1965 *A Simplex Method for Function Minimization. The Computer Journal* 7(4):308–313.

Özarık SS, da Costa P, Florio AM, 2024 *Machine learning for data-driven last-mile delivery optimization. Transportation Science* 58(1):27–44.

Parmentier A, 2022 *Learning to approximate industrial problems by operations research classic problems. Operations Research* 70(1):606–623.

Pepper JW, Golden BL, Wasil EA, 2002 *Solving the traveling salesman problem with annealing-based heuristics: a computational study. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 32(1):72–77.

Potvin JY, 1996 *Genetic algorithms for the traveling salesman problem. Annals of Operations Research* 63(3):337–370.

Ristad ES, Yianilos PN, 1998 *Learning string-edit distance. IEEE Transactions on Pattern Analysis and Machine Intelligence* 20(5):522–532.

Sadana U, Chenreddy A, Delage E, Forel A, Frejinger E, Vidal T, 2024 *A survey of contextual optimization methods for decision-making under uncertainty. European Journal of Operational Research* .

Scroccaro PZ, van Beek P, Esfahani PM, Atasoy B, 2023 *Inverse optimization for routing problems. arXiv preprint arXiv:2307.07357* .

Stott PH, 1977 *The utm grid reference system. IA. The Journal of the Society for Industrial Archeology* 1–14.

Sultana N, Chan J, Sarwar T, Abbasi B, Qin A, 2021 *Learning enhanced optimisation for routing problems. arXiv preprint arXiv:2109.08345* .

Varol T, Özener OÖ, Albey E, 2024 *Neural network estimators for optimal tour lengths of traveling salesperson problem instances with arbitrary node distributions. Transportation Science* 58(1):45–66.

Wall ME, Rechtsteiner A, Rocha LM, 2003 *Singular value decomposition and principal component analysis. A practical approach to microarray data analysis*, 91–109 (Springer, Boston).

Wang Z, Geng X, Shao Z, 2009 *An effective simulated annealing algorithm for solving the traveling salesman problem. Journal of Computational and Theoretical Nanoscience* 6(7):1680–1686.

Winkenbach M, Parks S, Noszek J, 2021 *Technical proceedings of the Amazon last mile routing research challenge* URL `https://hdl.handle.net/1721.1/131235`.

Xin L, Song W, Cao Z, Zhang J, 2021 *NeuroLKH: Combining deep learning model with Lin-Kernighan-Helsgaun heuristic for solving the traveling salesman problem. Advances in Neural Information Processing Systems* 34.

Zheng J, He K, Zhou J, Jin Y, Li CM, 2021 *Combining reinforcement learning with Lin-Kernighan-Helsgaun algorithm for the traveling salesman problem. Proceedings of the AAAI conference on artificial intelligence*, volume 35, 12445–12452.