Amazon Last-Mile Routing Research Challenge
Supported by the MIT Center for Transportation & Logistics

Technical Proceedings
Edited by Matthias Winkenbach, Steven Parks and Joseph Noszek

September 2021
URI: https://hdl.handle.net/1721.1/131235

Disclaimer
The responsibility for the contents and opinions expressed in the articles presented in this publication rests solely with their authors, and this publication does not constitute an endorsement by the Massachusetts Institute of Technology or Amazon of these contents or opinions.

Official website of the research challenge:
routingchallenge.mit.edu
# Table of Contents

I  Drivers' Intuition for the Traveling Salesman Problem  
   *Peter Dieter*

II  Last-Mile Delivery Trajectory Prediction Using Hierarchical TSP with Customized Cost Matrix  
   *Xiaotong Guo, Baichuan Mo and Qingyi Wang*

III  A Short-Sighted Machine Learning Implementation on the Vehicle Routing Problem  
   *Kelvin Anthony and Kenny Anthony*

IV  Route Sequence Prediction Through Inverse Reinforcement Learning  
   *Anselmo R. Pitombeira-Neto*

V  A Two-stage Traveling Salesman Model for the Last-Mile Routing Problem  
   *Qiu Runwen, Wang Yueting and Xie Yi*

VI  Data-Driven Vehicle Routing in Last-Mile Delivery  
   *Okan Arslan and Rasit Abay*

VII  A Hybrid Method with Rules and Optimization to Solve the Last-Mile Delivery Problem  
    *Fanyou Wu and Yang Liu*

VIII  A Data-Driven Metaheuristic Approach to Predict Delivery Route Sequences  
    *Juan Pablo Mesa, Alejandro Montoya and Raúl Ramos-Pollán*

IX  Delivery Location Dependent Traveling Salesman Problem Using Multi-Head Attention-Based Actor-Critic Method  
    *Ashutosh Nayak and Ahmad Hemmati*

X  Enhancement of Routing Solver by Learning from Good Routes  
    *Long He, Yiduo Huang and Tian Wang*

XI  Using Graph Neural Networks to Solve Route Sequencing Problems  
    *Andres Regal*

XII  A Machine Learning Framework for Last-Mile Delivery Optimization  
    *Alexandre M. Florio, Paulo da Costa and Sami Serkan Özarık*

XIII  A Deep Bi-directional Long-Short Term Memory Neural Network-Based Solution for the Vehicle Sequencing Problem  
    *Amr Eltawil and Sara Atef*
| XII | Learning Routing Models with Cluster Precedence Constraints  
Ivan Contreras and Dang Le |
| --- | --- |
| XIII | Distance Matrix Adjustment for Routing Problems  
Mayukh Ghosh, A.V. Mahes and Donato Maragno |
| XIV | Tabu Search-Based Two-Stage Stop Sequence Optimization with Zone-Travel Weight Information Learning from Historical Route Data  
Zhongyuan Lyu and Xuewu Chen |
| XV | Inverse Reinforcement Learning for Learning Driver Utility Function for Package Delivery Routing Problems  
Jianhan Song, Ashutosh Shukla and Josiah Coad |
| XVI | A Perspective on Driver’s Preferences for Route Planning  
Shanshan Huang, Rayner Rebello and Qiuyu Zhu |
| XVII | Simple Heuristic Algorithm to Solve the TSP With Spatial Structure and Time Window Adherence  
Ha Dao, Konstantin Geier and Bernhard Hilfer |
| XVIII | Combining Operational Research and Machine Learning to Solve Vehicle Routing Problems  
Terence Ngo, Rathea Uth and Romain Loirs |
| XIX | Local Search with Learned Constraints for Last Mile Routing  
William Cook, Stephan Held and Keld Helsgaun |
| XX | Combining Naive Bayes Model and TSP Routing Algorithm for Predicting High Quality Route Sequences  
Wenbin Zhu, Ying Fu and Luo Zhixin |
| XXI | A Hybrid Route Planning Heuristic Based on Simulated Annealing and Iterated Local Search  
Fellipe Pessanha and Pedro Vasconcellos |
| XXII | A Frequency-Based Routing and Scheduling Engine  
António Ramos and Manuel Lopes |
| XXIII | Learning to Solve the Vehicle Routing Problem  
Xinyang Yuan and Ji Gao |
| XXIV | Hybrid Adaptive Large Neighborhood Search for the Traveling Salesman Problem with Time Windows and Adjusted Costs  
Stefan Voigt |
| XXV | A Two-Stage Approach to Routing with Driver Preferences via Heatmaps  
Connor Lawless, Sotiris Ntanavaras and Anders Wikum |
| XXVI | Imitating a Variant of the Traveling Salesman Problem  
Venktesh Pandey, Tarun Rambha and Neha Singh |
| XXVII | Transformer Model with Graph Convolution Embeddings for Route Planning  
Rashmi Kethireddy |
XXX TSP with Learned Zone Preferences for Last-Mile Vehicle Dispatching
Rocsildes Canoy, Jayanta Mandi and Victor Bucarey

XXXI Combined Exact and Heuristics Based Approach to Hamiltonian Path Problem Optimization for Route Planning
Fernando Hernandez, Rafael Sotelo and Marcelo Forets
Article I

Drivers' Intuition for the Traveling Salesman Problem

Peter Dieter

Source code is available on GitHub:
https://github.com/PeterDieter/AmazonChallenge
Article I

Drivers’ intuition for the traveling salesman problem

Peter Dieter

Department of Management Information Systems and Operations Research, University of Paderborn
Warburger Str. 100, 33098 Paderborn, Germany

Abstract

We develop an optimization algorithm for the traveling salesman problem (TSP), which does not solely consider optimizing an objective value, but also incorporates drivers’ intuition. The algorithm consists of a constructive heuristic and an improvement heuristic. We do this by considering ‘zone centrality’ which measures how close a zone is to other unvisited zones. Drivers prefer less central zones when they make the decision to which zone to drive next. The algorithm results in scores of approximately 0.049 when tested on around 2,000 not seen instances which were classified as ‘high quality’. It outperforms algorithms that solely minimize the total route duration. The ‘zone centrality’ is therefore a relevant factor in drivers’ intuition.

1. Introduction

In the last decades, many optimization algorithms have been proposed for the TSP that solely optimize an objective value but neglect other, not easily quantifiable aspects of a route, that determine the quality of a solution. One of these aspects is drivers’ intuition We propose an algorithm that does not solely consider an objective value, but also incorporates drivers’ intuition. The algorithm consists of a constructive and improvement heuristic. Before the algorithm starts, the distance-matrix is manipulated by incorporating the air-line distance to better account for drivers’ intuition. In the constructive heuristic, so-called zones, which are an Amazon-internal construct used to subdivide stops into sub-areas, are placed. We do this by not only considering the distance between the zones but also considering the ‘zone centrality’ which measures how close a zone is to other unvisited zones. It was found that drivers prefer less central zones when they make the decision to which zone to drive next. The stops within a zone are placed by a simple nearest neighbor strategy and then ‘reordered’ in the improvement phase of the algorithm. The placement of zones largely determines the sequence similarity (score) and the scope of drivers’ choices is very limited once the zone sequence is determined. Therefore, the ‘reordering’ of stops within zones is solely done by minimizing the total travel time. The drivers’ intuition is therefore not considered at this step anymore. We use Simulated
annealing as an improvement heuristic. The order of the zones is not changed at this step anymore. This drastically reduces the solution space.

In terms of sequence similarity to historical realized routes of Amazon last-mile delivery drivers, the proposed algorithm outperforms approaches that are solely based on the objective value. The structure of this report is as follows: In section 2, previous literature on the traveling salesman problem (TSP) is reviewed. The proposed algorithm is described in section 3. Chapter 4 covers the results of the algorithm and possible extensions for further research.

2. Literature Review

Many constructive heuristics for the TSP have been suggested in the literature. These approaches range from simplistic methods such as the nearest neighbor algorithm to more sophisticated approaches like the savings algorithm (Clarke & Wright, 1964) or the Christofides algorithm (Christofides, 1976). Similarly, many improvement heuristics and exact methods have been developed to solve the TSP. Popular approaches are the Tabu-search which was first introduced by Glover & McMillan (1986) for scheduling optimization or the Concorde solver (Applegate et al., 2002) which uses a branch-and-cut algorithm to solve the TSP exactly.

While most of these sophisticated approaches guarantee a certain solution quality in terms of route length, they do not guarantee that drivers also accept following these routes. Drivers often deviate from a planned route due to their intuition. Anticipating this intuition can potentially help to plan routes that drivers accept and to improve arrival time estimations. This work addresses this shortcoming by incorporating drivers’ intuition.

3. Methodology

The structure of this section is as follows: Firstly, the travel time manipulation is explained. Secondly, the constructive heuristic is presented. This includes the introduction of zone distances, zone centrality, and finally the zone placement. Thirdly, the improvement heuristic is explained.

3.1. Travel Time Manipulation

Before the other operations start, we manipulate the travel time matrix by including the geographical distance. This is done because we expect drivers to intuitively also consider the air distance and not solely the travel time. The newly created travel time between node \( a \) and \( b \) is given by:

\[
TT_{ab}^{\text{new}} = TT_{ab}^{\text{old}} + 180 \times D_{ab}^{\text{geo}}
\]

where \( TT_{ab}^{\text{old}} \) is the original travel time matrix in seconds provided by Amazon and \( D_{ab}^{\text{geo}} \) is the geographical (haversine) distance matrix measured in kilometers. Although the elements of the new matrix have km*seconds as a unit, the matrix is further referred to as the travel time matrix.
3.2. Constructive Heuristic

Before the constructive algorithm is explained, we introduce the ‘zone distance’ and ‘zone centrality’.

3.2.1. Zone Distance Calculation

In order to place the zones in the constructive heuristic, the distance between zones needs to be determined. The distance between zone $X$ to zone $Y$ is the average of all their directed subdistances:

$$ TT_{XY}^{Zone} = \frac{1}{|X||Y|} \sum_{i=1}^{|X|} \sum_{j=1}^{|Y|} TT_{ij}^{new} $$

where $i \in X$ and $j \in Y$.

3.2.2. Centrality

An important measurement that is used to place the zones is the so-called ‘zone centrality’. This measure describes how central an unvisited zone is compared to other unvisited zones, as visualized in Figure 1. The triangle corresponds to the depot, the squares to the stops, and the circles to the zones.

When manually analyzing the data, it was found out that drivers prefer less central zones when they make the decision to which zone to drive next. The centrality $C$ of an unvisited zone $i$ is computed as follows:

$$ C_i = - \frac{1}{|U \setminus i|} \sum_{z \in |U \setminus i|} TT_{zi}^{Zone} $$

where $U$ is the set of unvisited zones. Please note that a high average distance to other zones results in a low centrality. Therefore, the sign of the term is reversed.

3.2.3. Zone Placement

The first zone is the zone with the lowest sum of the centrality plus 0.2 times the distance from the depot. This value has been tuned manually by ‘trial and error’. It is therefore predicted that the first zone
is less central and relatively close to the depot. Afterwards, the following zones are added with a similar logic. Instead of adding 0.2 times the distance from the depot to the centrality, the distance from the current zone is added with a weight of 1. The stops within a zone are added by the nearest neighbor logic. The pseudo-code of the algorithm is shown in algorithm 1.

### Algorithm 1: Zone Placement

**Result:** sequence

```python
sequence.append('Depot');
currentZone = min(Centrality + 0.2*DistanceToDepot);
sequence.append(Stops in currentZone with nearest neighbor);

while Not all zones visited do
    nextZone = min(Centrality + DistanceToCurrentzone);
    sequence.append(Stops in nextZone with nearest neighbor);
    currentZone = nextZone
end
```

3.3. Improvement algorithm

The second stage of the algorithm involves reordering stops within a zone. The order of the zones is not changed anymore. This drastically reduces the solution space. The algorithm that we use to reorder the stops is Simulated annealing as introduced by [Kirkpatrick et al.](1983). This approach is chosen because it allows for a controlled speed of convergence towards an optimum. The constructive heuristic is used as start solution and a worse solution is allowed as an intermediate solution with a certain probability. An improved intermediate solution is always accepted. The best solution found so far is stored. The algorithm stops after 3 seconds. A negative exponential cooling scheme of the following form is used:

$$T_k = e^{-kc}$$  \hspace{1cm} (4)

Where $T_k$ is the temperature at iteration $k$ and $c$ is a cooling factor which is set to 0.003. The cooling rate is not tuned. The algorithm is parallelized on every core minus one of the CPU. The total travel time is the objective function that is minimized.

3.3.1. Neighborhood Structure

The following three neighborhoods are used. The probability of choosing each of these neighborhoods is equal.

**Node Insertion**

In the node insertion neighborhood, a random node is inserted to a different random position in the same zone.

$$0-1-2-3 \rightarrow 0-2-1-3$$

_Dieter_
Node-Node Exchange
In the node-node exchange neighborhood, two nodes within a zone are swapped randomly.

\[0-1-2-3 \rightarrow 0-3-2-1\]

Two-opt swap
In the two-opt swap neighborhood, two arcs within a zone are swapped randomly.

\[0-1-2-3-4 \rightarrow 0-3-4-1-2\]

4. Results and Conclusions

We test the algorithm solely on routes, which are classified as ‘high quality’ routes. Even though the approach does not include any types of machine learning, the algorithm is constructed with the knowledge of approximately 500 routes, on which we also calibrated hyper-parameters. These 500 routes are therefore not included in the evaluation phase, to prevent any kind of overfitting. The average score on the test data is 0.04935 with a standard deviation of 0.0356. The distribution of the scores is presented in figure 2.

![Score distribution](image)

Figure 2: Score distribution

It is visible that the distribution is right-skewed, which is because the score is bound at 0. The highest score is 0.3495 and the lowest 0.00129.

By testing different route attributes, we found that the average distance between the stops and the depot has a large influence on the performance of the model. The algorithm works considerably better on routes where the depot is further away from the stops. More precisely, the Pearson correlation coefficient of the two variables is -0.385. The score of routes that have an average distance between the stops and the depot...
above the median is 0.0375, while routes below the median have an average score of 0.0613.

Multiple potential further model developments might be worth investigating. For example, more data-driven approaches might be applied. Parameters of the algorithm could be adjusted based on the characteristics of the route. An interesting extension of machine learning would be to enhance the prediction of the first zone in the drivers’ route sequence, as determining a ‘good’ first zone has a great influence on the score.

In terms of sequence similarity to historical realized routes of Amazon last-mile delivery drivers, the proposed algorithm outperforms approaches that are solely based on the objective value. The ‘zone centrality’ is therefore a relevant factor in drivers’ intuition.
References


Article II

Last-Mile Delivery Trajectory Prediction Using Hierarchical TSP with Customized Cost Matrix

Xiaotong Guo, Baichuan Mo and Qingyi Wang

Source code is available on GitHub:
https://github.com/XiaotongGuoMIT/Permission_Denied.git
Article II

Last-Mile Delivery Trajectory Prediction Using Hierarchical TSP with
Customized Cost Matrix

Xiaotong Guo, Baichuan Mo, Qingyi Wang

Department of Civil and Environmental Engineering, Massachusetts Institute of Technology
77 Massachusetts Ave, Cambridge, MA, USA

Abstract

In response to the Amazon Last-Mile Routing Challenge, Team Permission Denied proposes a hierarchical Travelling Salesman Problem (TSP) optimization with a customized cost matrix. The higher level TSP solves for the zone sequence while the lower level TSP solves the intra-zonal stop sequence. The cost matrix is modified to account for routing patterns beyond the shortest travel time. Lastly, some post-processing is done to edit the sequence to match commonly observed routing patterns, such as when travel times are similar, drivers usually start with stops with more packages than those with fewer packages. The model is tested on 1223 routes that are randomly selected out of the training set (including routes of all qualities), and the score is 0.0381. On the 13 routes in the given model apply set, the score was 0.0375.

1. Introduction

This report presents the thought processes, selected methodology, and expected results of the Amazon Last-Mile Routing Research Challenge by Team Permission Denied. In summary, the team went through four phases before arriving at the final submission.

Descriptive Analysis: Upon receiving the challenge, a thorough descriptive analysis is done. The first important finding is that, in most circumstances, the drivers finish all deliveries in one zone before moving on to the stops in another zone. This rule is only broken when backtracking exists. A further look at the scores confirms this intuition: assuming the zone sequence and intra-zonal stop sequence are correct, the loss on the score due to certain zones being revisited is only 0.009. If the zone sequence is correct and the stops in each zone are shuffled, the average score is around 0.02. Therefore, getting the zone sequence correct is the most important, and the team decides to adopt a hierarchical approach: solving for the zone sequence, and then the intra-zonal stop sequence. This greatly reduces the scale of the problem since the majority of the routes have around 150 stops (up to 250), but the number of zones is between 6 and 47. Second, the zonal
transitional probabilities are investigated. As most of the zones only appear in the training set once, an attempt at a frequency tabulation is not successful. On the other hand, 74% of the zonal transitions select the zone that is closest by travel time, making the step-by-step prediction algorithm potentially successful. Next, the correlation between package dimensions, package counts, delivery time windows, and sequence order is investigated but no apparent relationship is found.

**Benchmarking:** A benchmark model is created to establish an idea of the solution quality and expected performance. Since most drivers follow the navigation given by Amazon, a shortest-distance tour becomes a natural benchmark. The team solves a tour-based (where the start and end stations are both INIT) to generate zone sequences and a path-based (where the distance from the last zone to INIT is not counted) Travelling Salesman Problem (TSP) to generate intra-zonal stop sequences as benchmarks. Inside each zone, a path-based TSP is generated from the stop closest to the last zone to the stop closest to the next zone.

**Model Attempts:** Both naive TSP solutions achieve scores reasonable scores (around 0.06). To improve the performance, machine learning models are attempted. First, it is noticed that correctly predicting the first zone would significantly improve the TSP performance, therefore a neural network is constructed to predict the first zone based on the travel time, distance, package count and size, etc. Second, pure machine learning models to generate sequences are investigated, including myopic approaches that predict the next element based on previously predicted stops, as well as sequence-to-sequence (seq2seq) approaches that encode and decode the entire sequence. Third, different training methods are considered, including the traditional cross-entropy loss, customized weighted loss, as well as reinforcement learning using policy gradients. Lastly, some improvements are made to the benchmark TSP models by adding penalty costs to non-consecutive zone-ids. Due to the small sample size (6k), machine learning techniques cannot outperform the benchmark models. After experimenting with various modeling techniques, the team decides to use the TSP solution as the final submission.

**Hyperparameter Searching and Post-Processing:** The customized cost matrix involves hyperparameters that the team searched for over the given training set. Lastly, some post-processing patterns are identified to further improve the quality of our solution.

The highlights of the final submitted model are:

- Hierarchical modeling - To reduce the size of each optimization problem, the problem is broken down into zone routing and intra-zonal stop routing.

- Customized TSP cost matrix - To account for considerations in addition to shortest distance, the cost matrix is modified and the TSP performance improved by almost 0.01.

- Post-processing to match behavioral patterns - Some TSP sequences are reversed to accommodate delivery patterns such as stops with more packages are visited first instead of last, all else being equal.

- Stable hyperparameters - The cost hyperparameters have good generalizability and do not require re-training.
The rest of the technical report reviews the relevant literature and its compatibility with the research question; describes the selected model in detail, and discusses the expected results.

2. Literature Review

This problem is essentially a vehicle routing problem, except that the traditional setup for vehicle routing problems aims for the shortest distance traveled, but the problem of interest looks for the most similarity with the observed sequence. Two research communities have extensively studied the vehicle routing problem: machine learning and operations research. Literature in both communities is reviewed, with the pros and cons of the algorithms discussed for the problem of interest.

2.1. Operations Research

Given a set of locations one would like to visit, a Traveling Salesman Problem (TSP) can be solved to find the route with the minimum cost or distance. The overview and history of the TSP can be found in Applegate et al. (2011). Although TSP is a well-known NP-hard problem in combinatorial optimization, off-the-shelf integer optimization solvers (e.g., Gurobi and GLPK) are able to solve it efficiently for real-world instances. One key approach we utilized when solving the TSP is the cutting-plane method (Marchand et al., 2002), which is initially applied to TSP by Dantzig et al. (1954).

2.2. Machine Learning

Two types of architectures can be used to re-order the input sequence: step-by-step or sequence-to-sequence (seq2seq). Step-by-step prediction involves predicting the stops one by one, given the information from previous stops, as well as candidate stops. Since the information from candidate stops are crucial, feed-forward neural networks are not a good candidate since it does not attribute features to candidates. Instead, a feed-forward neural network with alternative-specific utility is adopted (Wang et al., 2020). This architecture draws the connection between discrete choice models with neural networks and uses neural networks to generate the utility for each candidate, and the candidate with the highest ‘utility’ is chosen. A sequence is then formed by repeatedly feeding the selected stop into the algorithm to get the next step until the end of the sequence is reached. The advantage of this algorithm is that it is at the stop level instead of the sequence level. Therefore, the sample size, which is critical for the success of machine learning algorithms, is significantly larger than the seq2seq models. The disadvantage of this algorithm is that it is myopic and only sees the next step candidates while making a selection.

In recent years, a lot of seq2seq prediction algorithms have been developed, mainly for natural language processing (NLP) tasks. Compared to step-by-step prediction, seq2seq models comprise an encoder and a decoder. All elements in the sequence are encoded before decoding starts, therefore a global view is attained. The architecture of encoder and decoder often involves variants of the recurrent neural networks (ex. long-short term memory networks) (Sutskever et al., 2014), or attention (Vaswani et al., 2017). Most seq2seq
problems are considered with mapping one sequence to another, whereas the problem of interest is concerned with re-ordering the input sequence. Pointer network is proposed to solve this type of problem, where the decoder uses self-attention to point to one of the input elements (Vinyals et al., 2015). The authors used a pointer network to solve TSP and achieved similar performance to TSP solvers. One drawback of the original pointer network is that it is sensitive to the order of inputs. The authors, therefore, added another encoding module to eliminate this influence (Vinyals et al., 2016). However, in our experiments, this dependency can be leveraged by arranging the input set in a meaningful sequence to improve performance. For example, ordering the input stops according to the TSP sequence would accelerate model convergence and improve the score. However, in the papers presented above, 1M training samples were fed into the network. Given that the training set only contains 6000 routes, score improvements on TSP solutions are unsuccessful.

The original pointer network uses cross-entropy loss (supervised learning). In this problem, the cross-entropy loss is very inefficient due to the way the score is calculated, since the loss only considers the probability of the correct position, and the loss for predicting all other positions is the same. But the scoring function considers similarity in addition to correctness. The scoring function is not differentiable and cannot be directly used as the loss function and use gradient descent. An alternative training method is reinforcement learning based on policy gradients (Ma et al., 2019; Bello et al., 2019). Using the well-known REINFORCE algorithm, we can directly optimize the non-differentiable score function. Researchers have found that this method has the same sample efficiency and better generalizability for TSP problems compared to supervised learning (Joshi et al., 2019). However, training with reinforcement learning in this particular problem with the sample size and given information also does not outperform TSP solutions.

2.3. Proposed Method

Our proposed method is built upon the traditional TSP with a customized distance matrix that implicitly contains drivers’ routing behaviors for the Amazon last-mile delivery. Compared to the existing TSP framework, which minimizes the total vehicle travel distance, we modified the distance matrix and generated optimal routes which minimized the total adjusted travel distance.

3. Methodology

3.1. Data

We observe that most of the drivers tend to visit all stops in a zone before going to the next zone. Hence, we divide the problem into two sub-problems. The first is to identify the zone sequence, and the second is to recognize the intra-zonal stop sequence.

The actual zone sequence is generated based on the order of each zone’s first appearance. An example is shown in Figure 1. For stops without zone id (due to missing data), we fill them with the zone ID of its (travel time-based) nearest stop.

Three important properties are noticed while observing the zone sequences:
• Most likely, the driver would finish a “major zone” first, then move to the next “major zone”. A major zone is defined as the zone ID before the dot. For example, the major zone for “A-2.2A” is “A-2”. For example, in Figure 1, the driver first finishes major zone “A-2”, then “A-1”, finally “P-13”.

• Within a specific major zone, two adjacent “inner zone” ids most likely have a “difference of one”. The “inner zone” is defined as the zone ID after the dot. For example, the inner zone for “A-2.2A” is “2A”. The “difference of one” is defined as follows. Given two inner zone IDs “XY” and “AB”, where X and A are numbers and Y and B are characters, we have

\[|X - A| + |\text{ord}(Y) - \text{ord}(B)| = 1\]  

(1)

where \(\text{ord}(\cdot)\) function returns an integer representing the Unicode character. For example, “1A” and “1B” has a difference of one, so as “1A” and “2A”. But “1A” and “2B” has a difference of two.

• When a driver finishes a “major zone” and move to another, the two adjacent major zone IDs are most likely to have a “difference of one”. For example, in Figure 1, the driver first finishes major zone “A-2”, then “A-1”. Those two major zone IDs have a difference of one.

To validate these three properties, we calculate the frequency that these rules hold in the data set. For all adjacent zone ID pairs, 87.67% of them have the same major zone ID (Property 1). For all adjacent zone ID pairs within a specific major zone, 82.49% of them have a “difference of one” (Property 2). For all adjacent zone ID pairs with major zone ID changes, 96.17% of these changes lead to a “difference of one” between two major zone IDs (Property 3). These statistics support the three properties, which implies that the zone ID includes a lot of information for the sequence estimation.

Another information we use is the planned service time and package volumes. Details on how these are utilized are shown in Section 3.3.3.

We also collected outside data sources from OpenStreetMap. Specifically, we extract the number of traffic signals and highway ramps around every stop. Unfortunately, this does not help to improve our model, thus is dropped from our final submission.

For model validation, we randomly separate the 6,112 routes into a training set (4,889 routes) and a testing set (1,223 routes), though our proposed solution does not require a training process.

3.2. Travelling Salesman Problem Formulation

With the observation that drivers visit all stops within the same zone first and then move to the next zone, we solve a standard TSP with a modified travel time matrix to generate zone sequence first and then solve multiple path-TSP to identify intra-zonal stop sequence.
First, we provide the formulation of the standard TSP solved for generating zone sequences. For a route instance with \( n \) zones, the set of zones is indexed by \( [n] = \{1, \ldots, n\} \) and the initial station location is indicated by index 0. Let \( V \) represent the set of all locations that need to be visited including the initial station, i.e., \( V = \{0, 1, \ldots, n\} \). \( t_{ij} \) denotes the travel time between any two locations, i.e., \( \forall i \neq j \in V \). The travel time between any two zones is calculated as the average travel time between all possible pairs of stops between two zones. The decision variable for this problem is \( x_{ij} \in \{0, 1\} \), \( \forall i, j \in V \). \( x_{ij} = 1 \) indicates that the driver will visit to the location \( j \) after visiting \( i \). Then, the TSP problem can be formulated as:

\[
\begin{align*}
\text{min} & \quad \sum_{i=0}^{n} \sum_{j=0}^{n} t_{ij} x_{ij} \\
\text{s.t.} & \quad \sum_{i=0}^{n} x_{ij} = 1 \quad \forall j \in V \tag{2a} \\
& \quad \sum_{j=0}^{n} x_{ij} = 1 \quad \forall i \in V \tag{2b} \\
& \quad \sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1 \quad \forall S \subset V, S \neq \emptyset, V \tag{2c} \\
& \quad \sum_{i \notin S} \sum_{j \in S} x_{ij} \geq 1 \quad \forall S \subset V, S \neq \emptyset, V \tag{2d} \\
& \quad x_{ii} = 0 \quad \forall i \in V \tag{2e} \\
& \quad x_{ij} \in \{0, 1\} \quad \forall i, j \in V \tag{2f}
\end{align*}
\]

Where the objective (2a) minimizes the total travel time for the tour. Constraints (2b) and (2c) make sure that each visited location has exactly one predecessor and one successor in the optimal tour. Constraints (2d) and (2e) are proposed to eliminate subtours in the optimal tour. Constraints (2f) avoid self loops and constraints (2g) guarantee decision variables are binary.

The problem (2) is an Integer Linear Programming (ILP) with exponential number of constraints due to constraints (2d) and (2e). To solve this problem efficiently, we implemented both constraints (2d) and (2e) as lazy constraints, indicating they are only added to the problem if subtours are identified in the current optimal solution. The lazy constraints are one of the implementation of the cutting-plane method.

To account for the observations made in the zone sequence (Section 3.1), we propose three heuristics to modify the travel time matrix, which is the input for generating the optimal zone sequence.

1. For travel time from the initial station to a zone \( i \), if the zone is not within either i) \( h \) closest zones from the initial station regarding travel times or ii) \( h \) closest zones from the initial station regarding Euclidean distances, we modify the travel time to \( t_{0i} \times \alpha \), where \( \alpha \) and \( h \) are both parameters for the first proposed heuristic approach.

2. For travel time between any two zones \( i \) and \( j \), if zone \( i \) and zone \( j \) are not from the same "major zone", we modify the travel time to \( t_{ij} \times \beta \), where \( \beta \) is the parameter for the second proposed heuristic.
3. For travel time between any two zones $i$ and $j$, if they are from the identical "major zone" and the difference between their zone ID after the dot does not equal to 1, we modify the travel time to $t_{ij} \ast \gamma$, where $\gamma$ is the parameter for the third proposed heuristic approach.

In the final submitted algorithm, we used the grid search approach to finalize values for all four heuristic parameters: $h = 9$, $\alpha = 1.04$, $\beta = 3.8$, $\gamma = 2.5$.

Solving the problem \(^2\) with the modified travel time matrix leads to the optimal zone sequence\(^1\) $S^* = (0, s_1, ..., s_n)$, where $s_i$ indicates the $i$-th zone visited in the optimal sequence after departing from the initial station. Then we solve the intra-zonal stop sequence using path-based TSP. Given a set of locations $V$ need to be visited and the starting location $v_o$ and the ending location $v_d$, we can formulate the path-TSP problem as follows:

\[
\begin{align*}
\min & \quad \sum_{i=0}^{n} \sum_{j=0}^{n} t_{ij}x_{ij} \\
\text{s.t.} & \quad \sum_{i=0}^{n} x_{ij} = 1 \quad \forall j \in V \setminus \{v_o,v_d\} \\
& \quad \sum_{j=0}^{n} x_{ij} = 1 \quad \forall i \in V \setminus \{v_o,v_d\} \\
& \quad \sum_{j \in V} x_{vo,j} = \sum_{i \in V} x_{iv_o} = 1 \\
& \quad \sum_{j \in V} x_{vd,j} = \sum_{i \in V} x_{iv_d} = 0 \\
& \quad \sum_{i \in S} \sum_{j \not\in S} x_{ij} \geq 1 \quad \forall S \subset V, S \neq \emptyset, V \\
& \quad \sum_{i \not\in S} \sum_{j \in S} x_{ij} \geq 1 \quad \forall S \subset V, S \neq \emptyset, V \\
& \quad x_{ii} = 0 \quad \forall i \in V \\
& \quad x_{ij} \in \{0,1\} \quad \forall i,j \in V
\end{align*}
\]

The path-TSP problem \(^3\) is similar to the standard TSP problem \(^2\) except that there will be no predecessors for the starting location $v_o$ and no successors for the ending location $v_d$, indicating by constraints \(^3\) and \(^3\). The complete sequence is generated according to Algorithm \(^1\) based on generated zone sequence, where a heuristic parameter $k = 3$ is utilized in the final implementation.

It is worth mentioning that all TSP instances are solved with the open-source ILP solver GLPK implemented with programming language Julia \(^{Bezanson et al. 2017}\) and optimization package JuMP \(^{Dunning}\).

---

\(^1\)Without loss of generality, we can assume the sequence starts from the initial station indexed by 0.
Algorithm 1 Complete sequence generation based on the generated zone sequence.

Input: optimal zone sequence $S^* = (0, s_1, ..., s_n)$, heuristic parameter $k$.

1: function CompletePath Generation($S^*$) 
2: $S_{\text{complete}}^* \leftarrow \{0\}$ \hspace{1cm} ▷ Initialize the complete sequence with the initial station
3: for $s_i = s_1, ..., s_n$ do
4: \hspace{1cm} Find the previous visited zone $s_i - 1$ and the next visited zone $s_i + 1$
5: \hspace{1cm} Calculate the average travel time between any stop $v \in s_i$ to all stops in zone $s_i - 1$ and zone $s_i + 1$
6: \hspace{1cm} Find $k$ nearest stops in zone $s_i$ regarding to zone $s_i - 1$ as the set $M$
7: \hspace{1cm} Find $k$ nearest stops in zone $s_i$ regarding to zone $s_i + 1$ as the set $N$
8: \hspace{1cm} Solve $k^2$ path-TSP (3) between any pair of stops in $M \times N$.
9: \hspace{1cm} Let the path $S_i^*$ with the minimum travel time as the optimal sequence of zone $i$
10: \hspace{1cm} Append the sequence $S_i^*$ to the complete sequence $S_{\text{complete}}^*$
11: return $S_{\text{complete}}^*$

et al., 2017). After generating the complete stop sequence $S_{\text{complete}}^*$, we enter the post-processing stage to further improve sequence performances.

3.3. Post-Processing

After solving the stop sequence by TSP, we observe that most of the high-score (i.e., low performance) routes are due to partially or fully reverse of the sequence (i.e., a sequence A-B-C-D is erroneously estimated as D-C-B-A). Hence, we propose a post-processing method to correct the erroneous estimation due to reversal.

We observe two properties from the data set:

- Most of the drivers tend to serve the business areas first. The potential reason may be that it also takes a longer time to deliver packages in a business building. Serving them first can make the total service time more controllable at the end of the journey. Hence, we expect that the planned service time at the first several stops is larger than that of the last several stops.

- Most of the drivers tend to deliver large-size packages first. This may be because carrying large-size packages in the vehicle is not fuel-efficient.

Based on these properties, for every generated stop sequence by TSP, we check whether we need to reverse it. Given a generated route $i$, let $p_{i}^+$ (resp. $p_i^-$) be the average planned service time of the first (resp. last) $p\%$ stops in route $i$. We will reverse route $i$ if

$$\frac{p_i^-}{p_i^+} \geq \theta,$$

where $p$ and $\theta$ are hyperparameters representing the proportion of stops and a threshold. We set $p = 15$ and $\theta = 1.22$ based on cross-validation on the test set. Eq. 5 means that in a generated sequence if the planned
service time for the last several stops is too large, we may have the reversal error and need to correct it by reverse the whole sequence.

After process by Eq. 5, we fixed all sequences that are already reversed. For the remaining sequences, we further check whether they need to be reversed based on package volumes. Specifically, given a generated route $i$, let $v_i^+$ (resp. $v_i^-$) be the total package columns (depth $\times$ width $\times$ height) of the first (resp. last) 15% stops in route $i$. We will reverse route $i$ if

$$\frac{v_i^-}{v_i^+} \geq \eta,$$

where $\eta = 3$ is used.

After post-processing, a sequence validity check is performed. Specifically, we check whether the first stop of the estimated sequence is the delivery station, and whether the estimated sequence has the same stop IDs as the actual one. If either of these two criteria does not hold, we return a sequence by simply sort the stops using zone IDs, which ensures that stops with the same zone IDs are close to each other.

4. Results and Conclusions

4.1. Performance

Although the submitted formulation does not require model training, we have separated the given training set into the training (4889) and test set (1223) for self-evaluation of the machine learning models. Therefore, all self-evaluation is done over the test set. To reduce the evaluation time, we implemented the scoring function using Cython. Compared to the evaluation code in Python provided by the challenge host team, our implementation evaluates the same set of routes by using only one-third of the computation time.

Figure 2 shows the score distribution generated by our final algorithm. The route performance score follows an exponential distribution and most routes have a score below 0.1. The average route score is 0.0381 for these 1223 testing routes. On the 13 routes in the given model apply set, the score was 0.0375.

![Figure 2: Route score performances.](image)
4.2. Discussion

Zone sequence dominates score. We observe that, if the zone sequence is perfectly predicted, even if the stop IDs within a zone are shuffled, the average route score can reach 0.0206. Hence, most of our jobs focus on predicting the zone sequence, instead of the stop sequence.

The three properties of zone IDs (see Section 3.1) may imply that drivers most likely follow the planned route and seldom deviate. As the zone ID is used to "help simplify the route planning process" (quoted from Slack Q&A), we believe that Amazon plans the route in a way that the zone IDs exhibit clear patterns. So the major challenge of this problem is to recover how Amazon plans the routes. This explains why TSP works better than machine learning methods under the given current information and sample size.

The reversal problem remains. Figure 3 shows an example of reverse prediction. Since we are not able to increase the first-zone prediction accuracy beyond 35%, after post-processing, the reverse issues still exist. The post-processing reduces our score on our test set from 0.391 to 0.381. However, if we can have a 100% correction rate for the reversal problems (i.e., always use the one with a smaller score), the score can reduce to 0.280, indicating that some further correction methods are needed. Note that we have tried to use the number of surrounding highway ramps as a new indicator, as well as using machine learning to predict the first zone, but it does not increase the model performance.

(a) Actual route
(b) Predicted route by TSP

Figure 3: Examples of reverse prediction
References


Article III

A Short-Sighted Machine Learning Implementation on the Vehicle Routing Problem

Kelvin Anthony and Kenny Anthony

Source code is available on GitHub:
https://github.com/kiguess/lmrrc-kk
A Short-Sighted Machine Learning Implementation on the Vehicle Routing Problem

Kelvin Anthony
School of Astronautics, Northwestern Polytechnical University
127 Youyi West Road, Xi’an, Shaanxi, P.R. China

Kenny Anthony
School of Energy and Power Engineering, Jiangsu University
301 Xuefu Road, Zhenjiang, Jiangsu, P.R. China

Abstract
The vehicle routing problem is very hard to solve in real life, where unidentifiable factors are setting a gap between an experienced driver and traditional computer programs. To account for this, we seek an approach to find out the previously ignored patterns, that is, the tacit knowledge of the driver and apply it to our new routing heuristic. In our approach, the past routes are firstly broken into individual paths, assigning the single point to single point paths the same score as the whole route itself, where higher scores constitute a better performance. We trained the XGBoost algorithm to predict the scores of the plausible paths to the three closest points from our current location as well as to the three closest points from each of the proposed next stops. By applying a weighted approach to the scores, we obtain the Overall Score of each path and take the highest scoring path as our next stop. This heuristic is then repeated until all stops are covered in our route. Testing our approach on the build dataset, we obtained the average route score of 0.180, the lower the better. Contrary to other classical heuristics that deliver better results by optimizing the bigger picture, by this short-sighted heuristic, we are opening possibilities for a set-up where couriers can also receive sudden pickup requests from customers along the way and are able to execute it immediately, which may improve the customer satisfaction and efficiency of logistic companies handling two-way shipments.

1. Introduction
The currently available data of previous routes are structured as sequences consisting of various stops ranging from tens to hundreds of points, along with the departure time from the station and travel time for each path. Categorical scores have been assigned to the routes based on their performance by Amazon.
To simplify the Machine Learning model, the routes are broken down into one point to one point paths, assuming that all the individual paths have the same score as their parent route. We then generate a new Python DataFrame with variables that are considered influential to the prediction performance, as indicated in the feature performance evaluation process. We convert categorical scores to numerical scores by assigning a numerical value of equal intervals to each categorical score, in this case, we assign Low as 1, Medium as 5, and High as 9 for a better readability. In addition, using small numbers might cause the model to be more sensitive towards floating point inaccuracies. The numerical scores chosen have the same intervals to provide a level playing field in the prediction function.

We use the XGBoost algorithm, a gradient boosting algorithm, to base our Machine Learning model upon. We define the Score as our target variable with all other variables in the predefined DataFrame as our feature variables. All these variables are regressed by reducing the squared error to obtain the most accurate estimation of the score while ensuring that the parameters are not causing the model to overfit the training set.

In this study, we develop a short-sighted heuristic to apply the Machine Learning algorithm to the Vehicle Routing Problem. We simply take the closest two levels of path, which is the next stop and a stop after the next stop, into our consideration, where we predict the scores of each path and store them in a list. A formula is used to deduce the overall scores of the path options. By this method, we choose the stop with the highest overall score as our next stop. Please note that we only connect the first level path between the current position and the next stop and not the second level path. This heuristic then continues to repeat in a loop until all points are connected to the route.

2. Literature Review

The vehicle routing problem is a classical problem in operation research. As it is an NP-hard problem, it is almost impossible to find the optimum route within an acceptable compute time when we have a large number of stops. Over the past decades, many studies have revolved around finding the most optimal and fast routing algorithms and heuristics. Several researchers propose some variation of the implementation of Large Neighborhood Search algorithm (Hintsch & Irnich (2018), Arnold & Sörensen (2019), Dumez et al. (2021)). Tabu search is also popular among researchers, with many studies based upon it (Brandão (2004), Jia et al. (2013), Qiu et al. (2018)). Park et al. (2021) leveraged the genetic algorithm to solve the vehicle routing problem with simultaneous pickup and delivery (VRPSPD). Most recently, the memetic algorithm, an extension of the genetic algorithm, has gained popularity for its ability to reduce premature convergence, ultimately leading to better results. Mańdziuk & Żychowski (2016), Sabar et al. (2020), and Liu et al. (2021) are among the few researchers working on integrating the memetic algorithm to the VRP.

However, it is worth noting that not all optimal solutions on the VRP are applicable in the real world since there may be undocumented route closures or certain unfavorable factors in some locations. In fact, the tacit knowledge of the driver often plays a key role in routing. To derive insights, identify patterns, and
account for unknown variables from previous real driver routes, we implement a Machine Learning algorithm in complement to our routing heuristics.

In our model, as we are trying to predict the score of each proposed path, we are looking for an algorithm that provides the ability to classify the characteristics of each path. To solve this, we choose the Decision Forest algorithm. To increase the learning ability of decision forests, the weak learners are being combined to make a strong learner, which is what we call boosting. In the case of Gradient Boosting Machines, the decision trees are the weak learners. Among them, the XGBoost algorithm is chosen for its outstanding performance, versatility, scalability, and speed (Chen & Guestrin, 2016).

Additionally, to develop a model that is able to integrate first mile pickups and last-mile deliveries, we develop a short-sighted algorithm, where we only consider the next two stops down the route. Although not being the most optimal model, our study enables sudden insertion of stops, which caters to immediate pickup requests from customers. Bergmann et al. (2020) find that the integration of first mile pickups and last-mile deliveries may deliver up to 30 percent savings in urban e-commerce distribution.

In response to the challenges faced above, we develop a novel approach to the VRP by combining a machine learning model and a short-sighted routing heuristic. To be specific, we integrated the XGBoost machine learning prediction, that identifies and scores the path according to patterns found from the tacit knowledge, to our routing heuristic, allowing us to benefit from the field knowledge of an experienced driver while also staying flexible to allow new pickup stops to be appended to the route.

3. Methodology

The currently available data of previous routes are structured as sequences consisting of various stops ranging from tens to hundreds of points, along with the departure time from the station and travel time for each path. Categorical scores have been assigned to the routes based on its performance by Amazon. To simplify the XGBoost Regression Machine Learning model, the routes are break down into one point to one point paths, assuming that all the individual paths are having the same score as its parent route. We then generate a new Python DataFrame with variables that are considered influential to the prediction performance, as indicated in the feature performance evaluation process.

Please note that we consider the differences in latitude and longitude in our model to account for the direction of the path, we believe that this approach is superior to that of including the actual distance since urban traffic depend heavily on time and direction.

Then, the data is split to training and testing sets with the proportion of 70% to the training set and 30% to the testing set with the random state of 2018. From the training set itself, we use the random state of 2019 to take 25% of the training set to be the validation set, leaving the rest 75% in the new training set.
Table 1: Variables included in the DataFrame

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hour</td>
<td>Number of hours since the start of the day (16:36 is taken as 16.6)</td>
</tr>
<tr>
<td>origin_lat</td>
<td>Latitude of the origin stop</td>
</tr>
<tr>
<td>origin_long</td>
<td>Longitude of the origin stop</td>
</tr>
<tr>
<td>dest_lat</td>
<td>Latitude of the destination stop</td>
</tr>
<tr>
<td>dest_long</td>
<td>Longitude of the destination stop</td>
</tr>
<tr>
<td>delta_lat</td>
<td>Obtained from dest_lat – origin_lat</td>
</tr>
<tr>
<td>delta_long</td>
<td>Obtained from dest_long – origin_long</td>
</tr>
<tr>
<td>time_taken</td>
<td>Time taken for this trip in seconds</td>
</tr>
<tr>
<td>score</td>
<td>Score of the route (High=9, Medium=5, and Low=1)</td>
</tr>
</tbody>
</table>

Based on the above variables, we train the XGBoost algorithm with the below parameters, noting that score is set as target variable.

Table 2: XGBoost Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Booster</td>
<td>gbtree</td>
</tr>
<tr>
<td>Objective</td>
<td>reg:squarederror</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.05</td>
</tr>
<tr>
<td>Max Depth</td>
<td>14</td>
</tr>
<tr>
<td>Subsample</td>
<td>0.9</td>
</tr>
<tr>
<td>Subsample Ratio of Columns by Each Tree</td>
<td>0.7</td>
</tr>
<tr>
<td>Subsample Ratio of Columns by Each Level</td>
<td>0.7</td>
</tr>
<tr>
<td>Evaluation Metric</td>
<td>RMSE</td>
</tr>
</tbody>
</table>

After we finish training and saving the model, we apply the model into our short-sighted routing heuristics. In principle, our model uses the prediction feature of the XGBoost algorithm to obtain the score estimate for each of the proposed paths, which are then calculated by the tweaked formulation defined below to obtain the overall score of a path. Note that 5 is the weight, obtained through repeated experiments, that we assign to the variable $\text{Score}_{S1}$ in order to prioritize higher scoring candidate to be chosen as the next stop and weaken the influence of paths originating from next stop to second level destinations as shown in the figure. The next stop candidate with the highest overall score will be set as our next stop.

$$\text{OverallScore}_{S1} = 5 \cdot \text{Score}_{S1} + \text{Score}_{1a} + \text{Score}_{1b} + \text{Score}_{1c}$$
Algorithm 1: Pseudocode of the Short-sighted Routing Heuristic

**Result:** Route Sequence

Current Stop = Departure Station;

while stops left ≥ 1 do
  Find Max Overall Score with the above formulation;
  Next Stop ← Stop with highest overall score;
  Link the path from Current Stop to Next Stop;
  Current Stop ← Next Stop;
end
4. Results and Conclusions

To assess the performance of our heuristic, the sequences generated by our algorithm is tested on a comparison basis with the real driver-taken sequence provided by Amazon. It combines elements of Sequence Deviation and Edit Distance with Real Penalty. The scoring formula is defined as follows:

$$\text{score} = \frac{SD(A, B) \cdot \text{ERP}_{\text{norm}}(A, B)}{\text{ERP}_e(A, B)}$$

where sequence A is the historically realized sequence of deliveries, sequence B is the algorithm-produced sequence of deliveries, SD denotes the Sequence Deviation of B with respect to A, ERP\text{norm} denotes the Edit Distance with Real Penalty applied to sequences A and B with normalized travel times, and ERP\text{e} denotes the number of edits prescribed by the algorithm on sequence B with respect to A. If edit distance with real penalty prescribes 0 edits, then the above formula is replaced by the sequence deviation, multiplied by 0. Sequences that perfectly match the driver-taken sequence are given a score of 0. Scores increase as the user-submitted sequence differs more and more from the driver-taken sequence. Complete random shuffles of all the stops in the driver-taken route typically receive scores between 0.8 and 1.2.

![Figure 3: Distribution of Scores when Tested in the Build Dataset](image)

When testing the model on the dataset used to train the model itself, we obtained the average score of 0.180583. The distribution is positively skewed with the skewness of 1.464 and kurtosis of 3.647. We observe that most scores belongs to the 0.1-0.15 range and the median is 0.164.
Meanwhile, when scored against the provided testing set, we obtained the following results:

<table>
<thead>
<tr>
<th>No</th>
<th>Route ID</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15bbae2d-bf07-4967-956a-173d4036613f</td>
<td>0.15289621211672733</td>
</tr>
<tr>
<td>2</td>
<td>3f166f0e-fd2e-47ab-96a0-6cbc99cc6ef</td>
<td>0.15411727540160575</td>
</tr>
<tr>
<td>3</td>
<td>5486294a-503f-4346-b8a9-862e888cbe7c</td>
<td>0.26199626613573446</td>
</tr>
<tr>
<td>4</td>
<td>693060a6-88bb-4324-9e9c-925d5240263c</td>
<td>0.18857377795140118</td>
</tr>
<tr>
<td>5</td>
<td>75d87f0-c39f-434f-bf3f-b159ef321909</td>
<td>0.19281615292821577</td>
</tr>
<tr>
<td>6</td>
<td>9475872b-287f-4c2c-8e29-887766a40e90</td>
<td>0.15027958838186056</td>
</tr>
<tr>
<td>7</td>
<td>a8f0009d-c50a-49e9-84d3-f9885ad14a54</td>
<td>0.14202723495705352</td>
</tr>
<tr>
<td>8</td>
<td>bcc07fea-86d2-41e4-9a58-cfc78956dccc7</td>
<td>0.10963238389656335</td>
</tr>
<tr>
<td>9</td>
<td>d1a8c3dd-fa67-455c-a68d-a2f2df6aad9d1</td>
<td>0.18016271458201608</td>
</tr>
<tr>
<td>10</td>
<td>e6687a5-2453-4edc-b86c-7558ab6d93f6</td>
<td>0.07227743642231328</td>
</tr>
<tr>
<td>11</td>
<td>2b8df6f6d-fcd4-438e-931c-3b84b36a5c6b</td>
<td>0.36520566155225170</td>
</tr>
<tr>
<td>12</td>
<td>f3261fad-5f97-44f6-ae7f-cf169f5d6452</td>
<td>0.17841644964316614</td>
</tr>
<tr>
<td>13</td>
<td>fffd257c-3041-4736-be7a-5efea8af1173</td>
<td>0.04690073429091316</td>
</tr>
<tr>
<td></td>
<td>Average Score</td>
<td>0.16886937601998633</td>
</tr>
</tbody>
</table>

On a deeper dive into our data, we identify the pattern where the model tends to perform worse such as in Route 11 when the training data does not overlap much with the testing set location. Vice versa, we find routes like Route 13 perform really well since it covers an area highly populated by stops in the training process. This conforms with the general principle of regression, where the prediction results are valid only when the model is applied within the same range with the training data.

Finally, this study lays off a foundation for future developments of short-sighted Machine Learning implementation on the Vehicle Routing Problem, particularly in the integration of first mile and last mile logistics. We expect to see further development in the inclusion of vehicle capacity and time window constraints that we ignored in this case, allowing our algorithm to provide a quick and immediate response to pick up requests along the way while also conforming to reality. Additionally, we assumed that the individual path scores equal to the route scores, allowing us to break the route into singular paths. Future researches should identify the connection patterns of multiple stops that made a certain segment of the route.
References


Article IV

Route Sequence Prediction Through Inverse Reinforcement Learning

Anselmo R. Pitombeira-Neto

Source code is available on GitHub:
https://github.com/anselmo-pitombeira/route-seq-prediction
Route sequence prediction through inverse reinforcement learning

Anselmo R. Pitombeira-Neto\textsuperscript{a,b}

\textsuperscript{a}OPL Lab - Operations Research in Production and Logistics, Federal University of Ceará, Fortaleza, CE, Brazil
\textsuperscript{b}Department of Industrial Engineering, Federal University of Ceará, Campus do Pici, Fortaleza, CE, Brazil

Abstract

We consider the problem of predicting the route sequence a driver will follow given a set of stops, related data and additional data on the delivered packages. We approach the problem from the perspective of inverse reinforcement learning. Given a sample of route sequences generated by a set of drivers, we want to learn a parameterized cost function that drivers are trying to minimize. Predicted route sequences are generated by using a rollout algorithm, which is a kind of approximate dynamic programming. We define our loss function as the average score over predicted route sequences from training data. As the loss function is nondifferentiable, we resort to Bayesian optimization, which allows us to efficiently find good parameter values given a limited computational budget. In preliminary experiments, after running a Bayesian optimization algorithm for about 30 iterations, predicted sequences achieved mean and median scores in a holdout sample equal to 0.0958 and 0.0823, respectively.

1. Introduction

We consider the problem of predicting the sequence of stops a driver will follow given a set of required stops, related data and additional data on the delivered packages. We approach the problem from the perspective of inverse reinforcement learning (IRL). Reinforcement learning is a technique related to dynamic programming in which we want to find a (sub)optimal decision policy for an intelligent agent such that it accomplishes some desired task (Sutton & Barto 2018). The desired behavior is achieved through the design of a reward (or cost) function. In contrast, in IRL we are given a sample of demonstrations from one or more intelligent agents and we want to learn a reward function compatible with the behavior of the agents (Ng & Russell 2000; Abbeel & Ng 2004).

From the point of view of a driver, the problem of predicting a route sequence is very similar to a traveling salesman problem (TSP) with time windows (TSPTW). It is well known in the literature that the TSP may be formulated as a dynamic program, in which the states are partial sequences of stops and the cost function is assumed to be the distance (or time) between a current stop and the possible next stops (Bellman 1962).
In our case, we want to solve the inverse problem: we have demonstrations of solutions to the TSPTW generated by a set of drivers, and we want to learn a cost function that the drivers are trying to minimize. If we can reasonably approximate this cost function, we may use dynamic programming to generate a route sequence given a set of stops, which can then be used as the predicted sequence of stops a driver would follow.

We use a linear model for the cost function and try to learn the parameters from the sample of demonstrations. However, we face some hurdles: in theory, we may use maximum likelihood (or even Bayesian inference) to learn the parameters, but trying to compute a likelihood function demands specifying a probability measure over all possible TSPTW solutions, which is computationally intractable. We then define our loss function as the average score over route sequences in a training sample and try to set the parameters so as to minimize the average score.

Moreover, in order to compute the score, we need to generate route sequences by solving the TSPTW with given parameters for the cost function. However, large TSPs cannot be solved by exact dynamic programming due to the exorbitant state space. We solve them heuristically by using a rollout algorithm, which is a kind of approximate dynamic programming. Finally, as the loss function is nondifferentiable, this precludes us from using gradient-based methods. (And computing finite differences will be overkill due to the large time to evaluate the score function in the whole sample.) We then resort to Bayesian optimization, which allows us to efficiently find good parameter values given a limited computational budget.

2. Literature Review

Dynamic programming (DP) is a well-known technique for solving sequential decision problems proposed by Bellman (1959). In contrast to mathematical programming, which produces optimal open loop controls for a decision problem, the objective in DP is to find a decision policy, i.e., closed loop controls. A decision policy specifies which decision to make at each possible state of a system or environment. DP suffers from the curse of dimensionality, which refers to the combinatorial explosion of the number of states of a system as the dimension of the state vector increases. In addition, DP assumes that a model of the system is available, which often is not the case.

Reinforcement learning (RL) is a machine learning technique whose mathematical framework is essentially the same as DP, but which tries to overcome the abovementioned limitations of DP by using approximation techniques such as Monte Carlo simulation, artificial neural networks, etc. (Sutton & Barto, 2018). Many of these techniques have been concurrently developed by operational researchers under the name approximate dynamic programming (Powell, 2011; Gosavi, 2003).

A noteworthy technique for approximately solving DP/RL problems is called the rollout algorithm. In DP/RL, an optimal policy is obtained if one knows the optimal value function, which may be computed by backward dynamic programming in the case of finite-horizon problems. The rollout algorithm is a kind of forward dynamic programming, in which the value function of a state in the next decision epoch is computed.
heuristically on the fly \cite{Bertsekas1997, Bertsekas1999}. It has been applied to both the TSP \cite{Guerriero2002} and the TSPTW \cite{Cazenave2012}. It is a component of the Monte Carlo tree search technique, which has been successfully applied to games \cite{Browne2012}.

In RL/DP, the reward/cost function is often specified to obtain a desired solution of the sequential decision problem. In contrast, in inverse reinforcement learning (IRL), we are interested in estimating a reward/cost function from a sample of observed trajectories produced by intelligent agents \cite{Ng2000, Abbeel2004}. IRL has been recently applied to medical decisions \cite{Belogolovsky2021} and to trajectory modeling \cite{Pitombeira-Neto2021}. A recent survey on IRL is presented by \cite{Arora2021}.

Our approach draws heavily from ongoing work from the author on modeling sparse trajectories with the use of IRL \cite{Pitombeira-Neto2021}. But, in contrast to this latter work, in which value functions may be computed exactly, the state space in the route sequence prediction problem is too large. The works on the application of the rollout algorithm to the TSP and the TSPTW sparked the idea of using a parameterized rollout algorithm to approximate the value function. As far as we know, our approach has not been tried before in the route sequence prediction problem.

3. Methodology

The core idea of our approach is to iteratively alternate between a route sequence prediction step and a parameter update step. For a given dataset related to a sample of observed route sequences, we apply a parameterized route sequence prediction algorithm and compute the score of each predicted route sequence. We then apply a Bayesian optimization step to suggest a new set of parameter values with the objective of minimizing the average score over the sample. The two steps are iterated until a predefined number of iterations is met.

3.1. Prediction of a route sequence given data

Given a set of stops $\mathcal{S}$ and related data $\mathcal{D}$, we assume a driver is an optimizer and tries to find a route sequence $r = (s_1, s_2, \ldots, s_n, s_1)$ so as to minimize some cost function unknown to an external observer, in which the starting $s_1$ is fixed and called the station and $n = |\mathcal{S}|$. We define as the state $x_t \in \mathcal{X}_t$ at a decision epoch $t$ a partial sequence:

$$x_t := (s_1, s_2, \ldots, s_t), \quad 1 \leq t < n,$$

in which $\mathcal{X}_t$ is the set of all combinations of stop sequences of size $t$. At a state $x_t$, a driver chooses a next stop $a_t$ from a set of remaining stops $\mathcal{A}(x_t) = \mathcal{S} \setminus \{x_t\}$ and then incurs an immediate cost given by

$$c(x_t, a_t) = \sum_{k=1}^{m} \phi_k(x_t, a_t)\beta_k, \quad (1)$$

in which $\phi_k(x_t, a_t), k = 1, \ldots, m$ represent features associated with the current partial sequence and the next stop, and $\beta_k, k = 1, \ldots, m$ are parameters. We assume that features are also functions of the data $\mathcal{D}$ related to the stops.
Notice that, given a set of parameter values, we may (in theory) find an optimal route sequence by solving the following problem:

\[
\min_{a_1, a_2, \ldots, a_{n-1}} \sum_{t=1}^{n-1} \alpha^t c(x_t, a_t) + \alpha^n c(x_n, s_1),
\]

\[\text{s.t.}\]
\[
x_{t+1} = (x_t, a_t), \quad t = 1, \ldots, n - 1,
\]
\[
a_t \in \mathcal{A}(x_t), \quad t = 1, \ldots, n - 1,
\]
\[
x_t \in \mathcal{X}_t, \quad t = 2, \ldots, n - 1,
\]
\[
x_1 = s_1,
\]

in which \(0 < \alpha \leq 1\) is a discount factor which controls “how far into the future” the agent plans. The optimization problem \((2)\) may be solved by backward dynamic programming by recursively solving the Bellman equation

\[
v_t(x_t) = \min_{a_t \in \mathcal{A}(x_t)} \{c(x_t, a_t) + \alpha v_{t+1}(x_{t+1})\}, \quad \forall x_t \in \mathcal{X}_t, \quad t = 1, \ldots, n - 1,
\]

in which \(v_t(x_t)\) is the value function associated with state \(x_t\) (also called a cost-to-go function), and \(v_n(x_n) = c(x_n, s_1)\). However, it will be computationally infeasible for a large set of stops, so we resort to a rollout algorithm, which is a form of forward dynamic programming.

In the rollout algorithm, at a given state \(x_t\), a decision \(a_t^*\) is made according to

\[
a_t^* \in \arg \min_{a_t \in \mathcal{A}(x_t)} \{c(x_t, a_t) + \alpha h_{t+1}(x_{t+1})\},
\]

in which \(h_{t+1}(x_{t+1})\) is an estimate of the value function \(v_{t+1}(x_{t+1})\) obtained by generating a sequence of stops by following a heuristic decision policy. We will use a greedy policy in relation to the cost function \(c(x_t, a_t)\) as a base heuristic, i.e., at each state \(x_t\) the greedy policy chooses the next stop which minimizes the immediate cost. The rollout algorithm is guaranteed to obtain solutions at least as good as the base policy (under some conditions) due to the policy improvement property of dynamic programming (Bertsekas, 2011).

In addition, due to the existence of time windows, we maintain a clock variable \(\Gamma\) which tracks the current time in the route sequence. We denote the left bound and the right bound of a time window at a stop \(s \in \mathcal{S}\) as \(w^{-}(s)\) and \(w^{+}(s)\), respectively. The purpose of this clock variable is to check it against the time windows and use this information in the cost function. It is initialized to the departure time and updated by the service times and travel times along the route.

At a given state \(x_t = (s_1, s_2, \ldots, s_t)\) and for each possible next stop \(a_t \in \mathcal{A}(x_t)\), we have defined the following features in the cost function \([1]\) according to the available data on the stops and packages:

1. \(\phi_1(x_t, a_t)\) - The travel time \(\tau_1(x_t, a_t)\) between the current stop \(s_t\) and a possible next stop \(a_t\);
2. \(\phi_2(x_t, a_t)\) - The service time \(\tau_2(a_t)\) in a next stop \(a_t\);
3. \(\phi_3(x_t, a_t)\) - The maximum dimension size of all packages in a next stop;
Algorithm 1 Rollout algorithm for computing a route sequence

Input: Stops $S$, station $s_1$, data $D$, features $\phi(,)$, parameters $\theta$

1: $S' \leftarrow S$
2: $\Gamma \leftarrow$ departure time at $s_1$
3: $x \leftarrow [s_1]$
4: while $S' \neq \emptyset$ do
5: $q^* \leftarrow +\infty$
6: for $a \in S'$ do
7: Compute $c(x, a) = \sum_{k=1}^{m} \phi_k(x', a) \beta_k$
8: $x' \leftarrow [x, a]$
9: $\Gamma' \leftarrow \Gamma + \tau_1(x, a) + \tau_2(a)$ \Comment{Tentative clock}
10: Compute approximate value function $h(x'; \Gamma')$ \Comment{Run greedy heuristic in Alg. [2]}
11: $q(x, a) \leftarrow c(x, a) + \alpha h(x')$
12: if $q(x, a) < q^*$ then
13: $q^* \leftarrow q(x, a)$
14: $a^* \leftarrow a$
15: end if
16: end for
17: $\Gamma \leftarrow \Gamma + \tau_1(x, a^*) + \tau_2(a^*)$ \Comment{Update route clock}
18: $x \leftarrow [x, a^*]$
19: $S' \leftarrow S' \setminus \{a^*\}$
20: end while
21: return $x$ \Comment{generated route sequence}

4. $\phi_4(x_t, a_t)$ - The time difference between the starting of the time window of a next stop and the current clock, truncated to zero if the difference is negative, i.e., $\max(0, w^-(a_t) - \Gamma)$;
5. $\phi_5(x_t, a_t)$ - The time difference between the end of the time window of a next stop and the current clock, i.e., $\Gamma - w^+(a_t)$;

Alg. [1] describes in detail the rollout algorithm to compute a route sequence, while Alg. [2] describes the procedure to compute the approximate value function $h(x)$ at a state $x$. Notice that in Alg. [2] we penalize $h(x)$ by a penalty $P$ in case a time window is not met and $\gamma \geq 0$ is a parameter which adjusts the magnitude of the penalty. The penalty $P$ should be a large value, so we have assigned to $P$ the sum of all travel times between stops.

3.2. Learning the Parameters of the Prediction Function

We define a prediction function

$$\hat{r} = f(S, D, \theta)$$
Algorithm 2 Greedy heuristic for approximating the value function at a state $x$

Input: Partial route $x$, stops $S$, station $s_1$, current tentative clock time $\Gamma'$, windows penalty $P$, features $\phi(\cdot)$, parameters $\theta$, data $D$

$S' \leftarrow S \setminus \{x\}$

$x' \leftarrow x$

$h(x) \leftarrow 0$

while $S' \neq \emptyset$ do

$c^* \leftarrow +\infty$

for $a \in S'$ do

Compute $c(x',a) = \sum_{k=1}^{m} \phi_k(x',a)\beta_k$

if $c(x',a) < c^*$ then

$c^* \leftarrow c(x',a)$

$a^* \leftarrow a$

end if

end for

$\Gamma' \leftarrow \Gamma' + \tau_1(x',a^*) + \tau_2(a^*)$ \Comment{Update route clock}

$x' \leftarrow [x',a^*]$

$h(x) \leftarrow h(x) + c^*$

if $\Gamma' < w^-(a^*)$ or $\Gamma' > w^+(a^*)$ then

$h(x) \leftarrow h(x) + \gamma P$ \Comment{Penalty for violating time window}

end if

$S' \leftarrow S' \setminus \{a^*\}$

end while

$h(x) \leftarrow h(x) + c(x',s_1)$

return $h(x)$

which produces a predicted route sequence $\hat{r}$ given the set of stops and related data by applying the rollout algorithm given in Alg. 1 in which $\theta = (\alpha, \beta_1, \beta_2, \ldots, \beta_m, \gamma)$ are parameters. We use this function to define the problem of predicting route sequences as a supervised learning task.

Given a set of observed route sequences $\{r_j\}_{j=1}^{J}$ and accompanying data $D_j$, we define as our loss function the average score over all stop sequences:

$$\ell(\theta) = \frac{1}{J} \sum_{j=1}^{J} \text{score}(f(S_j, D_j, \theta), r_j).$$

Our problem is then to determine the parameter values $\theta^*$ so that

$$\theta^* \in \arg\min_{\theta \in \Theta} \ell(\theta),$$

in which $\Theta$ is an appropriate parameter space. As the loss function is nondifferentiable, we resort to a
Algorithm 3 Bayesian optimization of the loss function

**Input:** Stops $S$, station $s_1$, data $D$, features $\phi(\cdot)$, $I_0$, $I$, $J$, $p(\ell)$

Compute $I_0$ random initial points $\mathcal{F} = \{(\theta_1, \ell_1), (\theta_2, \ell_2), \ldots, (\theta_{I_0}, \ell_{I_0})\}$

$\ell^* \leftarrow +\infty$

for $i = I_0, \ldots, I$ do

Update posterior distribution $p(\ell|\mathcal{F})$

$\theta_i \leftarrow \arg \min_{\theta \in \Theta} \hat{f}(p(\ell|\mathcal{F}))$  \hspace{1cm} $\triangleright$ Minimize acquisition function

for $j = 1, \ldots, J$ do

Compute predicted route sequence $\hat{r}_{ij} = f(S_j, D_j, \theta_i)$  \hspace{1cm} $\triangleright$ By using Alg. 1

end for

Compute loss function $\ell_i = \frac{1}{J} \sum_{j=1}^{J} \text{score}(\hat{r}_{ij}, r_j)$  \hspace{1cm} $\triangleright$ $r_j$ is the observed route sequence

if $\ell_i < \ell^*$ then

$\ell^* \leftarrow \ell_i$

$\theta^* \leftarrow \theta_i$

end if

$\mathcal{F} \leftarrow \mathcal{F} \cup \{(\theta_i, \ell_i)\}$

end for

return $\theta^*$

---

gradient-free method. Moreover, it is computationally expensive, so that using finite differences or random-search-based methods such as genetic algorithms will be overkill. We then apply Bayesian optimization, which is a technique designed for searching global optima of computationally expensive functions [Frazier, 2018]. In the experiments, we have used the expected improvement as acquisition function. We have also tested the upper confidence bound acquisition function, but it produced worse results in preliminary experiments. Alg. 3 details the Bayesian optimization algorithm used to minimize the loss function (3). In Alg. 3 $p(\ell)$ is a prior probability distribution over the loss function.

### 4. Results

In the computational experiments, we split the available dataset in training and testing samples. We then applied the Bayesian optimization algorithm (given in Alg. 3) for a fixed number of iterations and retained the parameter values which achieved the lowest average score over the training sample. The model has seven parameters, five $\beta$s directly related to the features in the cost function (1), the discount factor $\alpha$, and the penalty parameter $\gamma$. All parameters were assumed to be bounded in the $[0, 1]$ interval.

Regarding data treatment, we applied almost no preprocessing. Since we are interested in predicting high quality route sequences, we filtered only the 2718 high quality routes and split in approximately 70% training sample (1903 routes) and 30% testing sample (815 routes). We converted the departure times and
time windows to POSIX time (in seconds), so that we could appropriately track the route clock. The data were not normalized. In preliminary experiments normalization did not exhibit better performance, although we do think more experiments are needed to evaluate if normalization has an effect.

We ran Bayesian optimization for 30 iterations, shown in Fig. 1. We used the bayesian-optimization Python library (Nogueira, 2014), with expected improvement as an acquisition function. We started in a greedy solution with parameter values $\beta_1 = 1$ and all other equals zero, whose average score is 0.1028. Each iteration takes around 40 to 60 minutes in an Intel i5-9600K CPU with 16 GB RAM machine. The parameter values with lowest average score (= 0.09055) was found in iteration 26, with values $\beta_1 = 1.0, \beta_2 = 0.0, \beta_3 = 0.3047, \beta_4 = 0.0, \beta_5 =: 0.0, \alpha = 0.377, \gamma = 0.0$. Notice that travel time is the main explanatory feature, while features related to time windows were zeroed. We think this is an artifact of this particular solution, and suspect better solutions should attribute some weight to features related to time windows. Parameter $\alpha$ indicates that drivers seem to look some steps into the future, and therefore are not myopic. Fig. 2 shows the distribution of the scores of predicted route sequences in the holdout sample, while Table 1 shows score summary statistics.

![Figure 1: Iterations from the Bayesian optimization algorithm over the training sample. Lowest average score (= 0.09055) was found in iteration 26. Peaks are due to the Bayesian optimization exploration of the parameter space](image)

5. Conclusions

We have developed an approach to predict route sequences based on inverse reinforcement learning and Bayesian optimization. We used a parameterized linear model in the cost function and tried to learn the parameters in order to minimize the loss function defined as the average prediction score. In preliminary
Figure 2: Score distribution of 815 predicted route sequences in the holdout sample

Table 1: Score summary statistics in the holdout sample with 815 routes

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.0958</td>
</tr>
<tr>
<td>Median</td>
<td>0.0823</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.00447</td>
</tr>
<tr>
<td>Maximum</td>
<td>0.563</td>
</tr>
<tr>
<td>Std-dev</td>
<td>0.0604</td>
</tr>
</tbody>
</table>

experiments, we achieved a mean and median prediction scores of 0.0958 and 0.0823 in a holdout sample, respectively (zero is the perfect score). We think the main limitation of our approach is that the loss function is computationally expensive and nondifferentiable, so that it is hard to use more powerful approximation architectures such as artificial neural networks. On the other hand, it is worth noting that we used simple features in the linear model. We think better engineered features may potentially improve the prediction score.

References


Nogueira, F. (2014–). Bayesian Optimization: Open source constrained global optimization tool for Python. URL: https://github.com/fmfn/BayesianOptimization

Pitombeira-Neto


Article V

A Two-stage Traveling Salesman Model for the Last-Mile Routing Problem

Qiu Runwen, Wang Yueting and Xie Yi

Source code is available on GitHub:
https://github.com/runw99/Two-Stage-TSP-in-RC
A Two-stage Traveling Salesman Model for the Last-Mile Routing Problem

Qiu Runwen
Department of Business, South China University of Technology
382 Zhonghuan Road East, Panyu District, Guangzhou Higher Education Mega Centre, Guangzhou, China

Wang Yueting, Xie Yi
Department of Business, South China University of Technology
381 Wushan Road, Tianhe District, Guangzhou, China

Abstract

Our model finds the best zone sequence firstly and then the best stop sequence within each zone because we realize that high-quality routes tend to visit the stops in the same zone before moving onto the next zone. Decision tree model is applied to reveal the critical features of high-quality paths and one of the most important features is total travel time. Therefore, we take the total travel time and the total longitude latitude distance as the optimization objective and solve the challenge as a special TSP.

First, the latitude and longitude of a zone is defined as the average latitude and longitude of all stops in this zone and we apply the greedy algorithm to find initial solutions for the best zone sequence. If time permission, Local Search (LS) algorithm is used to find a better solution for the first stage problem. Secondly, the stops are connected in each zone using Dynamic Programming (DP) or LS. Finally, we joint the last stop in the zone with the nearest stop in the following zone, which is get from the zone sequence given by the first stage problem.

The final score achieves about 0.061 in 2718 high score routes.

1. Introduction

First of all, we try to understand what "high-quality" is. From the point of view of scores, the key of the problem is to obtain the characteristics of the high-quality route. We use various information of routes to predict route_score which is regarded as a classification problem. The accuracy of the model only reaches about 0.70 but it reveals the importance of different features affecting the route scores. The features are listed in descending order according to their importance (Table 1).
Table 1: the important characters of high-quality routes

<table>
<thead>
<tr>
<th>features</th>
<th>rank of the importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>total travel time</td>
<td>1</td>
</tr>
<tr>
<td>the number of missing time window</td>
<td>2</td>
</tr>
<tr>
<td>the number of time windows(package norepetition)</td>
<td>3</td>
</tr>
<tr>
<td>station code</td>
<td>4</td>
</tr>
<tr>
<td>executor capacity(cm3)</td>
<td>5</td>
</tr>
</tbody>
</table>

Next we try to understand how a stop chooses the next one (Figure 1). We split 2718 high-quality routes into 379444 record whose index is “routes id − stop id”. The characteristics of stops in high-quality routes are: (1) when there remains stops in the same zone, 99% of the next zones will be the same, and 98% of the next stops are selected from the nearest top 5 stops, (2) Otherwise, 98% stops will choose the nearest top 7 zone (the distance used is the minimum distance between two sets). After selecting the zone, 98% of the next stop will choose the stop from the nearest top 6.

![Figure 1: How a stop chooses the next stop in history data](image)

Therefore, we decide to solve the problem as a special TSP from the perspective of optimization and use the total travel time and the total longitude latitude distance as the optimization objective. The optimization
algorithm can be addressed in two stages:

1. Use TSP algorithm to find the optimal zone sequence.
2. Use TSP algorithm to find the optimal stop sequence.

Since there is no training stage, 2718 high-quality routes in the history is used to evaluate. The final score is about 0.061.

2. Literature Review

Travelling salesman problem (TSP) is a classic NP-hard problem, and the computing resources needed to find the optimal solution increase explosively with the increase of number of "city". Therefore, exact algorithm is almost only used in small-scale TSP (Wikipedia contributors (2021)). Recent academic hot spot research has focused on heuristic and approximation algorithms, and most of the problems solved are variants of TSP (Cheikhrouhou & Khoufi (2021)).

Considering that:

- We regard the stop scheduling problem in a zone as TSP. In historical data, the largest scale is 44 and the average scale is only 8, which belongs to small-scale TSP. The optimal solution of TSP can be obtained by using exact algorithms, such as branch and bound algorithms, dynamic programming and so on.

- It is significant to take time window into account. When the total travel time is optimized to a certain extent, the shorter travel time might cause more missing time windows which leads to a bad score. So it does not necessarily need to be optimized to the minimum. Our experiments also show this.

Therefore, we finally use greedy algorithm, DP and LS to solve TSP.

3. Methodology

During the model building period, we use information of 100 routes which are randomly picked up from the data in model_build_inputs to compare the effects of other modules quickly. The information of routes our model use is the latitude, longitude, zone id of every stop, and travel time between every two stops. According to our data analysis, scores of the sequences that visit stops in the order of zone are higher than those do not most of the time. Therefore, our model finds the best zone sequence firstly, and then the best stop sequence within each zone. These two stages are both regarded as TSP solved by greedy algorithm, DP and LS.
3.1. First stage: get zone sequence

The first stage of our model is to find a zone sequence that can go through all the zones with the total distance as short as possible. The latitude and longitude of a zone is defined as the average latitude and longitude of all stops in this zone. The distance between zone A and zone B is calculated as the curve physical distance on the surface of the earth as follows (Veness (2019)):

\[
\text{physicaldistance}_{AB} = 2R \times \arcsin \sqrt{\sin^2 \frac{\text{lat}_A - \text{lat}_B}{2} + \cos \text{lat}_A \times \cos \text{lat}_B \times \sin^2 \frac{\text{lon}_A - \text{lon}_B}{2}}
\] (1)

Where

- \( R \) is the radius of the earth.
- \( \text{lat}_A, \text{lon}_B \) is the latitude and longitude of zone \( A \); \( \text{lat}_B, \text{lon}_B \) is the latitude and longitude of zone \( B \).

We use padding to solve the problem that zone id of some stops are missing. For example, when zone id of stop A is unknown, if stop A is nearest to stop B and zone id of stop B is known, we fill the zone id of stop A with the zone id of stop B.

Because of the time limit, we firstly use greedy algorithm to find a good enough solution and then use LS to find a better solution for the first stage problem. The pseudocode is as follows (Algorithm 1.2):

\begin{algorithm}
\caption{greedy for zones}
1: Define a list called initialSol for initial solution
2: for each zone \( z \) do
3: if \( z \) is nearest to the station then
4: Add \( z \) to initialSol
5: end if
6: end for
7: while the length of initialSol is smaller than the number of zones do
8: for each zone \( z \) and \( z \) is not in initialSol do
9: if \( z \) is nearest to the last stop of initialSol then
10: Add \( z \) to initialSol
11: end if
12: end for
13: end while
\end{algorithm}
Algorithm 2 LS for zones

1: Define a list called seqList for better solutions
2: Add initialSol to seqList
3: Set bestSol to initialSol
4: set cnt to 0 for the number of times tempSol is worse than bestSol
5: while the length of seqList is greater than 0 and cnt is less than 2 do
6: Set tempSol as the last zone sequence of seqList
7: Delete tempSol in seqList
8: // exchange and get the neighbor
9: for each zone z1 do
10: for each zone z2 and z2 is not z1 do
11: Exchange the positions of z1 and z2 in tempSol and get neighborSol
12: if the distance of neighborSol is shorter than tempSol then Add neighborSol to seqList
13: end if
14: end for
15: end for
16: Set the popSol to the last solution of seqList
17: if popSol is better than bestSol then
18: Set bestSol as popSol
19: else
20: Increase cnt by 1
21: end if
22: Remove popSol from seqList
23: end while

3.2. Second stage: get stop sequence in every zone

As for the second stage problem, we connect all the stops in each zone using DP or LS at first. To ensure the speed of our model, LS is used to get a good enough solution when the number of stops in one zone is more than 9, otherwise, DP is applied. Setting the boundary value to 9 makes our model find a good enough solution in a short period of time. Then we connect the last stop in zone A with the nearest stop in zone B which follows zone A according to the zone sequence given by the first stage problem. DP algorithm is already very common and mature, so we will not introduce here.

The optimal objection is the sum of distance between each pair of adjacent stops, and distance is calculated by weighting travel time and physical distance. Travel time is from travel_times.json, and physical distance is calculated by Formula[1]. For the reason that travel time and physical distance both take a crucial part in the score of a sequence, we use weightings to combine these two objectives together and regard it as the objective of DP and LS. Experimental results show that the best weightings are 0.7 for travel time and
0.3 for physical distance. The optimization model is as follows, where \( n \) is the number of stops in zone A. And if stop \( j \) follows stop \( i \), \( x_{ij} \) equals to 1, otherwise 0.

\[
\max Z = \sum_{i \in A} \sum_{j \in A, j \neq i} x_{ij}d_{ij} \tag{2}
\]

\[
\begin{align*}
    d_{ij} &= 0.7 \times \text{traveltime}_{ij} + 0.3 \times \text{physicaldistance}_{ij} \\
    \sum_{j \in A, j \neq i} x_{ij} &= 1 \quad \forall i \in A \\
    \sum_{i \in A, i \neq j} x_{ij} &= 1 \quad \forall j \in A \\
    u_i - u_j + nx_{ij} &\leq n - 1 \quad \forall i, j \in A | 1 < i \neq j \leq n \\
    x_{ij} &\in \{0, 1\} \quad \forall i, j \in A \\
    u_i, u_j &\in \mathbb{R}
\end{align*}
\tag{3}
\]

The pseudocode is as follows (Algorithm 3):

**Algorithm 3** hybrid algorithm for stops

1: Define a list called totalSeq for stops from the same route
2: Define a list called stopList for stops from the same zone
3: Add station to totalSeq
4: Set nextZone to the first zone of zone sequence
5: **while**
6: \hspace{1em} **do** Set stop1 to the last stop of totalSeq
7: \hspace{2em} Set stop0 to the stop nearest to stop1 of nextZone
8: \hspace{2em} Set stopList to the stops which is in the same zone as stop0
9: \hspace{2em} Use greedy to get the initial solution of stopList
10: \hspace{2em} **if** the length of stopList is greater than 9 **then**
11: \hspace{3em} Use LS to solve TSP of stopList based on initial solution and add the solution to totalSeq
12: \hspace{3em} **else**
13: \hspace{4em} Use DP to solve TSP of stopList based on initial solution and add the solution to totalSeq
14: \hspace{2em} **end if**
15: \hspace{2em} Set nextZone to the next zone of stop0 in zone sequence
16: \hspace{1em} **end while**

4. Results and Conclusions

4.1. Performance of model

In our experiment, we aim to minimize the sum of travel time and physical distance without time window constraint, whose weight is 0.7 and 0.3 respectively. We choose greedy algorithm to get initial zone sequence, then apply LS to find a better solution. After that, hybrid algorithm is used to decide the stop sequence...
in each zone and the threshold is 9 (as we mentioned in 3.2). The model gives proposed sequences for the 2718 high-quality routes in \textit{model.build_inputs}. We compare the proposed sequences between the actual sequences and calculate the scores, the average score is about 0.061. Figure 2 shows the score of each route. Calculation time is another index of our model’s performance, which is about 6243.5 seconds.

![Figure 2: score of each route on 2718 routes](image)

### 4.2. Features of well or bad-performed routes

Figure 2 shows the distribution of the route-level scores. For instance, the point (680, 0.036260) means that the route whose score is 0.036260 ranked 680th in ascending order among 2718 routes. We define the routes whose scores obtained by our model are less than the first quartile (the first quartile = 0.036260) as well-performed routes. According to the score distribution, the routes reaching more than 0.125 are regarded as bad-performed routes. Especially, there are three routes whose scores are above 0.35 while the rest of routes are all less than 0.26, which is named as extremely-bad-performed routes. We compare 2718 all routes, 680 well-performed routes, 147 bad-performed routes and 3 extremely-bad-performed routes in two aspects: the number of stop they visit and the total number of packages in the route.

- **Number of Stops**

  All routes and well-performed routes are similar with regard to the distribution about the number of
It’s obvious that bad-performed routes, especially the extremely-bad-performed routes, visit less stops compared with well-performed ones.

Figure 3: the number of stops in different kinds of routes

<table>
<thead>
<tr>
<th>kind</th>
<th>all routes</th>
<th>well-performed routes</th>
<th>bad-performed routes</th>
<th>extremely-bad-performed routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>140.60</td>
<td>138.90</td>
<td>128.73</td>
<td>40.67</td>
</tr>
<tr>
<td>min</td>
<td>38.00</td>
<td>50.00</td>
<td>38.00</td>
<td>38.00</td>
</tr>
<tr>
<td>25%</td>
<td>118.00</td>
<td>116.00</td>
<td>97.50</td>
<td>39.50</td>
</tr>
<tr>
<td>50%</td>
<td>144.00</td>
<td>142.00</td>
<td>134.00</td>
<td>41.00</td>
</tr>
<tr>
<td>75%</td>
<td>165.00</td>
<td>162.00</td>
<td>157.50</td>
<td>42.00</td>
</tr>
<tr>
<td>max</td>
<td>222.00</td>
<td>222.00</td>
<td>209.00</td>
<td>43.00</td>
</tr>
</tbody>
</table>

- Number of Packages
  As for the number of packages, well-performed routes visit more stops, but deliver less packages ranging from 150 to 299. It is interesting to find out that bad-performed routes have more packages to deliver but visit less stops. extremely-bad-performed routes deliver at least 184 packages and the average number of packages is about 195, which is appreciably large because their travel time is almost as half long as all routes.
Figure 4: number of packages in different kinds of routes

Table 3: descriptive statistics about the number of packages

<table>
<thead>
<tr>
<th>kind</th>
<th>all routes</th>
<th>well-performed routes</th>
<th>bad-performed routes</th>
<th>extremely-bad-performed routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>239.72</td>
<td>234.75</td>
<td>240.51</td>
<td>194.67</td>
</tr>
<tr>
<td>min</td>
<td>150.00</td>
<td>150.00</td>
<td>163.00</td>
<td>184.00</td>
</tr>
<tr>
<td>25%</td>
<td>218.00</td>
<td>213.75</td>
<td>214.50</td>
<td>189.50</td>
</tr>
<tr>
<td>50%</td>
<td>241.00</td>
<td>235.00</td>
<td>241.00</td>
<td>195.00</td>
</tr>
<tr>
<td>75%</td>
<td>264.00</td>
<td>258.00</td>
<td>265.50</td>
<td>200.00</td>
</tr>
<tr>
<td>max</td>
<td>304.00</td>
<td>299.00</td>
<td>304.00</td>
<td>205</td>
</tr>
</tbody>
</table>

4.3. Future work

- Improve initial solution

As shown in the figure, the initial solution given by greedy algorithm affects the final solution of LS and DP to some extent. As for LS, if 1234 is the best sequence and 2341 is the initial solution given by greedy algorithm, LS may not get the best solution after limited number of exchanges. As for DP, because the first stop is wrong, DP can’t get the best solution anyway. In short, the method used to get the initial solution has a great influence on the final solution, and using other algorithms instead of greedy maybe better.
• Consider time window constraint
  Although we find that most of high-quality routes never miss time windows, the score will increase when we adjust the order based on the solution that our model gives to provide from missing time window. So our model don’t consider time window, that maybe one way to improve the performance.

• Use machine learning
  The problem is addressed as a TSP without machine learning. Adding some machine learning method especially Seq2Seq may decrease the score further.
References


Article VI

Data-Driven Vehicle Routing in Last-Mile Delivery

Okan Arslan and Rasit Abay

Source code is available on GitHub:
https://github.com/rasitabay/lastmile_challenge
Abstract

This report presents a methodological approach developed for the Last Mile Routing Research Challenge organized by Amazon and supported by the MIT Center for Transportation & Logistics in 2021. We develop a prescriptive method based on rules that are extracted through descriptive analysis of the data. The method involves solving the traveling salesman problem on a transformed graph. In this transformation, we only use the zone_ids for modifying the travel times between node pairs. The method is effective in generating high-quality solutions. The average score that the model achieved when tested locally is 0.0367 on the 1107 routes from the dataset, which share similar characteristics to the ones in the model apply phase.

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 in operations research. Different than distance-minimizing objective in the TSP, the aim in this challenge is to generate tours similar to those that were historically executed by the Amazon drivers. To this end, we present descriptive analysis of the data to help better understand the driver’s choices and develop prescriptive methods for generating high quality routes. In particular, our main approach is (1) to deduce rules through descriptive analytics, (2) to develop a network transformation guided by these rules and (3) to solve the classical TSP on the transformed network. We are expecting that in the model apply phase, our approach generates a feasible route for every problem instance, and yields an average score of 0.0367. We have three modeling strategies that governed choice of methods:

(I) **Effectiveness**: We take the score as the primary factor affecting our model selection.
(II) **Interpretability:** We prefer to maintain the complexity of the model low (in terms of data requirement) in order to make it easy to interpret the factors affecting the score.

(III) **Flexibility:** We prefer to keep the model flexible without much dependence on excessive input data.

Our method is effective because it achieves a competitive score on the provided dataset. It is also interpretable and highly flexible because it involves solving a TSP on a modified graph using only the travel time and zone_id data. There is no additional data requirement including the package dimensions, delivery time windows or service times. This flexibility also implies generalizability of the method to similar datasets.

In Section 2, we review the related literature. We then present our methodological approach in two parts. We first present a descriptive analysis in Section 3 and develop a network transformation approach in Section 4. Results and conclusions are presented in Section 5.

2. **Literature Review**

In this section, we review the methods for solving the TSP, which is one of the most touted problems in operations research. 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. The cost of arc $(i, j) \in A$ is $d_{ij}$. Let 0 represent the depot, and nodes $N \setminus \{0\}$ represent the customers. In this report, we use *node* and *stop* interchangeably. If $d_{ij} = d_{ji}$ for all $(i, j) \in A$, the problem is a directed TSP. It is undirected TSP, otherwise.

There is a multitude of both exact and approximate solution approaches in the literature (Applegate et al., 2011). In this challenge, the objective is the minimization of the average score, which requires the actual visit sequence of the customers to be known. Therefore, this objective function cannot be evaluated when solving the problem and the lower bounds on the objective function are not relevant in this context. On the other hand, any feasible solution for the TSP is also a feasible solution for the problem considered here. Given the limited time available to generate the routes, heuristics and metaheuristics are potentially the best candidates for generating such feasible tours. One of the best performing heuristics for the TSP is developed by Lin & Kernighan (1973) and an effective implementation is presented by Helsgaun (2000). For a review of heuristic algorithms developed for TSP, we refer the reader to Laporte (1992). Several metaheuristics are also developed for the TSP such as genetic algorithm (Potvin, 1996), tabu search (Knox, 1994), simulated annealing (Pepper et al., 2002; Wang et al., 2009), variable neighborhood search (Carrabs et al., 2007; Hore et al., 2018), among others. Kotthoff et al. (2015) report that the LKH algorithm (Helsgaun, 2000) and the genetic algorithm with a powerful edge assembly crossover operator (Nagata & Kobayashi, 2013) are the two most powerful algorithms for inexact solution of the TSP.

The TSP solution methods considered in the literature have a well-defined objective function, which is the minimization of the cost (in terms of a measure such as time, cost or distance). Our approach is different from the existing studies because we consider an objective function that is unknown during the solution of
the problem. In our approach, we transform the input graph in order to model the driver choices and use
the TSP as a subroutine for generating solutions.

3. Methodology, Part 1: Descriptive Analytics

The dataset contains 6112 routes from five major locations and their surrounding cities: Los Angeles, Seattle, Chicago, Boston, and Austin. The routes are classified as high, medium and low in terms of quality. The summary of the data is presented in Appendix A. Five remarks are in order, which guide the development of our prescriptive method.

Remark 1. *In the dataset, 4.78% of the stops have time windows that can impact the routes.*

Remark 2. *In the dataset, the travel time constitutes one of the most important features.*

Remark 3. *In the actual routes, the zone_ids of two consecutive nodes are the same 85.2% of the times.*

In the dataset, the ‘zone_id’ property of a node is of the following format: $L-M.PR$, where $L$ and $R$ are letters and $M$ and $P$ are numbers. We refer to each of these four entries as a token.

Remark 4. *In the actual routes, the zone_ids of two consecutive nodes change only in the 3rd or 4th token 11.3% of the times.*

Remark 5. *In the actual routes, the first and the last stops are not necessarily close to the depot.*

The justification of these remarks are optional and are presented in the remainder of this section. Interested readers can continue reading or they can directly proceed to Section 4 without loss of continuity.

3.1. Time windows

Time window is a critical property for algorithm selection. TSP with time windows is computationally more challenging than the classical TSP (without time windows). Therefore a thorough investigation of the data is necessary to see if it is indeed necessary to model the time windows in the solution method. In the provided dataset, there are 1,457,175 packages delivered on 6112 routes. Among these packages, 113,993 have time windows, which corresponds to 7.82% of the total number of packages. We now make two observations: (1) in the actual competition, there will not be any delivery reattempts of a package, and (2) in the dataset, the latest start time of any time window at a given stop is never after the earliest end time of any time window at the same stop. Considering these two observations, it is suboptimal to visit a stop more than once and all packages at the same stop must be delivered in a single visit. Thus, we consider the time windows per *stop* instead of per *package*. There are a total of 898,415 stops visited on 6112 routes. The number of stops with at least one package with a delivery time window is 59,172, which corresponds to 6.58% of the total stops. The earliest delivery time requirement of 32,711 of these stops are automatically satisfied,
because the soonest time that the delivery vehicle can arrive at the customer’s location (considering the departure time and the travel time from the depot directly to the stop) is after the start of the time window. The latest delivery time of 28,277 of these stops are also automatically satisfied because the time that the delivery vehicles serve the last customer on the actual routes is before the end of the time window for these stops. When we exclude those stops that without a need for special treatment for their start and end of time window requirements, 42,989 stops remain, which corresponds to 4.78% of the total customers. This justifies Remark 1.

3.2. Travel times

When the objective is defined as the minimization of the tour duration and the time windows are not considered, the problem is then a TSP. Using only the travel times, we solved the TSP problem for every route, which gave a competitive score of 0.0795. This naive approach demonstrates that the drivers prefer shorter route durations and travel time is an important measure in route selection. This justifies Remark 2.

3.3. Zones

Zoning is critical for delivery. Delivery region is usually partitioned into multiple zones considering several factors such as the geographical and structural differences between locations. In the 6112 routes provided, there are an average of 21.04 zone_ids per route including the depot. The average number of times that the drivers change region between two consecutive visits is only 23.61. In other words, the drivers generally visit the stops in the same zone_ids before changing zones. A sample route is plotted in Figure 1.

The observation that the drivers generally visit the customers in the same zone_id before going into another zone is an important observation. But the zone_id property contains more information than this. In particular, as a general rule of thumb, the more similar two zone names are, the closer they are geographically to each other, similar to military grid reference system∗. We now recall the definition of a token. In the dataset, the ‘zone_id’ property of a node is of the following format: L-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 subregions. Let $x_{k}^{ij} = 1$ if the $k^{th}$ token of the zone_id of nodes $i$ and $j$ are different, and 0 otherwise.

**Definition 1.** Given two nodes $i, j \in N$, we define $\eta_{ij} = \sum_{k=1}^{4} x_{k}^{ij}$ 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 1 shows the average number of times that a token difference is observed on 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. This justifies Remark 3. 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 stop in zone $A-1.2A$ and continues with

Figure 1: Map of actual route ‘RouteID_62955f5e-57a2-4425-bfe6-6989d8dae565’. Stops are marked by their zone_ids and the route is shown in blue line. The depot is not shown on this map. Google Map data ©2021
another stop in zone \textit{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.

<table>
<thead>
<tr>
<th>Token difference ($\eta_{ij}$)</th>
<th>Average number of times per route</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>124.38</td>
</tr>
<tr>
<td>1</td>
<td>17.38</td>
</tr>
<tr>
<td>2</td>
<td>1.88</td>
</tr>
<tr>
<td>3</td>
<td>0.34</td>
</tr>
<tr>
<td>4†</td>
<td>2.02</td>
</tr>
<tr>
<td>Total†</td>
<td>145.99</td>
</tr>
</tbody>
</table>

† Excluding the arcs from the depot

When the token difference equals 1 in Table 1, the two zone ids are different only in $i^{th}$ token for $i = 1, \ldots, 4$. Table 2 shows the average number of times that $i^{th}$ token is different in consecutive stops 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 2). For example, the vehicle goes from zone A1.1A to A1.2A. They go to another zone that changes only in the 4th token 5.22 times. This corresponds to 11.3\% of the stops, which justifies Remark 4.

<table>
<thead>
<tr>
<th>Token #</th>
<th>Average number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1$^{st}$</td>
<td>0.02</td>
</tr>
<tr>
<td>2$^{nd}$</td>
<td>0.81</td>
</tr>
<tr>
<td>3$^{rd}$</td>
<td>11.33</td>
</tr>
<tr>
<td>4$^{th}$</td>
<td>5.22</td>
</tr>
<tr>
<td>Total</td>
<td>17.38</td>
</tr>
</tbody>
</table>

3.4. Depots

The average travel time from the depot to the first stop is 1811.4 seconds, the travel time from the last stop 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. This signifies the importance of these two trips. We therefore further investigate the first and the last stops on the route from the depot. In particular, we sort the stops with respect to their distance from the depot, and plot the distribution of

\textit{Arslan and Abay}
the ranks of first and the last stops in Figure 2. The first stop on the route is in top 10% ranking in travel time from the depot in 35% of the routes. Similarly, the last stop 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 stops that are farther away from the depot can be selected as the first and the last nodes. This justifies Remark 5.

4. Methodology, Part 2: Prescriptive Analytics

In this section, we first discuss some preliminary tests to justify the reasons for not considering the time windows in our algorithm. We then present the metaheuristics we used for solving the TSP in Section 4.2 and develop a network transformation using Remarks 2-5 in Section 4.3. Lastly, we present implementation details in Section 4.4.

4.1. TSP with or without time windows?

In our preliminary tests, we solved the TSP instances (by only considering the travel times) with and without time windows. In these tests, considering the time windows did not improve the score. Furthermore, considering the small ratio of the stops in Remark 1 (4.78%) that require special treatment in terms of time windows, we prefer to neglect the time windows in our approach. The TSP with time windows is also computationally more difficult to solve and may lead to infeasible solutions. Since the score only mildly deteriorates when the sequence of the customer visits is locally correct, but not precisely in the same order
4.2. Metaheuristics for TSP

In our methodology, we solve the TSP for every route. Therefore, an effective implementation of a solution methods is critical. Given the limited time we have in the context of this competition, we investigated three online source code repositories [Skiena 2021, Bossek et al. 2021, Operations Research Group Bologna 2021] that compares the computational efficiency of the TSP solution algorithms. Taking into account the licensing requirement of the competition and the competitiveness of the algorithms, we found the genetic algorithm with a powerful edge assembly crossover operator, developed by [Nagata & Kobayashi 2013] and implemented by [Liu 2014] to be the most promising alternative. Its implementation is available online[‡] and is under Apache License 2.0.

4.3. Network transformation

Our network transformation is based on modifying the travel times between stops to discourage an arc from appearing in the solution provided by the metaheuristics. To this end, we penalize the arc costs by multiplying by constant values based on their zone_ids. This aligns with Remarks 2−4. Recall that \( x_{ij}^k = 1 \) if the \( k^{th} \) token of the zone_id of nodes \( i \) and \( j \) are different, and 0 otherwise. Consider graph \( \tilde{G} = (N, \tilde{A}) \), where \( \tilde{A} = A \) and the cost of arc \((i, j)\) is equal to \( t_{ij} \), which is defined as follows:

\[
t_{ij} = \begin{cases} 
    b_1d_{ij} & \text{if } i = 0 \\
    b_2d_{ij} & \text{if } j = 0 \\
    a_1d_{ij} & \text{if } i, j \neq 0, x_{ij}^1 = 1 \text{ and } \sum_{k=2}^{k=4} x_{ij}^k = 3, (i, j) \in A, \\
    a_2d_{ij} & \text{if } i, j \neq 0, x_{ij}^1 = 1 \text{ and } \sum_{k=2}^{k=4} x_{ij}^k = 2, \\
    a_3d_{ij} & \text{otherwise}
\end{cases}
\]

where \((b_1, b_2, a_1, a_2, a_3)\) are ‘discouragement multipliers’ with \( a_1 < a_2 < a_3 \). Parameter \( a_1 \) is the discouragement multiplier for the stops in the same zone. Due to Remark 3, it is convenient to set it equal to 1. Parameter \( a_2 \) is the multiplier between two stops \( i \) and \( j \) whose zone_ids share three tokens including the first token, due to Remark 4. By construction, \( a_1 < a_2 \). Parameter \( a_3 \) is the discouragement multiplier between all other stop pairs when neither of the two stops is the depot. If the tail or head of the arc is the depot, the arc cost \( d_{ij} \) is multiplied by \( b_1 \) or \( b_2 \), respectively. By changing these two discouragement multipliers, we can adjust the importance of the first and the last stop selection on the route, aligned with Remark 5.

[‡] https://github.com/sugia/GA-for-TSP
When the network is transformed, we run the aforementioned genetic algorithm on the corresponding TSP instance to generate the tour.

**Remark 6.** The algorithm presented in this study does not require a learning phase.

### 4.4. Implementation details

In this section, we present details on implementation of the algorithm.

- We adapted the genetic algorithm implementation of Liu (2014) in C++. As parameters of the algorithm, we set \( \text{maxNumOfTrial} = 1, \ Npop = 50 \text{ and } \ Nch = 50 \) (see Liu (2014) for details).

- This algorithm is developed for solving the symmetric TSP; however, the travel times in the dataset are asymmetric. By modifying the input graph, we can solve a symmetric TSP instance in order to find a solution for the asymmetric TSP (see, e.g., Jonker & Volgenant, 1983, 1986).

- We modified the genetic algorithm source code to read distance matrix and added a time limit of 4.42 seconds per instance not to violate the 4 hour running time bound.

- Since the zone ids are critical for the correct calculation of the arc costs, our implementation detects stops with zone id='nan' and assign the closest stop’s zone id.

- If genetic algorithm fails to provide a feasible solution, we run a simple greedy heuristic in order not to return an infeasible tour. In our computational tests, we have encountered 2 such occurrences.

- In our algorithm, we use \((b_1, b_2, a_1, a_2, a_3) = (0, 1, 1, 10, 100)\).

The Pseudo-code is presented in Algorithm 1. Note that, in this report, we present the best version of our algorithm in terms of efficiency, interpretability and flexibility. We have also implemented various other techniques, which did not improve over the method presented here. We report such approaches in Appendix B.

---

**Algorithm 1:** Route generating algorithm

**Input:** `new_route_data.json` and `new_travel_times.json`

**Output:** `proposed_sequences.json`

```plaintext
1 foreach route ∈ allRoutes do // Main loop
2     Create a symmetric TSP instance for the asymmetric network \( \hat{G} \) with arc costs as defined in [1]
3     success := Solve the instance using the genetic algorithm
4     if success !=0 then // Contingency plan
5         Run greedy heuristic on \( \hat{G} \)
6     Append the proposed route to `proposed_sequences.json`
7 return `proposed_sequences.json`
```

Arslan and Abay
5. Results and Conclusions

In the model apply phase, the routes will be high quality and will not contain ‘DELIVERY_ATTEMPTED’ status in any stops. There are a total of 1107 routes in the dataset that satisfies these two conditions and we present scores on this subset of the routes. We tested our algorithm with our baseline parameter settings, 

\((b_1, b_2, a_1, a_2, a_3) = (1, 0, 1, 10, 100)\)

and obtained an average score of **0.0367**. The distribution of the route-based scores is shown in Figure 3. Our algorithm with the same settings results in a score of 0.0403 on all 2718 routes with high quality.

5.1. Analysis for tuning parameters

We randomly generated 494 \((a_1, a_2, a_3)\) 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. The average score of these experiments is 0.0396 and the maximum score is 0.0543, demonstrating the robustness of our algorithm for different parameter settings.

The \(b_1\) and \(b_2\) parameters 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.
5.2. Discussion and avenues for further improvement

Our model scored relatively poorly specifically in Austin. The average scores for Austin, Boston, Chicago, Los Angeles and Seattle regions are 0.0573, 0.0377, 0.0438, 0.0339 and 0.0311, respectively. Another feature is the number of stops on the route. Our model also performs relatively poorly when the number of stops on the route is less than 100.

As mentioned in Section 5.1, we tested several different settings for tuning a parameters. When we consider the best performing parameter setting per route, the score we can potentially obtain reduces to 0.018. Therefore, parameter tuning by techniques such as reinforcement learning is a promising avenue to further improve the score (see, e.g., Kotthoff et al. 2015). We are currently working on training an algorithm for learning the best parameter settings for a given instance.

5.3. Conclusion

We have developed an effective, interpretable and flexible model for a real-world last mile delivery problem. The methodology involves a network transportation and a genetic algorithm. The best version of our algorithm scores 0.0367 on the 1107 routes from the dataset, which share similar characteristics to the ones in the model apply phase. Therefore, our expected performance on the real dataset is 0.0367.
Appendix A. Summary statistics

- There are 2718 high, 3292 medium and 102 low quality routes in the dataset.
- The routes are executed between July 19, 2018 and August 26, 2018.
- There are three types of vehicles in size with capacities of 4.2475 m$^3$, 3.313 m$^3$ and 3.1149 m$^3$.
- Number of nodes per route (including the depot) changes between 33 and 238 with an average of 146.99 per route. The average number of customer stops is therefore 145.99 per route.
- The customer service times change between 39.6 and 521.1 seconds, with an average of 116.6 seconds.
- The average travel time between two stops is 61.5 seconds.
- The average time from the depot to the first stop is 1811.4 seconds, the time from the last stop to the depot is 1857.7 seconds and the time travel time spent between stops is 8972.8 seconds.
- Average vehicle load is %74.2 of their capacity.
- Among 1,457,175 packages in the dataset, 1,446,129 have 'DELIVERED', 11,014 have 'DELIVERY ATTEMPTED' status and only 32 have 'REJECTED' status.
- The routes come from five major locations and their surrounding cities: Los Angeles, Seattle, Chicago, Boston, and Austin. Approximately half of the routes come from Los Angeles.
- There are a total of 17 depots in the dataset.

Appendix B. Other methods considered

Note that one can build various rules out of the zone_id naming convention. We also tested assigning discouragement multipliers according to the distance between the centers of gravity of the zones, which did not help in improving the results. Since we have no information as to how the regions are named, we preferred to follow a general rule. If more information is available on the zone naming convention, one can potentially improve the quality of the solution method.

We tested adding package service time to the arc costs in the graph, which did not improve the score. We also restricted visiting a stop with time windows directly from the depot, if the start of any time windows in the same zone is after the arrival of the vehicle to the zone. However, this did not improve the score.
References


Article VII

A Hybrid Method with Rules and Optimization to Solve the Last-Mile Delivery Problem

Fanyou Wu and Yang Liu

Source code is available on GitHub:
https://github.com/wufanyou/TLab-Last-Mile
Article VII

A hybrid method with rules and optimization to solve the last-mile delivery problem

Fanyou Wu
Department of Forestry and Natural Resources, Purdue University
715 W. State Street West Lafayette, IN, USA

Yang Liu
Department of Architecture and Civil Engineering, Chalmers University of Technology
SE-412 96 Gothenburg, Sweden

Abstract

Using historical data to help route planning is significant since the real world is complicated, and the quality of a route is not only defined by its theoretical cost. This report proposed a two-step method that involved learning history and performing classic solutions to vehicle routing problems (VRP). Specifically, the task of the Last Mile Routing Research Challenge is decomposed into two steps: 1) predicting zone level sequence and 2) perform VRP within a single zone. Our method reaches 0.042 locally based on a train test split. The code can be found https://github.com/wufanyou/TLab-Last-Mile.

1. Introduction

In real-life operations, the quality of a route is not only defined by its theoretical cost. Experienced delivery drivers have an implicit understanding of the complex operational environment in which they serve customers daily. To allow for more reliable, more efficient, and sustainable last-mile delivery, it is critical to leverage this tacit information for route planning. In this report, we proposed a two-step method which involved both learning the history and performing classic solution to vehicle routing problem (VRP).

2. Literature Review

The sequence structure of the route is very similar to language and meets the distributional hypothesis in linguistics - words (stops in this case) that are used and occur in the same contexts tend to purport similar meanings [Harris 1954]. So we can directly apply the state of the art Natural Language Processing (NLP) methods, e.g., transformer and LSTM [Vaswani et al. 2017 Hochreiter & Schmidhuber 1997].
Transformer and LSTM require large dataset sizes and more powerful computing resources (GPUs), which are not feasible to use in this task since no GPUs are allowed, and the dataset size is relatively small. We also tested word2vec [Mikolov et al., 2013], a lighter word embedding method, and found it is still the case that trains a well-performed model is complicated. Finally, we decided to give up machine learning-based methods due to their robustness issues. Automated machine learning (AutoML) might be a solution to prepare and select a robust model, but we have no time to implement those methods.

3. Methodology

It is very natural to formal this problem as a sequence prediction since the dataset contains actual sequence for each stop and each stop belongs to a zone. The drivers might not follow the planned route which might theoretically minimize the whole travel cost but in practical are not feasible or at least not suitable for drivers. So design a algorithm to learn real route pattern is critical. The data contains stop ID and zone ID. Figure [1] is an example of the zone level sequence. The zone ID can be further decomposed into two parts we call it primary zone ID and sub zone ID. The primary zone ID and sub zone ID can be even decomposed by an alphabet and a number.

The inner hierarchical structure makes this problem open to the research. After some of preliminary test, we found that predict zone level sequence and then perform VRP within single zone shows best result. A fine-grained stop level sequence prediction will cause many out-of-bag problem and consequently decrease the accuracy of prediction while a more rough major zone sequence prediction will lose many information. In general, we decompose this problem in two part: 1) predicting zone level sequence and 2) perform VRP within single zone. In this section, we will introduce details for this methods.

We finally submitted a simple sequence prediction algorithm to find the most extended potential zone sequence based on concatenating history zone sequence. Algorithm [1] is the procedure that shows how to obtain predicted order sequence $p$ given history set $\mathcal{H}$ and the new unknown sequence set $u$. Noted $p$ is a ordered sequence and a subset of $u$. We first obtain the most overlapped sequence in history as the initial proposed sequence. Then we walk thought the whole history trajectory and try to concatenate some sequence if they meet the criteria. This simple algorithm aims at using the hierarchical structure of the zone-id. After heavily analyzing those sequences, we found that the primary zone ID could help to improve the predicting performance. Specifically, if the head or the tail of the history sequence is the same as the currently proposed sequence, then concatenate two sequences. To solve the out-of-bag problem of $p$, we use...
average travel time and insert the rest elements in $u$ into $p$.

Algorithm 1: Zone sequence prediction

Input: $h_i \in H$, where $h_i$ is an ordered trajectory and $H$ is the total ordered trajectory set.

Input: $u$ as the unknown sequence set.

Output: $p$ as the predict ordered sequence.

1. $j \leftarrow \arg \max_i |u \cap h_i|$ \{get the most overlapped sequence in history\}
2. $p \leftarrow h_j \cap u$ \{set $h_j$ as the initial predicted sequence, note that $p$ is ordered\}
3. for each $h \in H \backslash \{h_j\}$ do
4. $h \leftarrow h \cap u$
5. if $|h| \geq 1$ then
6. if primary_zone($h |h|$) = primary_zone($p_1$) then
7. $p \leftarrow \text{cat}(h, p)$
8. else
9. $p \leftarrow \text{cat}(p, h)$
10. end if
11. end if
12. end for

After successfully predicted the route from the zone level, we used OR-Tools (Apache License 2.0) to optimize the inner route given a zone. Each zone contains several stops (10 - 20), and this small-scale problem is well studied. We treated it as a classic traveling salesman problem (TSP) and use Dantzig–Fulkerson–Johnson formulation (Dantzig et al., 1954):
\[
\begin{align*}
\min & \sum_{ij} c_{ij}x_{ij} \\
\text{s.t.} & \sum_{ij \in \delta(i)} x_{ij} + \sum_{ji \in \delta(i)} x_{ji} = 2 \text{ for all } i \in V \\
& \sum_{ij \in E(S)} x_{ij} \leq |S| - 1 \text{ for all } \emptyset \subset S \subset V, 2 \leq |S| \leq n - 1 \\
& x_{ij} \in \{0, 1\} \text{ for all } ij \in E
\end{align*}
\]  

Here \( V = \{0..n-1\} \) be the set of nodes and \( E \) the set of edges. \( E(S) \) be the set of edges induced by the subset of vertices \( S \) and \( \delta(v) \) the set of edges in the cut \( (v, V \setminus \{v\}) \). We optimized this problem by a greedy search method (Lawler, 1985). We tested some search options in the OR-TOOLS, and none of them is significantly better than any others. So we left the tool itself to decide which search algorithm to use.

4. Results and Conclusions

<table>
<thead>
<tr>
<th>Zone Route</th>
<th>Stop Route</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSP</td>
<td>TSP</td>
<td>0.092</td>
</tr>
<tr>
<td>Word2Vec</td>
<td>TSP</td>
<td>0.068</td>
</tr>
<tr>
<td>SP</td>
<td>TSP</td>
<td><strong>0.042</strong></td>
</tr>
</tbody>
</table>

Table 1: Comparison of different methods. The score for Word2Vec+TSP is not accurate. SP represents proposed sequence prediction.

We tested two different data splitting leave-one out data splitting and train (80%) and test (20%) splitting. Here we ignored the quality of sequence, since we found that the quality of sequence has less impact on the zone level sequence. The leave one out splitting requires more time to evaluate the performance, and train test splitting is more efficient. In this report, we focus on train and test splitting. Each station sub dataset was randomly divided into 80% for extracting history trajectory and 20% for evaluation. Table 1 shows some methods we tested. Our proposed zone sequence prediction (SP) plus TSP methods work best among those three.

Our methods work best for that route which zone are all shown in the history and become infeasible when the route is new. Also, if the route contains other restrictions, e.g., time window restrictions, our current method is not helpful. However, based on observation of package delivery requirement of the given data, the time window for delivery is often huge, so we conjecture that in the online test environment, ignore time window will not hurt the performance a lot.

Our proposed method is only suitable for this given dataset and has no generalization capacity. In the future, we would like to test more sequence prediction methods based on deep learning.

Wu and Liu  

VII.4
About Us

**Fanyou Wu** is a Ph.D. candidate in Forestry and Natural Resources Department, Purdue University, West Lafayette, USA. He has gained a wealth of experience in the theory of interdisciplinary applications of machine learning techniques and has won many championships in AI competitions organized by leading international AI conferences or research institutes, including the championship of JDD (2019), the championship of IJCAI-Adversarial AI Challenge (2019), the championship of KDD Cup (2020), the second place in NeurIPS 2020 Traffic4cast Competition (2020) and the second place in CVRP 2021 The 2nd Agriculture-Vision Prize Challenge (2021).

**Yang Liu** earned his Ph.D. degree in Transportation Engineering from the School of Transportation at Southeast University, Nanjing, China. Now, he is a postdoctoral fellow in the Chalmers University of Technology, funded by Marie Sklodowska-Curie Actions. He has won three championships of Alibaba’s Tianchi Algorithm Competition, the championship of IJCAI-Adversarial AI Challenge (2019), second place in the NeurIPS 2020 Traffic4cast Competition (2020) and the second place in CVRP 2021 The 2nd Agriculture-Vision Prize Challenge (2021).

Acknowledgement

Fanyou Wu thanks the support of this supervisor Dr. Rado Gazo. Yang Liu is from the Laboratory for Emerging Transportation Systems (LETS) and thanks to his supervisor, Dr. Xiaobo Qu, a well-known professor and an elected Member, Academia Europaea - The Academy of Europe.

References


Article VIII

A Data-Driven Metaheuristic Approach to Predict Delivery Route Sequences

Juan Pablo Mesa, Alejandro Montoya and Raúl Ramos-Pollán

Source code is available on GitHub:
https://github.com/mesax1/Eafit-UdeA-Amazon-Last-Mile-Routing-Challenge
Article VIII

A data-driven metaheuristic approach to predict delivery route sequences

Juan Pablo Mesa
Escuela de Ingeniería, Universidad EAFIT
Carrera 49 N° 7 Sur-50, Medellín, Antioquia, COLOMBIA

Alejandro Montoya
Departamento de Ingeniería de Producción, Universidad EAFIT
Carrera 49 N° 7 Sur-50, Medellín, Antioquia, COLOMBIA

Raúl Ramos-Pollán
Departamento de Ingeniería de Sistemas, Universidad de Antioquia
Cl. 67 N° 53-108, Medellín, Antioquia, COLOMBIA

Abstract

On delivery operations, vehicle routing problems regularly focus on optimizing route solution quality. However, real-life execution of routes differs from the computed routing plan. Therefore, the routing algorithms should use information from historic routes and drivers’ decisions in such a way that the planned route is as similar as possible to the real executed route. We propose a simple yet effective Greedy Randomized Adaptive Search Procedure (GRASP) to tackle the incorporation of historic route data into route planning. Our GRASP includes components in the objective function that take into account decisions observed on historical route sequences. To test the performance of our approach, we performed experiments on a set of 2326 routes. The results show that our algorithm gives acceptable results, with an estimated average similarity score of 0.0411 with respect to the observed routes.

1. Introduction

The first step to construct a method that is able to predict the sequence of a route is to understand the structure of the provided routes. Therefore we perform both a descriptive statistics analysis and a graphical exploration of all the routes. These explorations reveal that in many cases, stops with the same zone ID are visited contiguously before visiting other stops with a different zone ID. For example, Figure 1 shows a specific route where each stop is represented by a colored circle, and each color represents a different zone ID. It is evidenced that all stops from the same zone are visited contiguously, before visiting the next zone. The
statistical analysis of the routes, shows that 62.75% of the routes completely visit the all the stops within a zone before moving on to the next one. Additionally, 89.92% of the routes have 2 or less zone transgressions, which is visiting a stop with different zone ID before finishing the visits to all stops from the same zone.

Considering the evidence of the contiguous visits within each zone, we address the problem of the challenge as a Clustered Traveling Salesman Problem with Soft Time Windows (CTSPSTW). Even though in this problem some stops have a time window \((tw)\), if a stop is visited outside its \(tw\) the solution is still considered feasible. Unlike other TSP variants, the objective is to find the route sequence that resembles the most the observed route sequence, instead of minimizing the total travel cost. To solve this problem, we propose a Greedy Randomized Adaptive Search Procedure (GRASP). To guide the search of the GRASP to the most similar route sequence with respect to the observed route sequence, we use an objective function that is composed of several components that take into account the behavior of the historical route sequences. The rest of this paper is organized as follows. Section 2 discusses a brief literature review. Section 3 describes the methodology. Finally, Section 4 presents the results and conclusions.

2. Literature Review

The literature review is focused on the revision of the use of GRASP for some TSP variants, and to how different metaheuristics in routing problems include components in the objective function to guide the search.

GRASP is a multi-start metaheuristic proposed by Feo & Resende (1995), that consists basically of two phases: randomized construction and local search. The construction phase builds an initial feasible solution. Then, this solution is improved during the local search phase, until a local optima is found. This procedure is performed over several iterations, and the best overall solution is kept as the result.
Several authors have used the GRASP during the past 25 years to solve different routing problems variants, such as the TSP (Marinakis et al. 2005; Goldbarg et al. 2007) or the Capacitated Vehicle Routing Problem (Prins 2009; Marinakis 2012). More specifically, for the Clustered TSP variant, Mestria et al. (2013) proposed several GRASP-based heuristics which included path-relinking procedures. Additionally, Mestria (2018) further improves the results achieved, with the combination of GRASP, Iterated Local Search, and Variable Neighborhood Random Descent algorithms.

In previous studies about different vehicle routing problem variants, the objective function aims to minimize the total travel cost of the route (measured either with respect to time or distance). However, when some vehicle routing problem variants have additional constraints, the objective function may add components that penalize when a constraint is not met. Schneider et al. (2014) use this strategy in the form of a generalized cost function for an electric vehicle routing problem. This function considers total traveled distance, battery capacity violations, time windows violations, and total capacity violations. Similarly, Ha et al. (2020) use this strategy in their TSP with drone genetic algorithm. They calculate a penalization cost with the sum of total operational cost and a weighted penalization determined from constraints violations.

In the literature, there are different works that use penalization components in the objective function to guide the search to feasible solutions. However, to the best of our knowledge, no one has proposed a metaheuristic where the components that make up the objective function help to guide the search to find the sequence of stops that resembles the most the observed sequences from a real vehicle routing operation.

3. Methodology

This section is organized as follows: The first part contains the detailed explanation of the different components that make up the objective function of the GRASP. The next part explains the general structure of the GRASP, with the description of the processes of initial solutions generation and local search phase. Finally, the last part explains how to tune the parameters that are used in the proposed method.

3.1. Objective function

The proposed GRASP tries to minimize an objective function, defined in expression (1) as the sum of normalized total time ($ntt$), plus normalized historic zone headings ($nzh$), plus normalized historic zone angles ($nza$), plus normalized time windows penalization ($ntw$), plus normalized zones time cost ($nzt$), plus normalized zones distance cost ($nzd$). It is important to mention that considering that each component may have different magnitudes, a normalization of each component is done to keep their values in similar magnitudes. For our method, we assume that each component of the objective function has the same importance as the others, therefore, we do not use a weighing factor for each component. Below, we explain in detail each component of the objective function. For this explanation, we introduce the following notation for a set of stops whose route sequence is unknown: $N$ is the set of stops, and $Z$ is the set of zones for these
stops.

\[ o = ntt + nzh + nza +_ntw + nzt + nzd \] (1)

3.1.1. Normalized total time (ntt)

Even though the idea of the challenge is to find routes that resemble the observed route sequences, the motivation to include the total time comes from the background of routing problems, where it is preferable to have routes with short travel times.

To estimate \( ntt \), first we estimate the total time \( tt \), which is the sum of all travel times plus all service times of a route sequence, starting from the delivery station, visiting all the stops once, and returning to the delivery station. The travel times between stops are taken directly from the information supplied by the challenge. To calculate the total service time for a stop we sum the service times of its individual packages. To normalize \( tt \), we divide this value by the total time of a solution with low performance in terms of total time. To compute this solution we use a random furthest neighbor construction heuristic. In this heuristic, we start the solution of a route from the delivery station, and to add the next stop to the solution, this stop is selected randomly from a list of the \( k \) furthest neighbors to the current stop, where \( k \) is a parameter of the heuristic.

3.1.2. Normalized historic zone headings (nzh)

Considering that we represent this problem as a Clustered TSP, where each zone ID represents a cluster, we evaluate the importance of the order of the zones with respect to the similarity score from the challenge. To make this evaluation we use two approaches, where we calculate the score of routes assuming that we have privileged information of the problem. In the first approach, we assume that we know the correct order in which the zones must be visited, and we order the stops within each zone as a random permutation of stops. With this approach, we obtained a variability of 0.01 in score differences for several routes. For the second approach, we assume that we know the correct order of the stops within each zone, and we order the zones as a random permutation. With this second approach, we obtained a variability of 0.1 in score differences for the same routes used in the first approach. This signals that it is much more important to get a correct estimation of the order of the zones, than the correct order of stops within each zone. With this, we considered devising a mechanism to leverage historic information on the zone ordering in previous routes into the objective function. The intuition is that if certain zones have been typically visited in a certain order in the past, the objective function should promote that ordering. The \( nzh \) component captures this intuition.

For each zone we define a zone centroid, which is computed by averaging the latitude/longitude coordinates of the stops within the zone. Figure 2a shows a sequence of zone centroids as recorded in a route. Each zone centroid of a zone is represented with a gray circle. Figure 2b shows the zone centroids of several routes in the city of Austin and the direction of visiting the zones within each route. It can be clearly seen...
that certain parts of the city are always visited in the same order, and we aim at exploiting that information to compute the zone order of a new route. Nearby zone centroids of different routes will almost never fall in the exact same point, so to cope with this variability we decide to build a continuous vector field across the city using historic routes. This way, for any arbitrary point on the 2D plane we can compute a preferred estimated heading by averaging the observed directions of routes around it.

**Computing zone vector fields:** We fill up consecutive centroids in each training route with a sequence of step vectors as represented with gray arrows in Figure 2a. The length of these step vectors is determined by the parameter $\beta$. Our vector field is composed by all step vectors of all consecutive zones observed in the training (historic) routes.

Consider $Q$ a set of historic routes. From this set, we compute all step vectors formed by each sequence of zones (as shown in Figure 2a). This means, $F = \bigcup_{j \in Q} V_j$, where $V_j$ is the set of step vectors associated to the sequence of zones of route $j \in Q$. This set $F$ can be interpreted as a vector field. Each step vector $v \in F$ has an origin $o_v$ and a final position $f_v$.

**Heading vectors:** Using the vector field $F$, we want to estimate, for each zone $Z$ in a new route, the direction towards where a route sequence should go after visiting all stops in $Z$. To this end, we introduce the concept of heading vectors. For each zone centroid of zone $p \in Z$, a heading vector points in the direction where the route sequence should go. Figure 2a shows the heading vectors represented as red arrows. Each heading vector can be estimated using expression (2), where $\beta$ is the length of the step vectors, $D(p,o_v)$ is the distance from zone centroid $p$ to the origin $o_v$ of step vector $v$, and $\alpha$ is a weighing factor parameter. Estimated heading vectors are shown in red in Figure 2a.

$$h(p) = \sum_{v \in F} v \cdot e^{-D(p,o_v) \cdot \alpha}$$ (2)
Observe that as any step vector $v$ gets further away from $p$ its contribution vanishes. We use this fact to efficiently compute $h(p)$ by considering only the step vectors $v$ which are at a maximum radius distance from $p$ using a KDTree algorithm [Bentley 1975]. This maximum distance $max_d$ is calibrated so that $e^{max_d} < 10^{-3}$. The KDTree algorithm uses a binary tree data structure that stores the origin of each step vector. The tree divides the space where the vectors are located in half at each level of the tree, which allows us to find all step vectors $v$ within the range $max_d$ from $p$ with a query for each $p \in Z$.

**Zone order probability matrix:** Finally, to include the information of the heading vectors in the objective function, given $Z$ and $N$ we can estimate a probability matrix $H_{ij}$, that represents the probability that zone $i \in Z$ is visited before zone $j \in Z$. Expression (3) shows how we estimate this probability. $\delta_{ij}$ is the normalized euclidean distance between $i \in Z$ and $j \in Z$. This parameter is normalized so that maximum distance between any pair of zone centroids within the route is 1. $\epsilon_{ij}$ is the normalized cosine distance between the estimated heading vector $h(i)$ at $i \in Z$ and the direction vector going from $i \in Z$ to $j \in Z$. It is important to note that, when there is no historic information available between zones $i$ and $j$, we assign the value $H_{ij} = 0.5$ to avoid any bias regarding the order of these zones.

$$H_{ij} = \frac{1}{2} \delta_{ij} + (1 - \delta_{ij}) \epsilon_{ij}$$  \hspace{1cm} (3)

For a possible route $r$ found by the GRASP (hence the real sequence is not known), we can estimate the binary parameter $w_{ij}$, which is equal to 1 if zone $i \in Z$ is routed before zone $j \in Z$, and 0 otherwise. Therefore, we can estimate $nzh$ with expression (4).

$$nzh = \frac{\sum_{i,j \in Z} w_{ij} \cdot (1 - H_{ij})}{|N|}$$  \hspace{1cm} (4)

**3.1.3. Normalized historic zone angles (nza)**

Similarly to $nzh$, this component also tries to help to find the correct order of the zones, using the information from the heading vectors from the previous section. For a given route $r$ found by the GRASP, we can estimate a vector $u_{ij}$, formed by directly substracting zone centroid of $i \in Z$ from zone centroid of $j \in Z$. Using this vector, we can estimate a deviation angle $a_{ij}$ (in degrees) of vector $u_{ij}$ with respect to the direction of the heading vector for zone centroid of $i$ (which will be the origin of both vectors). To include this information of the deviation angle in the objective function, we introduce the matrix of deviation probability, defined as $A_{ij} = \frac{a_{ij}}{180}$. It is important to note that when $A_{ij} = 1$, it means that there is a high probability that the direction that the route takes after visiting zone $i$ is deviated from the historic order, and when $A_{ij} = 0$, the direction followed by the route corresponds to the estimated historic direction.

Now, using the route $r$, we can estimate the variable $x_{ij}$, which takes the value of 1 if zone $i \in Z$ is
immediately before zone \( j \in Z \), and 0 otherwise. Therefore, we can estimate \( nza \) with expression (5).

\[
nza = \frac{\sum_{i,j \in N} x_{ij} \cdot A_{ij}}{|Z|} \tag{5}
\]

3.1.4. Normalized time windows penalization (ntw)

In the challenge, as we mentioned previously, \( tws \) are soft. However, missing \( tws \) will probably provoke differences with respect to observed routes. Hence, the proposed method penalizes when a \( tw \) is not met.

Given a route sequence \( r \) found by the GRASP, we can estimate the binary variable \( k_{i} \), which is equal to 1 if the stop \( i \in N \) is visited outside its \( tw \), and 0 otherwise. Thus, expression (6) estimates \( ntw \).

\[
ntw = \frac{\sum_{i \in N} tw_{i}}{|N|} \tag{6}
\]

3.1.5. Normalized zones time cost (nzt)

Drivers might take decisions that enables them to execute routes where after visiting all stops in a zone, the next zone to visit is nearby, instead of having to drive longer to get to the next zone. To this end, this component gives information related to average travel time between zones.

To estimate the travel time \( o_{ij} \) between zone centroids of \( i \in Z \) and \( j \in Z \), we use expression (7), where \( N_{i} \) and \( N_{j} \) are the stop sets associated to zone \( i \) and \( j \in Z \), and \( t_{kl} \) is the travel time (given by the challenge) between stop \( k \in N_{i} \) and stop \( l \in N_{j} \).

\[
o_{ij} = \frac{\sum_{k \in N_{i}} \sum_{l \in N_{j}} t_{kl}}{|N_{i}| + |N_{j}|} \tag{7}
\]

Let \( s \) be a zone centroid that represents the location of a delivery station. To estimate the travel time \( o_{i} \) between zone centroids of \( i \in Z \) and \( s \), we use expression (8), where \( N_{i} \) are the set of stops associated to zone \( i \in Z \), and \( t_{k} \) is the travel time (given by the challenge) between stop \( k \in N_{i} \) and \( s \).

\[
o_{i} = \frac{\sum_{k \in N_{i}} t_{k}}{|N_{i}| + 1} \tag{8}
\]

Given a route \( r \) found by the GRASP, we can estimate the variable \( y_{id} \), which takes the value of 1, if zone \( i \in Z \) is visited immediately before or immediately after the delivery station \( s \), otherwise it takes the value of 0. Additionally, we can use the same variable definition of \( x_{ij} \) introduced previously. Hence, we can estimate \( nzt \) with expression (9). The first term of expression (9) corresponds to the sum of travel times between contiguous zones in \( r \), normalized with respect to the maximum travel time between zones. The second term corresponds to the sum of the travel time from \( s \) to the first zone in \( r \), and the travel time from the last zone in \( r \) to \( s \), normalized with respect to the maximum travel time between zones and \( s \).

\[
nzt = \frac{\sum_{i,j \in Z} x_{ij} \cdot o_{ij}}{\max(o_{ij})} + \frac{\sum_{i \in Z} y_{id} \cdot o_{i}}{\max(o_{i})} \tag{9}
\]
3.1.6. Normalized zones distance cost \((nzd)\)

As we explained previously in component \(nzt\), we try to prioritize connecting zones that are nearby in terms of travel time. However, in urban logistics, proximity in terms of travel time is not necessarily proportional to proximity in terms of distance. When a driver looks at a map with all the stops and zones that should be visited, the driver might perceive a distance proximity within zones which could encourage a visit in continuous order. This motivates the use of this component, which is similar to \(nzt\), but it uses a rectilinear distance between zone centroids using the Haversine distance formula.

To estimate the travel distance \(g_{ij}\) between zone centroids of \(i \in Z\) and \(j \in Z\), we use expression (10), where \(N_i\) and \(N_j\) are the stop sets associated to zone \(i\) and \(j \in Z\), and \(c_{kl}\) is the haversine distance between stop \(k \in N_i\) and stop \(l \in N_j\).

\[
g_{ij} = \sum_{k \in N_i} \sum_{l \in N_j} \frac{c_{kl}}{|N_i| + |N_j|} \tag{10}
\]

Let \(s\) be a zone centroid that represents the location of a delivery station. To estimate the travel distance \(g_i\) between zone centroids of \(i \in Z\) and \(s\), we use expression (11), where \(N_i\) are the set of stops associated to zone \(i \in Z\), and \(c_k\) is the haversine distance between stop \(k \in N_i\) and \(s\).

\[
g_i = \frac{\sum_{k \in N_i} c_k}{|N_i| + 1} \tag{11}
\]

Given a route \(r\) found by the GRASP, we can use the same definition of variables \(x_{ij}\) and \(y_{ij}\) introduced previously. Therefore, we can estimate \(nzd\) with expression (12). The first term of expression (12) corresponds to the sum of distances between contiguous zones in \(r\), normalized with respect to the maximum distance between zones. The second term corresponds to the sum of the distance from \(s\) to the first zone in \(r\), and the distance from the last zone in \(r\) to \(s\), normalized with respect to the maximum distance between zones and \(s\).

\[
nzd = \frac{\sum_{i,j \in Z} x_{ij} \cdot g_{ij}}{\text{max}(g_{ij})} + \frac{\sum_{i \in Z} y_{id} \cdot g_i}{\text{max}(g_i)} \tag{12}
\]
3.2. GRASP

Algorithm 1: GRASP with VND. Adapted from [Resende & Ribeiro 2003]

<table>
<thead>
<tr>
<th>Input: max_iter, k</th>
</tr>
</thead>
</table>
| 1 Objective function $f_{o_{\text{best}}} \leftarrow \infty$
| 2 for $i \leq \text{max}_{\text{iter}}$ do |
| 3 Solution $x \leftarrow \text{Random nearest neighbor}(k)$
| 4 $x \leftarrow \text{Variable Neighborhood Descent}(x, \text{neighborhoods})$
| 5 if $f_{o}(x) \leq f_{o_{\text{best}}}$ then |
| 6 $f_{o_{\text{best}}} \leftarrow f_{o}(x)$
| 7 $x_{\text{best}} \leftarrow x$
| 8 return $x_{\text{best}}$

Algorithm 1 describes the GRASP used in this challenge. Line 1, sets the cost of the objective function $f_{o}$ to infinite. Lines (2-8) enters the main loop of the GRASP, which is parallelized. One iteration runs in an available cpu thread. The parameter $\text{max}_{\text{iter}}$ indicates the maximum number of iterations that will be performed, and it is equal to the number of available cpu threads. Then, line 3 of the algorithm constructs an initial solution $x$ using the random nearest neighbor heuristic. Inside this heuristic, to add the next stop to the solution, this stop is selected randomly from a list of the $k$ nearest neighbors to the current stop, where $k$ is a parameter of the heuristic. After the initial solution $x$ is constructed, the algorithm applies the local search phase with a Variable Neighborhood Descent (VND) [Mladenović & Hansen 1997] in line 4. In lines 5 to 7, when a better solution (lower objective function value) is found, the overall best solution is updated. Finally, in line 8, after all iterations are performed, the best solution is returned. The VND in this GRASP uses the following 4 different neighborhoods with a best improvement configuration. Relocate stop relocates the position of one stop at a time within the same zone. Swap stops exchanges the position of two stops from the same zone. Relocate zone relocates the position of a complete zone within all the route sequence, without changing the order of stops inside it. Swap zones exchanges the position of two complete zones, without changing the order of stops inside their respective zone.

3.3. Parameters tuning

The proposed method has 3 parameters that must be tuned based on the data that is available. These parameters are: the list $k$ of nearest neighbors in the randomized nearest neighbor heuristic, the length $\beta$, in meters, of the step vectors for the estimation of heading vectors, and the weighing factor $\alpha$ also for the heading vectors. We tune these parameters during the training phase of the challenge.

To tune $k$ we use a training subset of the routes. For this challenge, the first 500 routes of the model_build_inputs dataset are used as training subset. We evaluate different values of $k$, and we select the $k$ that might give better results in terms of the score values used in the challenge. In this case, we use values of $k = [2, 3, 4, 5]$. For each possible value of $k$, we use the GRASP to solve each one of the routes.
of the training subset. We estimate the score of each route, and finally, we select the $k$ that generates the lowest average score for the training subset.

To tune parameters $\alpha$ and $\beta$ we use all routes in the model_build_inputs dataset. We evaluate different possible combinations of the pair $(\alpha, \beta)$, and we select the pair $(\alpha, \beta)$ that might give the most information from the historic routes. The amount of information given is estimated in terms of percentage of zones estimation and cosine distance estimation. In this case, we use values of $\alpha = [0.01, 0.05, 0.1, 0.15]$ and $\beta = [50, 100, 150, 200, 250, 300]$, which gives us 24 possible combinations of the pair $(\alpha, \beta)$. For each possible pair of $(\alpha, \beta)$, we create a vector field (with all the generated step vectors), then we estimate the heading vector for each zone centroid. For each zone, we calculate the cosine distance between the heading vectors and the observed vectors, which are vectors formed by joining the zone centroids of the route in the correct observed order. Subsequently, the percentage of zones estimation is estimated as the average percentage of zones whose cosine distance is not equal to 0. The cosine distance estimation is estimated as the average cosine distance of each zone, without taking into account the zones whose cosine distance is equal to 0. Then, we filter the pairs of $(\alpha, \beta)$ that have percentage of zones estimation $> 90\%$, and if no pair passes the filter, we lower the value by 10\% until we get pairs that pass the filter. Finally, we select the pair of $(\alpha, \beta)$ with the highest value of cosine distance estimation.

4. Results and Conclusions

To estimate the vector fields, the heading vectors, and the different probability matrices, we used Python 3.8. Particularly for the calculation of the KDTree, we used the implementation on the Scikit-learn library (Pedregosa et al. 2011). The GRASP was implemented in C++, using the C++17 version standard. Additionally, for the calculation of the haversine distances, we used the value of $R = 6372.8$ as the radius of the Earth in kilometers. The parameter max_iter equals the number of threads available in the cpu. With the parameter tuning phase we find that the best randomization parameter for the $k$ nearest neighbor solution constructor is $k = 5$ and the best values to construct the vector fields are $\beta = 200$ and $\alpha = 0.01$. The following results in this section make use of these values.

The performance of the model is evaluated on the set of routes with route_score = High, and no delivery reattempts, taken from the model_build_inputs dataset. This means, the evaluation set comprises 2326 routes (out of the 6112 provided in model_build_inputs) for performance evaluation of our method.

A Leave one out cross-validation strategy is used to create matrices $A_{ij}$ and $H_{ij}$. Figure 3 shows the distribution of the route-level scores obtained by our model on the evaluation set. We obtained an average score of 0.041180, shown in red. The best score for a route is 0.002455, and the worst score is 0.253594. Only 155/2326 routes have scores greater than 0.1. We also evaluated our GRASP on the test dataset (that was provided) with only 13 routes, and we obtained an average score of 0.033296.

From the set of features in the datasets supplied by the challenge, we did not find attributes with similar values shared amongst the set of routes where we performed particularly well or poorly. However, when we
trace the sequence of *zone centroids* from a route (Figure 4), we observe a relationship with the zones orders. For many of the routes where our algorithm performed poorly, the order in which the zones are visited does not follow an easily predictable pattern. With the data that is available, we are not able to explain some decisions with respect to the order of the zones. For example, Figure 4a shows a route with poor score. There is no logic, known to the authors, in the order of the sequence from zones 0 to 11, and the historic information available does not gives useful information for the prediction of that order. Meanwhile, Figure 4b shows a route where we obtain an acceptable score. The decisions to perform this order of the route can be explained with travel times, historic routes information, and proximity of the zones.

We have a method that combines optimization techniques with the use of historical data, which can give acceptable results. An advantage of our method is that it is based on a classic metaheuristic which can be flexible enough to add more components to the search strategies, such as new neighborhoods or other type of constraints that the routing problem might have.
References


Article IX

Delivery Location Dependent Traveling Salesman Problem Using Multi-Head Attention-Based Actor-Critic Method

Ashutosh Nayak and Ahmad Hemmati

Source code is available upon request from the authors.
Article IX

Delivery Location Dependent Traveling Salesman Problem Using Multi-head Attention based Actor-Critic Method

Ashutosh Nayak
University of California, Graduate School of Business (ashnayak@ucdavis.edu)
Davis, CA, USA

Ahmad Hemmati
University of Bergen, Department of Informatics (Ahmad.Hemmati@uib.no)
Bergen, Norway

Abstract

Traveling Salesman Problem with Time Window Constraints (TSPTWC) has been studied extensively in the literature. Traditional TSP considers distance between two locations but does not consider the location of the delivery stops. Amazon routing challenge differs from traditional TSPTWC as it involves 1) information about the delivery stop location (time window constraints and geographical location); 2) information on driver behavior (order in which the stops are visited). We propose a three-step actor-critic reinforcement learning method to solve the routing challenge. In the first step, the delivery stops are divided into clusters. In the second step, we use the weights learned from multi-head attention model to sequence the clusters. In the third step, we use LKH3 algorithm for routing within each cluster. The proposed model is one of the first attempts to use a deep learning based reinforcement learning model to solve TSP with time window constraints while considering the two differences from traditional TSPTWC. We have made the report in a presentation style and tried to reduce the equations.

1. Introduction

Amazon sends millions of packets every day from Amazon warehouses to customers using Amazon logistics. Once the warehouse manager decides the packets for each delivery vehicle, Amazon uses an algorithm and prescribes the routing sequence to the drivers at the start of the route (at warehouse station). Amazon routing challenge aims at finding a methodology to route different amazon delivery vehicles such that each route qualifies as "High Quality" route. The quality of a route is measured by a latent amazon scoring criteria, unknown to the competition participants.
The routing challenge can be considered as solving a set of independent Traveling Salesman problem with Time Window Constraints (TSPTWC). However, since the delivery stops in different routes are spread across same cities, geographical information can be used to provide improved routing solution. TSPTWC considers various objective functions e.g. minimizing total travel time, minimizing time taken to serve last customer, minimizing travel time + time window violation. However, routing challenge deviates from traditional TSPTWC in two major aspects: 1) a delivery stop may contain multiple packages and each package may have different time window constraints while location of the geographical stops is also considered ; 2) drivers might not follow the routes provided by solving traditional objective function as they use their personal experiences and preferences in visiting different localities. to handle (1), we track the number of packages with time window constraint and also track the earliest time window constraints for each stop. To handle (2), we use machine learning to predict the likelihood that a given stop would be visited first by a driver. We do not know why a driver would visit that stop earlier, but historical data (which is used by machine learning) suggests that a particular stop (or locality) is often visited first by multiple drivers (for example, city center is Boston is visited first by almost all the drivers). If a locality is consistently visited first by multiple drivers (even if it was shown by Amazon provided algorithm), it indicates that drivers want to visit that locality early in their tour (otherwise they might have deviated).

We reconcile these differences from traditional TSPTWC in a reinforcement learning model for Amazon routing challenge. We propose an actor-critic model (details in Section 3) that aims at minimizing the total travel time, total time window constraint violation and deviation from proposed routing solution from high quality routes in the training data set. We use deep learning based attention models to build neural networks for actor and critic. While the actor network aims at minimizing the objective function cost (total travel time + total time window constraint violation + deviation from high quality routes), the critic network aims at minimizing the difference between baseline cost function and reward value function from the actor network (critic network helps speeding up the learning process in neural networks). Major contribution of this method include:

- We propose a multi-head attention problem for TSPTWC. This is one of the first works that considers geographical information and past driver behavior for routing.

- The proposed model is agnostic to the location of the delivery stops. Hence, the same model can be easily deployed to other cities.

- The proposed model integrates machine learning with reinforcement learning to take advantage of the past information available about stop (location and stop orders) and current information available about the stops (number of packages and number of time window constraints).

Before we discuss the methodology, we first discuss extant literature used in this project. We focus on literature that have been used for building our proposed model and do not provide an extensive literature on TSP or TSPTWC.

Nayak and Hemmati
2. Literature Review

In this Section, we discuss the main papers we used in our proposed model. We use a three step method as we discuss in detail in the next Section and as shown in Figure 3. Table 2 shows the literature we used in our project. We adopted the code for multi-head attention based reinforcement learning using actor-critic method from Deudon et al. (2018). We use tensorflow 2.0 as against tensorflow 1 in Deudon et al. (2018). Researchers have also used pytorch for the implementation of similar algorithm.

<table>
<thead>
<tr>
<th>Method</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-head Attention</td>
<td>Vaswani et al. (2017), Li et al. (2019), Li et al. (2018)</td>
</tr>
<tr>
<td>2OPT Algorithm</td>
<td>da Costa et al. (2020), Kefi et al. (2016)</td>
</tr>
</tbody>
</table>

Figure 1: Model Build and Model Apply in the Proposed 3 Step Model. We first build the reinforcement learning model and use the weights from the model (in the build phase) to output the sequence in which clusters should be visited (in the apply phase). Once the sequence is obtained, we use LKH3 algorithm to route the stops within a cluster. We then use 2OPT heuristics to improve the current route solution.

3. Methodology

We use a three-step model as shown in Figure 3. On top of the three step model, we apply 2OPT heuristics to stochastically check if we can further improve the routing solution. In this section we provide a detailed explanation of the different steps in Figure 3.
3.1. Data

First, we prepare the data from different routes to use in our attention based Traveling Salesman Problem with Time Window Constraints (TSPTWC). We collect the following information from each stop:

- latitude and longitude: We use latitude and longitude in grouping different stops into \( n = 30 \) clusters.

- adjusted latitude and adjusted Longitude: We subtract the latitude and longitude of the station from the latitude and longitude of a stop to make our model agnostic to exact geographical location and make stops relative to the station for that route. This also ensures that data for building the reinforcement learning is transferable across cities. It is also based on the observation that stops closer to the depots are visited first.

- Number of packages in a stop that have time window constraints within 10 hours of departure time. It allows us to keep a check on time window constraint violation (delay in delivery).

- timeEnd: We also track the minimum upper limit for time window for each stop. For example, if in a stop, the driver has to deliver 4 packets and all have to be delivered by 6pm, 7 pm, 8pm, and 9 pm, \( \text{timeEnd} = 6 \) pm for this stop.

- Number of packages to be delivered in a stop in that route.

- Normalized stop order for stops in training set. We normalize the order in which a stop was visited by the total number of stops in that route. As shown in Figure 2, we observe a pattern in the order in which stops are visited.

![Figure 2: Stop order for different delivery stops in LA. Stop order have been normalized by diving stop order with the number of stops in that route. Yellow indicates the stops that were scheduled later in the route while red color indicates stops that were scheduled early in the route. Figure shows how certain parts of the city are always visited early/ later in the route.](image)
Figure 3: It shows the color of the stops in one of the routes. We can observe that location of the delivery stops depend on the type of locality. In this example, the depot is to the south east of these stops.

After collecting these data from the route data and package data information, we use k-nearest neighbour to find the normalized stop order for each stop (values range from 0 to 1). We use two different values of $K = \{10, 30\}$. The idea is to estimate which stops are visited first by drivers. While KNN with 10 neighbours could find stop order for nearby stops, KNN with 30 neighbours could find if a locality is preferred more by drivers to start their route. Once we build the data-set, we start with first step of our model - clustering.

3.2. Clustering

We group the stops in a route into $n = 30$ clusters using KMeans algorithm. We use latitude and longitude for the KMeans and each stop is assigned a cluster number. Clustering reduces the size of the neural network model we discuss next. Also, routing within a cluster can be done fast and reach near optimal solutions. After clustering, we get a 7 dimensional vector for each cluster which is fed as input to the attention based reinforcement learning model. The 7 dimensions include:

- Adjusted latitude and adjusted longitude
- Number of stops in a cluster
- Number of packets that have time window constraints
- Minimum value of time window constraints (earliest time window end)
- Stop order prediction (normalized) using $K = \{10, 30\}$

Next we discuss the reinforcement learning model used in our approach.
3.3. Reinforcement Learning

We feed the input data (of size = batch size x number of clusters x dimension of data for each cluster) into a multi-head attention model as shown in Figure 4. Left side of the figure shows the multi-head attention encoder. Note that the cluster data is arranged based on the cluster number assigned while clustering. It is agnostic to location of the cluster. We do not use positional encoding in this work. Therefore, the depot (station) for a route could be in any of the 30 clusters.

![Encoder and Decoder network used in Reinforcement Learning model. The encoder takes in 7 dimensional input for 30 clusters and transform it into higher dimension embedding. These embedding are passed through N layers of multi-head attention layers. Each layer has a multi-head attention layer and another embedding layer. Decoder uses the output from the encoder to sequentially list the clusters in the order in which drivers should move for that particular route.](image)

3.3.1. Encoder Network

The encoder and decoder model is adopted from [Deudon et al. (2018)](https://www.deudon.com). In the encoder network, we input the data for a route (clubbed into \( n \) clusters). These encoder network outputs a batch size x number of clusters x embedding size vector after passing the input through the encoder network. Please refer to [Li et al. (2018)](https://www.li.com) for multi-head attention model. Encoder model (sequential operations) is shown in Equation 1-Equation 4. Note that Equation 2 and Equation 3 are part of the single step of multi-head attention model. For multi-layer attention model (we use 3 layers in our model), We repeat Equation 2 and Equation 3 for \( N \) times (\( N = 3 \) in our model).

\[
h_{\text{encoder}} = \text{FeedForward}(\text{input})
\]  

(1)
\[ h_{\text{encoder}} = \text{MultiheadAttention}(h_{\text{encoder}}) \] (2)

\[ h_{\text{encoder}} = \text{FeedForward}(h_{\text{encoder}}) \] (3)

\[ \text{output}_{\text{encoder}} = \text{FeedForward}(h_{\text{encoder}}) \] (4)

### 3.3.2. Decoder Network

Once we get the output from encoder network, we use context information and masking to select the next cluster to visit. Similar to here, we use 3 previously visited clusters to form the context. The use of context is shown in the right hand side of the Figure. These three clusters include - the station (C1), last to last visited cluster (C2) and last visited cluster (C3). We use embedding obtained from encoder layer and replace it with C1, C2 and C3. We also use masking to mask the cluster that has already been visited.

When we start ordering the clusters, we provide a matrix (masks) that ensures that the cluster with depot is masked. This matrix is updated as more clusters are scheduled. We use softmax function (Equation 5 - Equation 6) to select the next cluster. We use 10000* masks in Equation 5 to ensure that clusters that have already been visited do not get selected again. We learn the weights for \( v, W_{c1}, W_{c2}, W_{c3} \) shown in Equation 5 during the training phase of reinforcement learning. Once a cluster is selected, we update C2 and C3 by replacing it with appropriate embedding from the encoder network. Note that all the equations are in a matrix format and returns output for the batch size.

\[
\text{score} = \left( v \ast \tanh(\text{output}_{\text{encoder}} + W_{c1} \ast C1 + W_{c2} \ast C2 + W_{c3} \ast C3) \right) \ast -10000 \ast \text{masks} \] (5)

\[
\text{nextCluster} = \max(\text{softmax}) \] (6)

### 3.3.3. Critic Layer

We use a actor- critic method to train the weights of the reinforcement learning model. Actor-critic method reduces gradient variance and increases the speed of learning. We use a feed forward neural network shown in Equation 7 - Equation 8. Note that \( \text{FeedForward} \) is fully connected neural network. We use the number of stops in a route as shown in Equation 8 so that the actor layer learns the output faster. We learn of weights of \( W_{\text{critic}} \) and \( b_{\text{critic}} \) during training.

\[ h_{\text{critic}} = \text{FeedForward}(\text{input}) \] (7)

\[ \text{output}_{\text{critic}} = \text{NumberOfStops} + W_{\text{critic}} \ast h_{\text{critic}} + b_{\text{critic}} \] (8)
3.3.4. Optimization

During the training period, we use two loss functions to train the weights of the actor (encoder and decoder) and critic layer. The first loss function in Equation 9 aims at reducing the total cost of a route. Total travel time is measured in minutes and time window constrain violation is measured in minutes. Deviation from stop order (mean stop order in a cluster) is shown in Equation 10. The second loss in Equation 11 function aims at reducing the squared distance between the cost function from actor layer and output from critic layer.

\[
\text{cost}_{\text{actor}} = \text{TotalTravelTime} + \text{TimeWindowConstraintViolation} + \text{DeviationCost} 
\]

\[
\text{DeviationCost} = \sum_{\text{allCluster}} \frac{(\text{StopOrder}_{\text{actor}} - \text{StopOrder}_{\text{actual}})^2}{\text{NumberOfStops}} 
\]

\[
\text{TrainLoss} = \text{MeanSquareError} (\text{cost}_{\text{actor}} - \text{output}_{\text{critic}}) 
\]

3.4. Routing Within Cluster

Now that we know the order (from the reinforcement learning model) in which each cluster is to be visited, we use LKH3 algorithm to route within clusters. LKH3 is fast and provides good routing solution. While routing within a cluster, we aim just to minimize the total travel time within that cluster. To avoid circular loops in the routing within a cluster, we always include one stop from previous cluster and one stop from next cluster.

3.5. 2OPT

We use 2OPT heuristics as an extra step to improve the performance of a given route. We run 2OPT for 1000 iterations for each route. While running the 2OPT heuristics in the model apply phase, we use travel time (in minutes) and constraint violation (in minutes) as the objective function.

3.6. Pseudo Code

The pseudo code for the methodology is also shown below.
Pseudo Algorithm

1. Predict normalized stop order (0 - 1) for each stop using K-nearest neighbours.
2. Predict the stop order using two values of $K = \{10, 30\}$.
3. Group delivery stops in a route into $n = 30$ clusters using latitude and longitude of the delivery stops.
4. Use actor-critic method (with objective function as shown in Equation ??) to learn the weights of attention based deep learning model.
5. Use the learnt weights in step 4 to provide an order in which each cluster is to be visited
6. Use LKH3 algorithm to route within each cluster. While routing within a cluster, use one point of the next cluster to avoid circular routes within a cluster.
7. Use 2OPT algorithm to improve the routing. We used 1000 2OPT iterations per route to select an improved route.

4. Results and Conclusions

We could not run the model due to some issues with re-cli python version and tensorflow dependencies. However, we enlist some observations from our model which could help in improving the performance of the model:

- Using a multi-head attention in decoder layer improved the performance of the model (based on a unit test method where we overfit just one route). However, it takes longer to train this model and the model would not have done well within 10 hours training period.

- Instead of using 2 KNNs (with $K = 10$ and $30$), prediction from XGBoost can be used to predict the normalized stop order. We did not observe much improvement when using XGBoost due to only 6000 data points. However, with more data points, Amazon can improve the XGBoost predictions to predict normalized stop order.

- Dynamic algorithms such as Held-Karp (HK) performed better than LKH3 algorithm. We did not use HK as it may take more than 1 second to route within 1 cluster if the number of stops in a cluster exceed 20. Without 4 hour time constraint, HK algorithm can be used by Amazon for improved solutions.

- We used XGBoost model to predict the quality of the routes. We used SHAP values to find the main factors that decide if a route in high quality or not. Based on the SHAP values, we select time window violation and total travel time as major factors in estimating the quality of the route. hence, we used these two factors in building the optimization function.

- We did not have the computational power to run hyper-parameter optimization on different neural network arguments such as number of layers. We submitted the minimal model that performed decently for our objective function.
• We also tried a non-clustering method (an attention model with dimension 300 as the upper limit of number of stops). In this methodology, we use padding and masking for the reinforcement learning algorithm. This method suffered from lack of training time. Since the dimension in this method is higher (300 for maximum number of stops as against 30 for number of clusters), it needs bigger model and longer training periods in TPU. Amazon could use their higher computational resources for this method. This method will avoid the need for clustering and LKH3 algorithm.

• As observed in Figure ?? all stops in a cluster may not see much variation in the stop orders. We could improve the performance of the model by again normalizing the stop order within each route.
Appendix A. Optional Appendix

All the parameters used in the model (number of layers, learning rate, dimensions of the encoder and decoder layers) are provided in the code files (just before the final training block).
References


Article X

Enhancement of Routing Solver by Learning from Good Routes

Long He, Yiduo Huang and Tian Wang

Source code is available upon request from the authors.
Article X

Enhancement of Routing Solver by Learning from Good Routes

Long He
NUS Business School, National University of Singapore

Yiduo Huang
Department of Civil and Environmental Engineering, University of California, Berkeley

Tian Wang
Information Systems Technology and Design Pillar, Singapore University of Technology and Design

Abstract

The last-mile delivery problem is usually modeled as an optimization problem with the objective of minimizing total distance, duration or cost. However, the route quality is determined by more complicated factors in the real world, and the distance matrix could be asymmetric and time-variant. This paper proposed a method that can learn from experienced delivery driver routes using deep Graph Convolutional Networks. The output of the neural networks, represented by adjacency matrix of edges in a preferred route, can be used to enhance the routing solvers so that the heuristic-based solver can generate routes similar to routes by experienced drivers or superior algorithms. Our experiments show that this method can generate high-quality route using 803 high-quality routes by drivers where each route has 100-200 stops—improves the performance of OR-tools solutions by 1.16% using Amazon’s scoring metric. Furthermore, our method outperforms route solvers on the route score calculated by Sequence Deviation, Edit Distance, and Normalized Edit Distance with Real Penalty.

1. Introduction

The last-mile delivery service is crucial for e-commerce platforms like Amazon. The routing of vehicles for delivery is considered in the literature as vehicle routing problem (VRP), where a driver need to figure out a route to visit all the stops in a given list so that this route is the shortest. There are many variants of VRP. For example, VRP with time window considers the case when some stops can only be visited in a given time window and the driver will be penalized if the delivery doesn’t satisfy the time window constraint.

However, in reality, drivers rarely follow the routes suggested by the VRP solutions as highlighted by the Amazon Last-Mile Routing Research Challenge. There are many possible reasons for such deviations.
First, unlike the theoretical formulation of VRP, the platforms may not have the perfect knowledge about the distance matrix since the travel time between two locations is stochastic and time-dependent. Their estimation of travel time using historical travel data can be biased. Also, the actual distance matrix is non-Euclidean and non-symmetric. Moreover, the VRP itself is NP-hard so we may not find the accurate solution even if we have the actual travelling time matrix. Second, drivers may have their own routing preferences which reflect the private knowledge that is not yet captured by the distance matrix used by the platforms or the theoretical models of VRP.

Hence, there is a potential to address these issues by learning from experienced delivery drivers, and extract their tacit knowledge about the complex operational environment to improve the routing solutions. Given the route choice of some of the best drivers, our goal is to learn from their experience and give route suggestions to all the drivers. In our approach, we explicitly incorporate drivers’ tactic knowledge by adjusting the platform’s travel (distance or time) matrix before it is used as input for the routing solver. We assume that the “actual” travel cost along the drivers’ route choice is smaller than the one based on the given biased estimation. Specifically, we learn the probability of choosing a particular edge (a.k.a arc or link between two locations) in the driver’s route and adjust the travel matrix based on those predicted edge probabilities. Then we solve the travelling salesman problem (TSP) with soft time windows using heuristics to provide route suggestions to new drivers.

2. Literature Review

There is a recent trend of using deep learning techniques to solve combinatorial optimization problems. To study the routing problem on a graph and get insight from the graph structure, graph network tricks are widely used in previous studies. For example, [1] has used Gated graph convolution networks in [2] to solve the TSP problem. In their experiments, the TSP of ~100 nodes can be solved efficiently after learning from historical route choice or integer programming solvers’ suggested solutions. [3] use a unique combination of reinforcement learning and graph embedding to solve the TSP based on their structure2vec [4] technique.

Instead of directly predicting the solutions, i.e., the edges to be traveled in the routing decision, we keep the combinatorial optimization structure by still calling a routing solver. The machine learning methods are invoked to adjust the travel distance/time matrix which is the input to the routing engine. That is, by learning drivers’ routes which ought to be more preferred than the routes from solvers based on the original travel matrix, we can include drivers’ tactic knowledge by modify the travel matrix based on the prediction of edges in the driver’s preferred routes.

3. Methodology

Our approach consists of (1) predicting the edges that are traveled in a preferred route; and (2) adjust the travel matrix based on the predictions. The first step is developed based on the idea in [1]. Consider a
fully connected graph $G = (N, A)$, $N := \{1, 2, \ldots, |N|\}$, $A := \{(i, j) : i \in N, j \in N\}$. At each node, we have a set of packages to be delivered $PKG(i)$. Given the graph as our input, we know the features of each node and each arc, including:

- Location features (latitude, longitude, expected delivery time) of each node $x_i \in \mathbb{R}^3$, $i \in N, X := \{x_i\}$.
- Historical travel distance on each arc $d_{ij} \in \mathbb{R}, (i, j) \in A, D := [d_{ij}]$.
- Neighbor information $\delta^k_{ij} \in \{0, 1, 2\}$, $(i, j) \in A$. $\delta^k_{ij} = 1$ if $j$ is one of $i$’s $k$ nearest neighbor, $\delta^k_{ij} = 2$ if $i = j$, $\delta^k_{ij} = 0$ otherwise. And $\Delta^k = [\delta^k_{ij}]$.
- Time windows $t_i := [t^0_i, t^1_i] \in \mathbb{R}^2$, where $t^0_i = \max_{g \in PKG(i)} \{t^0_{ig}\}$ is the maximum start time among all packages at that station, and $t^1_i = \min_{g \in PKG(i)} \{t^1_{ig}\}$ is the minimum end time among all packages at that station. We assume $t^0_i < t^1_i$, otherwise, we will denote split the set of packages and one stop becomes two different stops, so that there’s a feasible time window for each stop. Let $T := [t_i]$.

We will train a graph convolution network (GCN) to output an matrix $P^{TSP} := [p^{TSP}_{ij}]$. Each $p^{TSP}_{ij} \in [0, 1]$ represents the probability that $(i, j) \in A$ is chosen in an experienced driver’s TSP route as follows:

$$p^{TSP}_{ij} = GCN(X, D, \Delta^k)$$ (1)

In the second step, using the $P^{TSP} := [p^{TSP}_{ij}]$ matrix, we can adjust the distance matrix accordingly

$$d^*_{ij} = d_{ij}(1 - w_0p^{TSP}_{ij})$$ (2)

Finally, after adjusting the distance matrix. With the given time window constraints, we can use an mixed integer programming solver as our blackbox solver (BBS) as follows:

$$\min_{y_{ij} \in \{0, 1\}} \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} d^*_{ij}y_{ij}$$

s.t.

$$\sum_{i=1, i \neq j}^{N} y_{ij} = 1, \forall j \in [N]$$

$$\sum_{j=1, j \neq i}^{N} y_{ij} = 1, \forall i \in [N]$$

$$\sum_{i \in S} \sum_{j \neq i, j \in S} y_{ij} \leq |S| - 1, \forall S \subset [N], |S| \geq 2.$$

Let the output route solution (the permutation or sequence) based on $y^*_{ij}$ be $\pi^*$

$$\pi^* = BBS(D^*, T)$$ (3)

where $D^* = [d^*_{ij}]$ is the adjusted distance matrix.

3.1. Graph Convolutional Networks

We follow the similar spirit in [1] for estimating the routing probability using GCN. The details are as follows.
Input layer. For each node, we will embed the information as \( d \)-dimension vectors \( \alpha_i^0 \in \mathbb{R}^d \):

\[
\alpha_i^0 = A_1 x_i + b_1,
\]

where \( A_1 \in \mathbb{R}^{d,3} \) and \( b_1 \in \mathbb{R}^d \) are parameters to be learned.

For each arc, we will also embed the information as \( d \)-dimension vectors \( \beta_{ij}^0 \in \mathbb{R}^d \):

\[
\beta_{ij}^0 = A_2 d_{ij} + b_2 || A_3 \delta_{ij}^k
\]

where \( A_2, A_3 \in \mathbb{R}^{d/2} \) and \( b_2 \in \mathbb{R}^{d/2} \) are parameters to be learned. We use || as the concatenation operator.

Convolution layers. We use \( l_0 \) convolution layers to process the \( d \)-dimension embedded edge and node information. For each layer, we have

\[
\alpha_{i}^{l+1} = \alpha_{i}^{l} + ReLU(BN(W_1^l \alpha_{i}^{l} + \sum_{j \in N} \eta_{i}^{l} W_2^l \alpha_{j}^{l}))
\]

\[
\beta_{ij}^{l+1} = \beta_{ij}^{l} + ReLU(BN(W_3^l \beta_{ij}^{l} + W_4^l \alpha_{i}^{l} + W_5^l \alpha_{i}^{l}))
\]

with

\[
\eta_{i}^{l} = \frac{\sigma(\beta_{i}^{l})}{\epsilon + \sum_{j \in N} \sigma(\beta_{j}^{l})}
\]

where \( W \in \mathbb{R}^{d,d} \), \( \sigma \) is the sigmoid function, \( ReLU \) is the rectified linear unit, and \( BN \) is the batch normalization. This structure is the same as \([2]\), where \( \eta_{i}^{l} \) is called a dense attention map.

Multi-layer perception (MLP). The edge embedding given by the last convolution layer \( \beta_{ij}^{l_0} \) will be the input of our multi-layer perception module, and the output is an probability \( p_{ij}^{TSP} \in [0,1] \) representing the potential that this edge is chosen in a preferred route, i.e., the adjacency matrix of a preferred route is

\[
p_{ij}^{TSP} = MLP(\beta_{ij}^{l_0})
\]

We denote the number of layers in the MLP by \( l_1 \).

Loss function. For each instance, we know the drivers’ preferred route: \( Y = [y_{ij}] \), \( y_{ij} = 1 \) if \((i,j) \in N\) is chosen, otherwise \( y_{ij} = 0 \). The loss function is just the binary cross-entropy loss averaged over mini-batches:

\[
LOSS(Y, p^{TSP}) = - \sum_{(i,j) \in N} (y_{ij} \log(p_{ij}^{TSP}) + (1 - y_{ij}) \log(1 - p_{ij}^{TSP})).
\]

3.2. Routing solvers

To obtain a feasible route, we use OR-tools by google as our blackbox solver \([5]\). We adjust the distance matrix using equation \([2]\) while we keep the time window information. We use the default parameters of OR-Tools solver for both our model and the comparison experiments. (i.e. first_solution_strategy = PATH_CHEAPEST_ARC, local search options = AUTOMATIC)
4. Results

4.1. Hyperparameter Configurations

We use GCN to predict the adjacency matrix of a good route. Our GCN consists of $l_0 = 3$ convolutional layers and $l_1 = 1$ layer in the MLP module with hidden dimension $d = 8$ for each layer. The value of best performing $w_0$ in equation (2) was found to be $0.205$. The learning rate and optimizer follow the same setting as in [1].

4.2. Experiment results

We randomly sample 827 routes with route score = “High” to train our graph convolutional network (GCN). Our model is then tested on using 209 routes with route score = “High”, and then compared with performance of Google OR-Tools. Using Amazon’s scoring metric, the lower the score, the closer the route is to a preferred one.

<table>
<thead>
<tr>
<th>Method</th>
<th>Evaluation Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>0.096027</td>
</tr>
<tr>
<td>OR-Tools</td>
<td>0.09715</td>
</tr>
</tbody>
</table>

Table 1: Performance of our method and Google OR-Tools on 209 with route score = “High”. Evaluation score is computed based on Amazon’s scoring metric.

5. Conclusions

In this research, we propose to use the predicted adjacency matrix of preferred route tour to adjust the travel distance matrix which is used in a route solver. This approach allows the route solver to use tactic knowledge learned from previous preferred routes conducted by high-performing drivers or better algorithms. Our experiments show that our approach improves the performance of OR-tools solutions by 88.10% using Amazon’s scoring metric.

References


URL https://developers.google.com/optimization/
Article XI

Using Graph Neural Networks to Solve Route Sequencing Problems

Andres Regal

Source code is available on GitHub:
https://github.com/a-regal/amzn-lastmile-challenge
Article XI

Using Graph Neural Networks to Solve Route Sequencing Problems

Andres Regal

Engineering Department, Universidad del Pacifico
Av. Salaverry 2020, Jesus Maria, Lima, Peru

Abstract

Graph Neural Networks (GNNs) have had recent success in solving combinatorial optimization problems. Since, by design, they are made to extract features from graphs, handling the networks generated from vehicle routing problems can have solution quality advantages. In this paper, we present a GNN to solve route sequencing problems by adjusting existing architectures from the literature. By preprocessing the original networks into a directed bipartite graph, a tree-like structure was chosen to represent the sequential nature of designing a route. Using this structure, our model presents an average score of 0.92, while performing at a 0.88 score when testing using high quality routes only.

1. Introduction

Vehicle routing has been a core problem within the operations research literature since its introduction by George Dantzig and John Ramser in 1959. Being a combinatorial optimization problem, and given the NP-Hard nature of finding an optimal solution, several algorithms have been developed to approach the optimal solution in large instances. It is within these large instances that machine learning has potential to accelerate solution times and improve solution quality. Specifically, deep learning neural network architectures have been used in recent efforts to leverage the learning capabilities of recurrent, convolutional and graph networks and the computational power available with modern GPUs.

Within this competition, our goal was to estimate a model that can imitate a driver’s domain knowledge when performing a route while retaining the cost minimization objective of a Vehicle Routing Problem (VRP). To do this, we focused on transforming the data into a graph structure that can be used as input for a GNN. This graph is composed of directed, weighted edges, an edge index matrix and a node feature matrix. The weights are represented by the travel time, the node features contain the total volume to be delivered, total service time and the start and end times of the most narrow time window within a node,
and the edge index matrix represent the source and target nodes for each edge (see Section 3 for more detail).

For each route, several bipartite graphs, where the number of nodes decrease as nodes are picked by the network, are added to the training set. This enables us to learn only from high efficiency routes. By transforming the 2718 high efficiency routes we obtain a dataset of ~500,000 graphs to learn from. The choice of using a GNN relies on the data structure we implement. To capture the spatial and temporal decision making that a driver must perform, a graph allows the network to see both the spatial characteristics (distance) and temporal characteristics (number of candidate nodes, decreasing per timestep) that lead to a stop being chosen as the next-in-route. This approach can be thought of as estimating the value of each stop at a given route step, where only those stops that have not been visited are considered for evaluation.

2. Literature Review

Recent works on using GNNs for combinatorial optimization have had success at finding the optimal solution of known instances and overcoming current solvers either in computational time or solution quality for heuristic approaches. The most relevant papers to our model are those of Gasse et al. (2019), Li et al. (2018), and Kool et al. (2018).

The first of these, and the most relevant to our methodological approach is Learn2Branch from Gasse et al. (2019). In their paper, the core concept is to develop a GNN that generates a branching policy for branch-and-bound combinatorial optimization solvers. They represent a Mixed Integer Linear Program (MILP) as a bipartite graph by constructing nodes that represent either optimization variables (either binary or continuous) and constraints. The edges of this graph represent which nodes are associated to which constraint. Using imitation learning, the network learns from an "expert" (the solver’s branching policy) and minimizes the cross entropy between the proposed branching policy from the network and the branching policy from the solver. This helps accelerate solution times, since the network’s inference in milliseconds guides the branching policy of the SCIP solver, and solution quality in Set Covering, Facility Location and Combinatorial Auction instances.

A second paper important paper is Li et al. (2018). In their paper, the authors proposed a method based on graph reduction and tree search. Their algorithm takes an initial graph, represents it as a reduced, equivalent graph and processes it with a GNN. The GNN is used to generate probability maps regarding whether or not a node belongs to the optimal solution. These maps are used iteratively to label the nodes in the search tree, where the complete labellings are refined by rapid local search, choosing the best result. The authors tested this approach on Maximal Independent Set, Minimum Vertex Cover and Maximal Clique instances, where their proposed method either tied the SOTA with less than 50% solution time or it solved...
A third approach worth mention is that of Kool et al. (2018), where a transformer-based attention recurrent neural network (RNN) was designed to solve Travelling Salesman Problems (TSP). To adapt the RNN architecture to the TSP graph structure, a specific masking, decoding and input context were designed to handle the recurrent graph features. To optimize the network, the authors implemented the reinforcement learning algorithm REINFORCE with greedy rollout baselines algorithm, which showed computational benefits within the results. Compared to Pointer Networks (Vinyals et al., 2015), for instances of less than 100 nodes the attention model takes at most 6 seconds to solve the instances with a 4.53% optimality gap. These computational benefits contrast with exact solutions provided by solvers like Gurobi, where solution times go up to 17 minutes but with no optimality gap.

For further research into the use of GNN to solve combinatorial optimization problems, we refer the reader to Vinyals et al. (2015); Bengio et al. (2020); Bello et al. (2016); Vesselinova et al. (2020) and Dai et al. (2017).

3. Methodology

As mentioned in Section 1, our approach is centered around graph neural networks. In order to use them, the data provided within the challenge must be transformed into a graph structure. To do this, 4 steps are required: (i) construct the distance matrix, (ii) get the start node of a route, (iii) get the actual sequence for labeling, and (iv) get the total volume, service time and time windows. The first of these is simple enough. Within the routes file, for each stop the travel time to all other stops is provided. Since the stops are alphabetically ordered in all other files, assigning a 0-n index, where n is the number of stops, for each stops allow a simple n x n travel time matrix to be constructed.

The second task is to get the start node for each route, such that the sequence of directed graphs that represent a route can be generated appropriately. This involves validating which stops within the route file are not of type "Drop-off" and getting its index (i.e. stop 13). To get the actual route sequence, extracting the ”actual” attribute from each route provides each stop code and its index in the route. Integrating this into the graph requires assigning each stop code (i.e. ‘VE’) a unique integer id (i.e. 103) and sorting by the index in the route. With this, for each graph, its label is assigned as the next element in the sequence. This is illustrated in Figure 1, where a start node N₀ is connected to three other nodes. Within the sequence, the first node to visit is N₂, so it is labeled as the target (dotted line). This process is repeated for all graphs within the route.
Finally for each stop, a set of node features need to be constructed. As mentioned within the (iv) task, for each node, the total service time, total volume to be delivered and the start and end times of the shortest time windows must be processed into a node feature matrix. Both the total service time and total volume are easily computed. For each package within a stop, the expected service time in seconds in accumulated into the expected total service time for that stop. Similarly, for each package the dimensions feature is used to compute the package’s volume and accumulate it as the total volume. The time windows require some additional logic. The initial time window start and end times are set to 0 and 86400, respectively (to represent a full day in seconds). Next, for each package that has a time window constraint, the time windows are updated such that the most strict time window can be computed (i.e. the tightest window between the start and end times).

This preprocessing approach looks to provide the network with critical information within the driver’s decision making process: how far are the next stops?, how long is a delivery stop going to take?, does the stop involve delivering large volumes? and how difficult is it to satisfy the time windows established by the stop? Since the labels are which stop is to be selected, this can be modelled as a classification problem. For each graph, the network’s output looks to select the target node such that the cross-entropy loss (see Eq. 1) is minimized.

\[
    \text{loss}(x, \text{class}) = -x[\text{class}] + \log(\sum_j \exp(x[j]))
\]  

(1)

Architecture-wise, given the computational constraints of the challenge, the GNN implemented for this paper consists of 1 graph convolution layer and a mish activation function. To implement this network, PyTorch (Paszke et al., 2019) and PyTorch Geometric (Fey & Lenssen, 2019) were used as the main frameworks using Python. A graph convolution layer, which extends the convolution operation to nodes and edges within a graph, can be defined (using the notation from Kipf & Welling (2016)) as follows:
\[ H^{l+1} = \sigma(\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} H^l W^l) \]  

Where \( \hat{A} = A + I_N \) is the adjacency matrix of the undirected input graph with added self-connections. \( I_N \) is the identity matrix, \( \hat{D}_{ii} = \sum_j \hat{A}_{ij} \). \( W^l \) represents the trainable weights for layer \( l \), \( H^l \) is the matrix of activations in the \( l^{th} \) layer, and \( \sigma(\cdot) \) represents the activation function (such as ReLU, Mish, softmax). With this feature extraction, two things are missing from this network: an activation function and regularization.

Since graph neural networks are computationally expensive (especially on large graphs), adding depth and regularization components is a tough trade-off in training time. As such, instead of using a ReLU-Dropout combination, we opted to use the Mish activation function (see Eq. 3). Mish was presented in Misra (2019) as an alternative to the Swish activation function. The core concept behind Mish was to provide a non-monotonic, self-regularized activation function that is able to preserve small negative weights and a smooth profile (see Misra (2019) for a detailed analysis of the function and its development via Neural Architecture Search).

\[
Mish(x) = x \ast \tanh(\text{softplus}(x)) = x \ast \tanh(\ln(1 + e^x))
\]  

Through different benchmarks, such as CIFAR-10, ImageNet-1k and MS-COCO, Mish improved the performance of the best architectures presented (such as ResNet-50 and PeleeNet) in image classification tasks. As such, since we model route sequencing as a classification task it is natural to include Mish given its properties. Regarding training, we used an Adam optimizer (Kingma & Ba, 2014) with learning rate of 0.003, \( \beta_1 = 0.9 \) and \( \beta_2 = 0.999 \) for 100 epochs. This amount of epochs accounts for \( \sim 11:30 \) hours of CPU processing time. Accounting for data reading, dataset generation and training time, the total processing time is \( \sim 11:45 \).

Finally, for inference, a sequencing algorithm was developed to provide an adaptive graph generation for unseen routes. Algorithm 1 presents the pseudocode for the sequencing algorithm. This algorithm takes as parameters: a distance matrix, time window matrix, total volumes and stop times, a start node and a model. First, the algorithm initializes two sets: available and visited. These sets keep track of which nodes are available and which nodes have been visited. Next, indices is defined as the difference between this two sets, which is used to dynamically adjust which features and edges are to be converted into the input graphs.

With the initial indices defined, the first bipartite graph can be constructed using the start node as the source and all other nodes (since indices will just be the whole set of nodes minus the start). Using Pytorch Geometric’s convention, the node features are stored into an "x" variable, the weights of each edge correspond to the "edge_attr" variable and the definition of the adjacency matrix is defined by the "edge_index" variable. These three variables are instanced with the crate_graph_components function, which indexes the distance matrix to the current stop’s row and the column to the indices, and creates the node features.
Algorithm 1 Sequencing algorithm for inference

**Parameters**: distance matrix \((mat)\), time window matrix \((tw)\), total volume array \((vl)\), total service time array \((st)\), initial start node \((\text{stop})\), model

Set available = \{ 0 \ldots N \}

Set visited = \{ \text{stop} \}

Set sequence = {}  

Set indices = available - visited

Set node\_features = concat\((tw[:,0], tw[:,1], vl, st)\)

Set \(x, edge\_attrs, edge\_index\) = create\_graph\_components\((mat, \text{stop}, indices)\)

for step in 0:length\((vl)\) do
    if length\((\text{visited})\) == length\((vl)\)-1 then
        sequence.append\((\text{available}-\text{visited})\)
        break
    else
        graph = create\_graph\((x, edge\_index, edge\_attrs)\)
        out = model\((graph)\)
        selected\_node = arg\_max\((\text{softmax}\((\text{out})\))\)
        selected\_stop = indices\[selected\_node\]
        sequence.append\((selected\_stop)\)
        visited.add\((selected\_stop)\)
        stop = selected\_stop
        indices = available - visited
        Set \(x, edge\_attrs, edge\_index\) = create\_graph\_components\((mat, \text{stop}, indices)\)
    end if
end for

return sequence

and edges according to the indices. Next, a for loop is initialized with a set number of iterations at The create\_graph function mentioned in the algorithm refers to converting the graph into a structure for the framework of choice (Pytorch Geomtric uses a Data object as seen in the implementation). If all nodes but one are visited, the algorithm appends this last node to the sequence and breaks (avoiding exceeding the length of the list and indexing errors).

The core of the algorithm lies in the else block. Once the graph is within the structure of the framework (graph), it passed through the network to get the output probabilities (out). Using the softmax function, the selected node is found by applying softmax to the outputs and using the argmax operator (the index
of the highest value within the output). This node is added to the sequence and visited set, the indices are updated, and then the following bipartite graph is built where the nodes within the graph exclude the previous source node. Once all iterations are complete, the final sequence is returned. Within the implementation, a generalization of this algorithm to also match the output JSON format for the challenge is added.

4. Results and Conclusions

Following our methodology using 200 epochs, the model performs at a 0.92 average score. Overall, the PDF for the 6112 route scores can be visualized in Figure 2. The distribution has 3 peaks of similar density, the first around 0.6, the second close to 0.9 and the third around 1.2. The main statistics for all the training routes are presented in Table 1.

![Figure 2: Overall distribution of route scores](image)

Across all 6112 training routes, the best performing routes score at 0.17, while the worst performing routes score at $\geq 1.12$. Those routes that performed especially well were mostly compact (i.e. the stops were not too far from each other), with lax time windows and with few stops. This contrasts with the worst performing routes, where time windows are stricter and stops are further apart, increasing the complexity of the decision. Overall, this score is not high, since the biggest share of the distribution is above 0.73. Even so, further analysis into the route scores by efficiency level shows slight improvements over the training set.

Since the training data for the GNN was focused on the high efficiency routes, it is natural to look at the scores over high efficiency routes for reference on model performance. The score does improve to 0.87, but this still is not good enough of a performance. The best routes in this level do show high levels of similarity to the actual routes (0.18) but the worst performing routes drag the mean score up. This is replicated across efficiency levels, and the scores are logically higher since the model is not trained to produce medium/low
Table 1: Scoring statistics by route level

<table>
<thead>
<tr>
<th>Route Level</th>
<th>Count</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>2718.0</td>
<td>0.878731</td>
<td>0.281484</td>
<td>0.186441</td>
<td>0.673201</td>
<td>0.878008</td>
<td>1.098277</td>
<td>1.764595</td>
</tr>
<tr>
<td>Medium</td>
<td>3292.0</td>
<td>0.961025</td>
<td>0.243195</td>
<td>0.177671</td>
<td>0.782066</td>
<td>0.969549</td>
<td>1.141967</td>
<td>1.648028</td>
</tr>
<tr>
<td>Low</td>
<td>102.0</td>
<td>0.970873</td>
<td>0.252465</td>
<td>0.261824</td>
<td>0.792323</td>
<td>0.985833</td>
<td>1.168790</td>
<td>1.554036</td>
</tr>
<tr>
<td>Total</td>
<td>6112.0</td>
<td>0.924593</td>
<td>0.264232</td>
<td>0.177671</td>
<td>0.731942</td>
<td>0.928670</td>
<td>1.124915</td>
<td>1.764595</td>
</tr>
</tbody>
</table>

efficiency routes, which would cause strong dissimilarities to these kinds of sequences, hence the high scores.

Overall, our model did not perform as expected. Given the computational constraints, our model lacked the following: a deeper architecture, stricter regularization, higher levels of feature extraction, a learning rate scheduler and training epochs. Regarding the first of these, the depth of architecture, it is clear from the literature review that most state of the art models require several graph-oriented layers. The number of layers used in the literature is perfectly fine when using a GPU, but graph neural networks are known to be slow on CPU only.

This is also true for regularization. Since we are using a single layer, dropout is not an option since this layer is the one used for prediction. Even so, when using two layers, training the "subnetworks" generated by dropout hinders the CPU processing time. When choosing the network’s architecture, we had to balance several aspects of its computational complexity. In our implementation, a single layer that extracts a 1D feature vector was used to fit the computational budget. This limits the amount of features that the GNN can extract and learn from, as well as the kinds of layers and operations over the resulting graphs that may be of use for prediction. Modern deep learning approaches rely not only on regularizers and activation functions, but on optimizer performance boosters such as learning rate scheduling to avoid local minima.

All in all, even if our experimental design followed the state of the art of using graph neural networks (and specifically, graph convolutions), the computational constraints limited the success of our model. To truly understand how good the model, the Mish activation, and the data structure truly are for this problem, further research would need to focus on (i) expanding the training environment to GPUs, (ii) add greater feature extraction (such as simple GCNs, Spline Convolutions or Graph Self-Attention) and regularization, and (iii) increase training time with learning rate schedulers.
References


Article XII

A Machine Learning Framework for Last-Mile Delivery Optimization

Alexandre M. Florio, Paulo da Costa and Sami Serkan Özarık

Source code is available on GitHub:
https://github.com/aflorio81/Amazon-MIT-RoutingChallenge
A Machine Learning Framework for Last-Mile Delivery Optimization

Alexandre M. Florio, Paulo da Costa, Sami Serkan Özarık

Department of Industrial Engineering and Innovation Sciences
Eindhoven University of Technology, P.O. Box 513, 5600MB Eindhoven, The Netherlands

Abstract

In the context of the Amazon/MIT routing challenge, this paper presents a machine learning framework for optimizing last-mile delivery routes. Contrary to most vehicle routing problem settings where an objective function is clearly defined, in the real-world setting considered in the challenge an objective is not explicitly specified and must be inferred from data. Leveraging techniques from machine learning and classical traveling salesman problem heuristics, we propose a “pool and select” algorithm to prescribe high-quality last-mile delivery sequences. In the pooling phase, we exploit structural knowledge acquired from data, such as common entry and exit regions observed in training routes. In the selection phase, we predict the scores of candidate sequences with a high-dimensional, pre-trained and regularized regression model. The score prediction model, which includes a large number of predictor variables such as sequence duration, compliance with time windows, earliness, lateness, and structural similarity to training data, displays a good prediction accuracy with an out-of-sample mean squared error of 0.00121. Overall, the framework is able to prescribe competitive delivery routes, as measured by an average out-of-sample score of 0.0474 across several development datasets. Moreover, the framework requires only a fraction of the maximum computational budget allowed in the challenge: less than 40 minutes are needed for training the score prediction model, and on average less than two seconds are required to determine a visiting sequence given a test route.

1. Introduction

The design of efficient last-mile delivery routes must take into account several factors such as customer availability and time windows, spatiotemporal congestion patterns, parking availability, turn restrictions, among many others. For tractability, most vehicle routing models and corresponding solution methods typically assume a single optimization direction (e.g., route duration) with eventual side constraints (e.g., time
windows). Not surprisingly, experience often shows a significant deviation between routing plans prescribed by decision support tools and actual routes executed by drivers.

Given the difficulty of translating all factors behind efficient delivery routes into a tractable optimization model, the Amazon/MIT routing challenge called for a data-driven method for designing high quality last-mile routes. In this perspective, relevant characteristics of efficient routes must be inferred from a training dataset with a large number of actual delivery sequences, which are pre-classified by domain experts (e.g., drivers) with respect to their quality. This paper presents a prescriptive analytics framework to acquire and learn essential information about such training data and prescribe high-quality delivery sequences. In particular, we propose a “pool and select” algorithm, which, in the pooling phase, exploits structural knowledge acquired from data, such as common entry and exit regions observed in the tours, and generates candidate sequences with classical traveling salesman problem (TSP) heuristics. In the selection phase, the algorithm predicts the scores of candidate sequences with a regression model and prescribes the sequence with the best expected performance.

The remainder of this paper is organized as follows: in Section 2 we review the relevant literature on last-mile delivery and machine learning (ML) methods for combinatorial optimization; in Section 3 we describe our learning framework in detail; finally, in Section 4 we present and discuss several results on validation datasets and give further improvement directions.

2. Literature Review

Last-mile delivery refers to the logistics of freight transportation in the final leg of the journey to the customer. Due to the large increase in e-commerce orders, logistics service providers are faced with challenges. The problems of last mile delivery are typically caused by the fact that deliveries are composed of individual orders that must be delivered to several different addresses. In spite of increasingly growing research on optimization methods applied to city logistics, there remains a gap in the literature concerning real-world applicability of those methods to improve efficiency in the last-mile \cite{Savelsbergh2016}. The canonical vehicle routing problem (VRP; see, e.g., \cite{Toth2014}) has a known objective and seeks for cost-minimal routes. In recent decades, however, the number of VRP variants has grown quite substantially, as a result of the diverse set of applications, especially in last-mile delivery \cite{Cattaruzza2017}. Large proportion of VRP studies incorporate hard time windows (see, e.g., earlier work by \cite{Savelsbergh1985, Gendreau1998}). Nonetheless, as also can be observed from the routing challenge data, only a relatively small percentage of customers may have time windows in real-world cases. As a result, innovative solutions based on customer-related data available from various sources are proposed in order to achieve high success rates and low costs in last-mile delivery. \cite{VanDuin2016} show that the delivery efficiency is closely related to (demographic) characteristics of an area. \cite{Florio2018} introduce customer availability profiles (CAPs) to model customer presence throughout the delivery period and improve hit rates. Özarık

\textit{Florio, da Costa and Özarkin}
et al. (2021) and Voigt et al. (2021) make use of CAPs to solve a routing and scheduling problem for last-mile deliveries with uncertain customer presence.

Contrary to most VRP settings, when real-life cases are considered, as in the challenge, an optimization direction is not explicitly specified and, therefore, must be inferred from data. Li & Phillips (2018) identify the variables that can predict whether a route will deviate from the planned sequence and predict the degree of deviation on distance. They show that routes with a higher number of stops are more likely to deviate. To the best of our knowledge, however, no previous study focuses on data-driven methods to learn drivers’ tacit knowledge and, based on this knowledge, to improve the planning of last-mile delivery routes.

To take advantage of real-life data, reduce the reliance on expert heuristics, and exploit similarities between instance characteristics, ML methods have been proposed to solve combinatorial optimization problems (Bengio et al., 2021). Several works have focused on routing problems such as the VRP or its simplest form the TSP. With the current advances in deep learning, several methods have achieved impressive results in learning construction and improvement heuristics. In particular, Vinyals et al. (2015) revisited the problem and proposed Pointer Networks (PtrNet) to learn optimal TSP sequences generated with Concorde (Applegate et al., 2011). Following this work, Bello et al. (2017) extended the PtrNet to learn tour construction policies without requiring optimal solutions via reinforcement learning. Later, several works extended the deep learning and deep reinforcement learning approaches (Deudon et al., 2018; Kool et al., 2019, 2021; Joshi et al., 2019), resulting in lower optimality gaps. Recently, other methods considered the task of learning improvement policies for the TSP, i.e., learning to generate new solutions starting from a poor one (Wu et al., 2021; da Costa et al., 2020). These deep learning methods have achieved promising results for small instances, surpassing the solution quality of specialized TSP heuristics.

While these methods achieve impressive results in terms of solution quality, they are limited to small instances and require long training times and specialized hardware to be able to learn effective policies that are competitive with simple heuristics such as the Lin-Kernighan-Heulsgaun (LKH3) (Helsgaun, 2017). Thus, we focus our attention on methods that do not require many hours of batched computation and specialized hardware. For this reason, to select good TSP tours, our proposed method combines expert-designed search heuristics from operations research with a classical regression model. In the proposed ML approach, the objective is to learn a relationship between sequence features and solution quality, given a fixed dataset of previous sequences.

Our method is similar to search algorithms that use ML to make informed decisions about promising search regions. Previous work on combining search and ML considered algorithm selection methods, in which an ML model is used to choose the best algorithm out of a portfolio of options for a problem instance (Bischl et al., 2016). Other methods considered ML-guided large neighborhood search by learning the decisions about destroy and repair operators (Hottung & Tierney, 2020). Closely related methods considered learning metrics of the variable-value selections and estimates of the objective using an ML model trained offline over available solutions (Hottung et al., 2020). While our “pool and select” method generates good quality
sequences, we point out that one drawback is the reliance on an offline ML model that may fail to generalize to instances outside the training data distribution, a common issue in this class of methods (Lombardi & Milano, 2018).

3. Methodology

3.1. General Definitions

We denote by $\mathcal{H}^{TR}$ and $\mathcal{H}^{TE}$ the set of routes in the training and test data, respectively, and define $\mathcal{H} = \mathcal{H}^{TR} \cup \mathcal{H}^{TE}$. Given a route $h \in \mathcal{H}$, we denote by $S(h)$ and $d(h)$ the set of stops and the station code (e.g., "DLA9" or "DCH2") associated with $h$, respectively. The special stop $s(h) \in S(h)$ indicates the station. The zone of a stop $s \in S$, if available, is described by a string in the format "A-1.2B". We combine the station code $d(h)$ and the stop zones to define macro, micro and nano zones, as follows:

**Definition 1 (Macro, micro and nano zones).** Given a route $h$ and a dropoff stop $s \in S(h)$ with a zone identifier in the format "A-1.2B", the macro, micro and nano zones of $s$ are defined as $z_M(s) = d(h) \oplus "-A"$, $z_m(s) = d(h) \oplus "-A-1"$, and $z_n(s) = d(h) \oplus "-A-1.2B"$, respectively, where $\oplus$ denotes string concatenation.

A feasible sequence for a route $h$ is a permutation $\lambda = (u_1, \ldots, u_{|S(h)|})$ of $S(h)$ such that $u_1 = s(h)$. We denote by $\Lambda(h)$ the set of sequences that are feasible to a route $h \in \mathcal{H}$. Further, we let $\lambda_A(h)$ be the actual (or realized) sequence associated with a training route $h \in \mathcal{H}^{TR}$.

A travel time matrix $T(h) \in \mathbb{R}_{\geq 0}^{|S(h)| \times |S(h)|}$ is associated with the set of stops $S(h)$ of each route $h \in \mathcal{H}$. In the data made available in the challenge, multiple packages may be delivered at a stop, and time windows are associated with packages and not stops. As the average number of packages per stop is generally low, we aggregate package information at the stop level. Therefore, with each stop $s \in S(h)$ a time window $[e(s), \ell(s)]$ and a service time $\sigma(s)$ are associated. The stop time window corresponds to the intersection of the stop packages’ time windows and the stop service time is the sum of the stop packages’ service times. If there is no package with time window at stop $s$, we define $[e(s), \ell(s)] = [-\infty, +\infty]$.

In practice, especially in an urban setting, travel times are highly variable due to congestion and other network conditions. To assess the compliance of sequences with time windows under uncertain travel and service times, given a route $h$ we define two travel time scenarios represented by the matrices $t^B(h)$ and $t^W(h)$, where $t^B, t^W \in \mathbb{R}_{\geq 0}$ are best- and worst-case parameters concerning the realization of travel times, respectively. Similarly, we define best- and worst-case service time multipliers $\sigma^B, \sigma^W \in \mathbb{R}_{\geq 0}$. Then, the compliance of sequences with time windows is measured by the *earliness* and *lateness* metrics, defined below:

---

1 No case where such intersection is empty was encountered in the data.
Definition 2 (Stop and sequence earliness and lateness). Given a route \( h \in \mathcal{H} \) and a sequence \( \lambda \in \Lambda(h) \), we let \( \tau(s|T,\sigma) \) be the arrival time at stop \( s \in S(h) \) when following sequence \( \lambda \), considering a travel time matrix \( T \) and service times adjusted by multiplier \( \sigma \). The earliness and lateness of a stop \( s \) are defined as \( E(s) = (\tau(s) - \tau(s|\sigma^B T(h),\sigma^B))^+ \) and \( L(s) = (\tau(s|\sigma^W T(h),\sigma^W) - \ell(s))^+ \), respectively. The earliness and lateness of sequence \( \lambda \) are defined as \( E(\lambda) = \sum_{s \in S(h)} E(s) \) and \( L(\lambda) = \sum_{s \in S(h)} L(s) \), respectively.

Finally, we let \( D(\lambda) \) denote the duration of a sequence \( \lambda \in \Lambda(h) \) under travel times \( T(h) \), where the duration takes into account only the transit times, i.e., service times are ignored when computing \( D(\lambda) \).

3.2. Structural Information

Macro, micro and nano zones play an important role in our framework. In particular, we leverage the structural information conveyed by macro, micro and nano zone transitions observed in the training data to generate candidate sequences for test routes and to evaluate the quality of proposed sequences. This structural information is collected by processing training routes as described in Algorithm 1. First, procedure \textsc{CollectInfo} initializes macro, micro and nano zone transition counters (lines 1-3). Then, for each route in the set of training routes passed as input, the procedure increments the counters according to the transitions observed in the actual sequences (lines 7-13). Note that stops with unspecified zones are ignored when processing the training data (line 5).

\begin{algorithm}
\caption{\textsc{CollectInfo}}
\begin{algorithmic}[1]
\Require Set of training routes \( H \)
\Ensure Structural info \( I \)
\State \( \delta_M(i,j) \leftarrow 0 \quad \forall i,j \in \bigcup_{h \in H, s \in S(h)} \lambda_M(s) \) \hfill \Comment{Initialize macro zone transition counters}
\State \( \delta_m(i,j) \leftarrow 0 \quad \forall i,j \in \bigcup_{h \in H, s \in S(h)} \lambda_m(s) \) \hfill \Comment{Initialize micro zone transition counters}
\State \( \delta_n(i,j) \leftarrow 0 \quad \forall i,j \in \bigcup_{h \in H, s \in S(h)} \lambda_n(s) \) \hfill \Comment{Initialize nano zone transition counters}
\For {\( h \in H \)}
\State \( (s_1, \ldots, s_F) \leftarrow \text{\textsc{SubseqKnownZones}}(\lambda^\Lambda(h)) \) \hfill \Comment{Fetch from \( \lambda^\Lambda(h) \) the stops with known zones}
\State \( s_{F+1} \leftarrow s_1 \) \hfill \Comment{Consider transition from last stop to station}
\For {\( i : \{1, \ldots, F\} \)}
\If {\( \lambda_M(s_i) \neq \lambda_M(s_{i+1}) \)} \hfill \Comment{If there is a macro zone transition}
\State \( \delta_M(\lambda_M(s_i), \lambda_M(s_{i+1})) \leftarrow \delta_M(\lambda_M(s_i), \lambda_M(s_{i+1})) + 1 \)
\EndIf
\If {\( \lambda_m(s_i) \neq \lambda_m(s_{i+1}) \)} \hfill \Comment{If there is a micro zone transition}
\State \( \delta_m(\lambda_m(s_i), \lambda_m(s_{i+1})) \leftarrow \delta_m(\lambda_m(s_i), \lambda_m(s_{i+1})) + 1 \)
\EndIf
\If {\( \lambda_n(s_i) \neq \lambda_n(s_{i+1}) \)} \hfill \Comment{If there is a nano zone transition}
\State \( \delta_n(\lambda_n(s_i), \lambda_n(s_{i+1})) \leftarrow \delta_n(\lambda_n(s_i), \lambda_n(s_{i+1})) + 1 \)
\EndIf
\EndFor
\EndFor
\State \textbf{return} \( I = \{\delta_M, \delta_m, \delta_n\} \) \hfill \Comment{Return macro, micro and nano zone transition counters}
\end{algorithmic}
\end{algorithm}

Once structural information \( I = \{\delta_M, \delta_m, \delta_n\} \) about a set of training routes is available, we are able to compute the similarity of a sequence with respect to \( I \). Given a route \( h \) and a sequence \( \lambda = (u_1, \ldots, u_{|S(h)|}) \),
the similarity at the nano level is given by:

$$\Delta_n(\lambda|I) = \sum_{i=1}^{F} \delta_n(z_n(s_i), z_n(s_{i+1})),$$

where \((s_1, \ldots, s_F)\) is the subsequence of \(\lambda\) obtained by removing the stops with unknown zones, and \(s_{F+1} = s_1\). Likewise, the similarities of \(\lambda\) with respect to \(I\) at the micro and macro levels, denoted by \(\Delta_m(\lambda|I)\) and \(\Delta_M(\lambda|I)\), are computed by using \(\delta_m\) and \(\delta_M\) instead of \(\delta_n\), and \(z_m(\cdot)\) and \(z_M(\cdot)\) instead of \(z_n(\cdot)\) in Equation (1), respectively. Finally, the number of nano transitions along sequence \(\lambda\) is defined as:

$$T_n(\lambda) = \sum_{i=1}^{F} [z_n(s_i) \neq z_n(s_{i+1})],$$

and the number of micro and macro transitions along \(\lambda\), denoted by \(T_m(\lambda)\) and \(T_M(\lambda)\), are defined in a similar fashion by replacing \(z_n(\cdot)\) by \(z_m(\cdot)\) and \(z_M(\cdot)\) in Equation (2), respectively.

3.3. Solution Method Overview

Given a test route \(h \in H^{TE}\), our algorithm to prescribe a high-quality sequence consists of the following two steps:

(i) Sequence pooling: generate a subset \(\Lambda \subset \Lambda(h)\) of candidate sequences.

(ii) Sequence selection: return the sequence \(\lambda^* = \arg \min_{\lambda \in \Lambda} \hat{g}(f(\lambda, h, \Lambda))\) with the best predicted score, where \(\hat{g}: \mathcal{F} \mapsto \mathbb{R}\) is the prediction function that returns a numerical score given a feature vector \(x \in \mathcal{F}\), and \(f(\cdot)\) is the feature extraction function that returns a feature vector \(x \in \mathcal{F}\) given a proposed sequence, a test route, and statistics on \(\Lambda\).

The method employed for pooling candidate sequences is based on the random insertion (RI) TSP heuristic [Rosenkrantz et al., 1977]. For our purposes, RI is particularly advantageous since it generates a diverse pool of sequences, is highly efficient, and performs well in asymmetric instances [Glover et al., 2001]. The pooling procedure is described in Algorithm 2. First, we determine the set of candidate entry and exit stops (lines 1-6). Next, until we reach the desired pool size, we select randomly an entry and an exit stops from the sets of candidate stops (line 9). Procedure SELECTRANDOM bias the random selection by favoring stops from nano regions that appear more frequently as entry and exit regions in the training data. Then, we compute by RI a sequence where the first and last stops are fixed to the entry and exit stops selected, respectively, and add the new sequence to the pool (lines 10-11).

After analyzing the structure of sequences in the training data, we noticed that high quality sequences tend to serve all stops in a zone before moving to the next zone, both at the nano and micro levels. In order to generate candidate sequences that follow this same pattern, we apply RI on a modified travel time matrix, in which travel times between stops from different nano and micro zones are penalized by multiplicative factors \(1 + \pi_n\) and \(1 + \pi_m\), respectively, where \(\pi_n, \pi_m > 0\) are parameters of the framework (see Appendix A).
Algorithm 2 \texttt{POOLSEQUENCES}

\begin{algorithmic}
\State \textbf{Input:} Route $h$, structural info $I = \{\delta_M, \delta_m, \delta_n\}$, pool size $P$
\State \textbf{Output:} Set of sequences $\Lambda$
\State $E^+ \leftarrow \{s \in S(h) : \delta_n(d(h), z_n(s)) \geq 1\}$ \Comment{Set of candidate “entry” dropoff stops}
\State $E^- \leftarrow \{s \in S(h) : \delta_n(z_n(s), d(h)) \geq 1\}$ \Comment{Set of candidate “exit” dropoff stops}
\If{$E^+$ is empty} \Comment{Initialize the sequence pool}
\State $E^+ \leftarrow S(h) \setminus \{\pi(h)\}$
\EndIf
\If{$E^-$ is empty} \Comment{Select random entry and exit dropoff stops}
\State $E^- \leftarrow S(h) \setminus \{\pi(h)\}$
\EndIf
\State $\Lambda = \emptyset$
\While{$|\Lambda| < P$} \Comment{Constructs a sequence by applying random insertion heuristic}
\State $e^+, e^- \leftarrow \textsc{SelectRandom}(E^+, E^-, \delta_n)$
\State $\lambda \leftarrow \textsc{RandomInsertion}(e^+, e^-)$
\State $\Lambda \leftarrow \Lambda \cup \{\lambda\}$
\EndWhile
\State \textbf{return} $\Lambda$
\end{algorithmic}

3.4. Learning Framework

The learning framework is summarized in Algorithm 3. First, we initialize a regression model (line 1) and split the high score training routes into $B$ batches (line 2), where $B \in \mathbb{N}^+$ is a parameter (see Table A.2 for the complete list of parameters and their settings). For each batch, we first collect structural information of the training data (line 4), using Algorithm 1. Next, for each training route in the batch, we generate a pool of candidate sequences with the insertion method described in Algorithm 2 and compute statistics on these sequences (lines 6-7). Only a small number of sequences is kept in the pool (line 8) to avoid an unnecessarily large regression model. For each sequence left in the pool, we compute its score with respect to the actual sequence in the training route (line 10). This score is computed with the Levenshtein distance-based algorithm used to evaluate sequences in the challenge. Next, we extract sequence features and add a data point to the regression model (lines 11-12). In addition to the sequence, the feature extraction procedure \textsc{Features} receives as input the training route, pool statistics and structural information in order to generate a large set of covariates and improve the prediction power of the model. As output, procedure \textsc{Features} returns a feature vector $x \in \mathcal{F}$, where $\mathcal{F}$ is the feature space. All features used in the score prediction model are detailed in \textit{Appendix B}. Once all batches are processed, the score prediction model is solved (line 13) and structural information of the entire training set is collected (line 14). The output of the learning phase consists of the prediction function $\hat{g} : \mathcal{F} \rightarrow \mathbb{R}$, which maps the feature space to a numerical score, and the structural information about the training set.

The score prediction model $\mathcal{M}$ is solved by Lasso regression \cite{Tibshirani:1996}. The regularization hyperparameter is selected based on a grid search procedure using cross validation data. Since the Lasso method also performs variable selection, it is particularly useful in our case as the feature space $\mathcal{F}$ is high-
Algorithm 3 Learning Framework (“model-build” phase)

**Input:** Training set $H^{TR}$  
**Output:** Prediction function $\hat{g}(\cdot)$, structural info $I^{TR}$

1: $\mathcal{M} \leftarrow \emptyset$  \hspace{1cm} \triangleright \text{Initialize score prediction (regression) model}
2: $H^{(1)}, \ldots, H^{(B)} \leftarrow \text{Split}(H^{TR}, B)$  \hspace{1cm} \triangleright \text{Split high score routes in } H^{TR} \text{ into } B \text{ batches}
3: \textbf{for } $b : (1, \ldots, B)$ \textbf{do}
4: \hspace{1cm} $I \leftarrow \text{CollectInfo}(H^{TR} \setminus H^{(b)})$  \hspace{1cm} \triangleright \text{Structural info on } H^{(b)} \text{ is ignored when processing batch } b$
5: \hspace{1cm} \textbf{for } h : H^{(b)} \textbf{ do}
6: \hspace{1.1cm} $\Lambda \leftarrow \text{PoolSequences}(h, I, P^{TR})$  \hspace{1cm} \triangleright \text{Pool } P^{TR} \text{ candidate sequences for } h$
7: \hspace{1.1cm} $S \leftarrow \text{Statistics}(\Lambda)$  \hspace{1cm} \triangleright \text{Compute statistics on } \Lambda$
8: \hspace{1.1cm} $\text{Resize}(\Lambda, \bar{p})$  \hspace{1cm} \triangleright \text{Keep only } \bar{p} \text{ sequences in } \Lambda$
9: \hspace{1.1cm} \textbf{for } $\lambda : \Lambda$ \textbf{ do}
10: \hspace{1.2cm} $y \leftarrow \text{Score}(\lambda, h)$  \hspace{1cm} \triangleright \text{Compute score of sequence } \lambda \text{ with respect to } \lambda^{A}(h)$
11: \hspace{1.2cm} $x \leftarrow \text{Features}(\lambda, h, S, I)$  \hspace{1cm} \triangleright \text{Extract features to vector } x$
12: \hspace{1.2cm} AddDataPoint($\mathcal{M}, x, y$)  \hspace{1cm} \triangleright \text{Add to } \mathcal{M} \text{ data point with input } x \text{ and output } y$
13: $\hat{g}(\cdot) \leftarrow \text{Solve}(\mathcal{M})$  \hspace{1cm} \triangleright \text{Solve score prediction model}
14: $I^{TR} \leftarrow \text{CollectInfo}(H^{TR})$  \hspace{1cm} \triangleright \text{Collect structural info from the entire training set}$
15: \textbf{return } \hat{g}(\cdot), I^{TR}$

dimensional (over 1,000 features, see Appendix B). In addition to improving interpretability and reducing model prediction variance, a small number of relevant features also improves the efficiency of the sequence selection algorithm discussed in the next section.

3.5. Sequence Selection

Given a test route $h \in H^{TE}$, the procedure to prescribe a sequence $\lambda \in \Lambda(h)$ is described in Algorithm 3. Considering the structural information acquired from training data, we first generate a pool of candidate sequences (line 1). Then, we compute statistics on the generated pool (line 2), as some sequence features are better assessed relative to other features. For example, instead of taking the sequence duration in isolation, it is more sensible to use as a score predictor variable the ratio between the sequence duration and the minimum duration among the sequences in the pool. Next, we remove from the pool sequences that are not likely to yield a good score (lines 3-4). The criteria used in this step is the ratio between the sequence similarity (to training data) at the nano zone level and the number of nano zone transitions in the sequence. By keeping only promising sequences in the pool, we speed up the last step (line 5), which requires extracting features and evaluating the score prediction function for each sequence left in the pool.
Algorithm 4 Sequence Selection (“model-apply” phase)

**Input:** Test route $h$, prediction function $\hat{g}(\cdot)$, structural info $I^TR$

**Output:** Proposed sequence $\lambda^*$

1: $\Lambda \leftarrow$ POOLSEQUENCES($h, I^TR, P^TE$)  \hspace{1cm} $\triangleright$ Pool $P^TE$ candidate sequences for $h$
2: $S \leftarrow$ STATISTICS($\Lambda$)  \hspace{1cm} $\triangleright$ Compute statistics on $\Lambda$
3: $\alpha = \max_{\lambda \in \Lambda} \Delta_n(\lambda|I^TR)/T_n(\lambda)$  \hspace{1cm} $\triangleright$ Maximum similarity by transition (nano level)
4: $\Lambda \leftarrow \{\lambda \in \Lambda : \Delta_n(\lambda|I^TR)/T_n(\lambda) \geq \eta \alpha\}$  \hspace{1cm} $\triangleright$ Heuristically prune unpromising sequences from $\Lambda$
5: $\lambda^* \leftarrow \arg \min_{\lambda \in \Lambda} \hat{g}(\text{FEATURES}(\lambda, h, S, I^TR))$  \hspace{1cm} $\triangleright$ Select from $\Lambda$ the sequence with the best predicted score
6: return $\lambda^*$

4. Results and Conclusions

4.1. Validation Datasets

To evaluate the performance of our framework, we create five validation datasets (datasets A-E) by sampling and moving high score routes from $H^TR$ to a validation set $H^VA$. To this end, we apply a stratified sampling method. First, we partition all high score routes from $H^TR$ into a set of macro zone buckets $\{B_z\}_{z \in Z_M}$, where $Z_M$ is the set of all macro zones. A route $h$ belongs to bucket $B_z'$ if the main macro zone of $h$ is $z'$, that is, if $\sum_{s \in S(h)} [z_M(s) = z'] \geq \sum_{s \in S(h)} [z_M(s) = z]$ for all $z \in Z_M$ (ties broken arbitrarily). Then, given parameters $S_t \in \mathbb{N}^+$ (sampling threshold) and $S_r \in (0, 1)$ (sampling ratio), we select randomly $\min\{1, S_r |B_z|\}$ routes from each bucket $B_z$ such that $|B_z| \geq S_t$, and move those routes to $H^VA$. Since our framework relies heavily on structural information about entry and exit regions and zone transitions, this stratified sampling strategy ensures that most zones are well represented in both training and validation sets.

4.2. Summary of Results and Discussion

Table 1 summarizes the results obtained on the validation datasets A-E. Columns $S_t$ and $S_r$ indicate the sampling threshold and sampling ratio, respectively, used to generate each dataset. Column “MSE” reports the mean squared error of the score prediction model when evaluated in out-of-sample sequences in the learning phase. The group of columns “Score” reports the actual scores obtained in test routes. Because of the random insertion heuristic, our method is intrinsically probabilistic. For this reason, we evaluate the algorithm in expectation by solving each dataset 10 times.

In total, the computational experiments on datasets A-E generated 11,050 proposed sequences. At the individual route level, the distribution of scores varies depending on certain route features. As shown in Appendix C (Figure C.1), the framework prescribes better sequences in some stations (e.g., "DSE4" and "DSE5") than in others (e.g., "DBO1" and "DCH2"). We also notice better performance in routes with a relatively low number of time windows. The mean score obtained in routes with up to seven time windows (5,410 routes) is 0.0396 (16.3% below the average), while the mean score in routes with eight or more time windows...
windows (5,640 routes) is 0.0547 (15.7% above the average). Figure C.2 compares the score distributions of the two route groups.

| Dataset | $S_t$ | $S_r$ | High Score Routes | Model $|H^VA|$ | Score | Average | SD | CV |
|---------|-------|-------|------------------|--------|-------|---------|----|----|
| A       | 2     | 6%    | 2,516            | 202    | 0.00118 | 0.0481  | 0.00035 | 0.0072 |
| B       | 4     | 7%    | 2,522            | 196    | 0.00109 | 0.0501  | 0.00107 | 0.0213 |
| C       | 6     | 8%    | 2,501            | 217    | 0.00137 | 0.0451  | 0.00080 | 0.0177 |
| D       | 8     | 9%    | 2,485            | 233    | 0.00114 | 0.0492  | 0.00046 | 0.0093 |
| E       | 10    | 10%   | 2,461            | 257    | 0.00129 | 0.0444  | 0.00088 | 0.0198 |
| Average |       |       | 2,497            | 221    | 0.00121 | 0.0474  | 0.00071 | 0.0151 |

Table 1: Results on validation datasets A-E (10 runs per dataset)

Finally, we also investigate which features are most relevant for predicting the score of a sequence. Table C.6 lists the intercept term and the 15 features with the highest absolute average regression coefficients, where the average is taken over the five datasets. As we see, high quality (low score) sequences exhibit a low duration ratio, especially in routes (instances) where the nearest dropoff stop is close to the station. In addition, few zone transitions (at the nano and micro levels), good time window performance and high structural similarity to training sequences correlate with high quality (low score) sequences.

4.3. Further Improvements

We conclude this paper with some promising avenues for further improving the performance of the proposed framework:

(i) Given the tight timelines of the challenge and the relatively long time required for evaluating models and algorithms in expectation over a range of datasets, there has been no systematic tuning of several (hyper-)parameters of our framework (see Table A.2). Therefore, significant gains may be achieved by performing a more thorough tuning of parameters.

(ii) Although effective, the random insertion pooling algorithm used in our framework does not impose hard time windows. Given the relatively worse performance of our method in routes with many time windows, it may be beneficial to employ, in combination with random insertion, other pooling algorithms that exploit time window information to generate a higher quality sequence pool.

(iii) Our method does not leverage temporal information (e.g., day of the week and time of the day of routes), nor physical package information (e.g., dimensions), nor executor capacity, mostly because in the exploratory data analysis no clear relationship between sequence scores and such information could
be identified. Nevertheless, there might be relevant relationships that our model fails to capture since those features are not included.
Acknowledgment

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 754462.

Appendix A. Model and Algorithm Parameters

Table A.2 summarizes the model and algorithm parameters and their corresponding values as used in our implementation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t^B$</td>
<td>Travel time matrix multiplier (best-case scenario)</td>
<td>0.75</td>
</tr>
<tr>
<td>$t^W$</td>
<td>Travel time matrix multiplier (worst-case scenario)</td>
<td>1.25</td>
</tr>
<tr>
<td>$\sigma^B$</td>
<td>Service time multiplier (best-case scenario)</td>
<td>0.9</td>
</tr>
<tr>
<td>$\sigma^W$</td>
<td>Service time multiplier (worst-case scenario)</td>
<td>1.1</td>
</tr>
<tr>
<td>$\pi_n$</td>
<td>Travel time penalty factor for zone transition (nano level)</td>
<td>1.25</td>
</tr>
<tr>
<td>$\pi_m$</td>
<td>Travel time penalty factor for zone transition (micro level)</td>
<td>1.25</td>
</tr>
<tr>
<td>$B$</td>
<td>Number of batches</td>
<td>50</td>
</tr>
<tr>
<td>$P^{TR}$</td>
<td>Poolszie (training)</td>
<td>1,000</td>
</tr>
<tr>
<td>$P^{TE}$</td>
<td>Poolszie (testing)</td>
<td>5,000</td>
</tr>
<tr>
<td>$\overline{p}$</td>
<td>Number of data points to generate per training route</td>
<td>10</td>
</tr>
<tr>
<td>$\eta$</td>
<td>Similarity by transitions threshold for pruning unpromising sequences</td>
<td>0.925</td>
</tr>
</tbody>
</table>

Table A.2: Parameters

Appendix B. Score Prediction Model Features

Given a route $h$, structural information $I$, a sequence pool $\Lambda \subset \Lambda(h)$ and a candidate sequence $\lambda \in \Lambda$, the score prediction model predicts the score of $\lambda$ by considering not only features of $\lambda$ but also combinations of those features with statistics on $\Lambda$ and features of $h$. Tables B.3, B.4 and B.5 describe the pool statistics, route features and sequence features, respectively, that are used to generate the predictor variables of the model. In Tables B.4 and B.5, column “Included” indicates whether a feature is included in the final score prediction model. In addition to the marked features, all multiplicative interaction terms between route and sequence features are also included in the model, so that relevant non-linear relationships can also be identified.
Feature Definition

<table>
<thead>
<tr>
<th>Feature</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_duration</td>
<td>( \min_{\lambda \in \Lambda} D(\lambda) )</td>
</tr>
<tr>
<td>p_with_out_arrivals</td>
<td>(</td>
</tr>
<tr>
<td>p_wout_out_arrivals</td>
<td>( 1 - p_{\text{with out arrivals}} )</td>
</tr>
<tr>
<td>avg_earliness</td>
<td>( \sum_{\lambda \in \Lambda} E(\lambda) /</td>
</tr>
<tr>
<td>avg_lateness</td>
<td>( \sum_{\lambda \in \Lambda} L(\lambda) /</td>
</tr>
<tr>
<td>avg_totness</td>
<td>( \text{avg_earliness} + \text{avg_lateness} )</td>
</tr>
<tr>
<td>min_sim_by_trans_macro</td>
<td>( \min_{\lambda \in \Lambda} \Delta_M(\lambda</td>
</tr>
<tr>
<td>min_sim_by_trans_micro</td>
<td>( \min_{\lambda \in \Lambda} \Delta_m(\lambda</td>
</tr>
<tr>
<td>min_sim_by_trans_nano</td>
<td>( \min_{\lambda \in \Lambda} \Delta_n(\lambda</td>
</tr>
<tr>
<td>max_sim_by_trans_macro</td>
<td>( \max_{\lambda \in \Lambda} \Delta_M(\lambda</td>
</tr>
<tr>
<td>max_sim_by_trans_micro</td>
<td>( \max_{\lambda \in \Lambda} \Delta_m(\lambda</td>
</tr>
<tr>
<td>max_sim_by_trans_nano</td>
<td>( \max_{\lambda \in \Lambda} \Delta_n(\lambda</td>
</tr>
</tbody>
</table>

Table B.3: Statistics collected over a pool of sequences \( \Lambda \)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>station</td>
<td>( d(h) ) (one-hot encoded)</td>
</tr>
<tr>
<td>n_stops</td>
<td>(</td>
</tr>
<tr>
<td>n_tws</td>
<td>( \sum_{s \in S(h)} [\ell(s) - e(s) \leq 6 \text{ hours}] )</td>
</tr>
<tr>
<td>n_strict_tws</td>
<td>( \sum_{s \in S(h)} [\ell(s) - e(s) \leq 4 \text{ hours}] )</td>
</tr>
<tr>
<td>p_tws</td>
<td>( \text{n_tws/n_stops} )</td>
</tr>
<tr>
<td>p_strict_tws</td>
<td>( \text{n_strict_tws/n_stops} )</td>
</tr>
<tr>
<td>dropoff_nearest</td>
<td>Distance from station to nearest dropoff stop</td>
</tr>
<tr>
<td>dropoff_area</td>
<td>Area of the minimum bounding rectangle over the dropoff stops</td>
</tr>
<tr>
<td>dropoff_density</td>
<td>( \text{n_stops/dropoff_area} )</td>
</tr>
<tr>
<td>n_packs</td>
<td>Total number of packages</td>
</tr>
<tr>
<td>packs_per_stop</td>
<td>( \text{n_packs/n_stops} )</td>
</tr>
<tr>
<td>service_time</td>
<td>( \sum_{s \in S(h)} \sigma(s) )</td>
</tr>
<tr>
<td>distinct_macro</td>
<td>(</td>
</tr>
<tr>
<td>distinct_micro</td>
<td>(</td>
</tr>
<tr>
<td>distinct_nano</td>
<td>(</td>
</tr>
</tbody>
</table>

Table B.4: Features of a route \( h \)
<table>
<thead>
<tr>
<th>Feature</th>
<th>Definition</th>
<th>Included</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_duration</td>
<td>$D(\lambda)/\text{min_duration}$</td>
<td>✓</td>
</tr>
<tr>
<td>dur_by_stime</td>
<td>$D(\lambda)/\text{service_time}$</td>
<td>✓</td>
</tr>
<tr>
<td>earliness</td>
<td>$E(\lambda)$</td>
<td>✓</td>
</tr>
<tr>
<td>r_earliness</td>
<td>$E(\lambda)/\text{avg_earliness}$</td>
<td>✓</td>
</tr>
<tr>
<td>lateness</td>
<td>$L(\lambda)$</td>
<td>✓</td>
</tr>
<tr>
<td>r_lateness</td>
<td>$L(\lambda)/\text{avg_lateness}$</td>
<td>✓</td>
</tr>
<tr>
<td>totness</td>
<td>$E(\lambda) + L(\lambda)$</td>
<td>✓</td>
</tr>
<tr>
<td>max_stop_earliness</td>
<td>$\max_{s \in \lambda} E(s)$</td>
<td>✓</td>
</tr>
<tr>
<td>max_stop_lateness</td>
<td>$\max_{s \in \lambda} L(s)$</td>
<td>✓</td>
</tr>
<tr>
<td>max_stop_totness</td>
<td>$\max_{s \in \lambda} E(s) + \max_{s \in \lambda} L(s)$</td>
<td>✓</td>
</tr>
<tr>
<td>miss_early</td>
<td>${</td>
<td>s \in \lambda : E(s) &gt; 0}</td>
</tr>
<tr>
<td>miss_late</td>
<td>${</td>
<td>s \in \lambda : L(s) &gt; 0}</td>
</tr>
<tr>
<td>out_arrivals</td>
<td>${</td>
<td>s \in \lambda : E(s) &gt; 0 \lor L(s) &gt; 0}</td>
</tr>
<tr>
<td>p_out_arrivals</td>
<td>$\text{out_arrivals}/\text{n_tws}$</td>
<td>✓</td>
</tr>
<tr>
<td>sim_macro</td>
<td>$\Delta_M(\lambda</td>
<td>I)$</td>
</tr>
<tr>
<td>sim_micro</td>
<td>$\Delta_m(\lambda</td>
<td>I)$</td>
</tr>
<tr>
<td>sim_nano</td>
<td>$\Delta_n(\lambda</td>
<td>I)$</td>
</tr>
<tr>
<td>trans_by_distinct_macro</td>
<td>$T_M(\lambda)/\text{distinct_macro}$</td>
<td>✓</td>
</tr>
<tr>
<td>trans_by_distinct_micro</td>
<td>$T_m(\lambda)/\text{distinct_micro}$</td>
<td>✓</td>
</tr>
<tr>
<td>trans_by_distinct_nano</td>
<td>$T_n(\lambda)/\text{distinct_nano}$</td>
<td>✓</td>
</tr>
<tr>
<td>sim_by_trans_macro</td>
<td>$\Delta_M(\lambda</td>
<td>I)/T_M(\lambda)$</td>
</tr>
<tr>
<td>sim_by_trans_micro</td>
<td>$\Delta_m(\lambda</td>
<td>I)/T_m(\lambda)$</td>
</tr>
<tr>
<td>sim_by_trans_nano</td>
<td>$\Delta_n(\lambda</td>
<td>I)/T_n(\lambda)$</td>
</tr>
<tr>
<td>d_sim_by_trans_macro</td>
<td>$\Delta_M(\lambda</td>
<td>I)/T_M(\lambda) - \min_\text{sim_by_trans_macro}$</td>
</tr>
<tr>
<td>d_sim_by_trans_micro</td>
<td>$\Delta_m(\lambda</td>
<td>I)/T_m(\lambda) - \min_\text{sim_by_trans_micro}$</td>
</tr>
<tr>
<td>d_sim_by_trans_nano</td>
<td>$\Delta_n(\lambda</td>
<td>I)/T_n(\lambda) - \min_\text{sim_by_trans_nano}$</td>
</tr>
<tr>
<td>r_sim_by_trans_macro</td>
<td>$(\Delta_M(\lambda</td>
<td>I)/T_M(\lambda))/\max_\text{sim_by_trans_macro}$</td>
</tr>
<tr>
<td>r_sim_by_trans_micro</td>
<td>$(\Delta_m(\lambda</td>
<td>I)/T_m(\lambda))/\max_\text{sim_by_trans_micro}$</td>
</tr>
<tr>
<td>r_sim_by_trans_nano</td>
<td>$(\Delta_n(\lambda</td>
<td>I)/T_n(\lambda))/\max_\text{sim_by_trans_nano}$</td>
</tr>
</tbody>
</table>

Table B.5: Features of a sequence $\lambda$
### Appendix C. Additional Results

<table>
<thead>
<tr>
<th>Feature</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>(intercept term)</em></td>
<td>0.044</td>
<td>0.049</td>
<td>0.044</td>
<td>0.040</td>
<td>0.039</td>
<td>0.043</td>
</tr>
<tr>
<td>r_duration</td>
<td>0.109</td>
<td>0.113</td>
<td>0.090</td>
<td>0.087</td>
<td>0.105</td>
<td>0.101</td>
</tr>
<tr>
<td>r_duration × dropoff_nearest</td>
<td>−0.007</td>
<td>−0.049</td>
<td>−0.050</td>
<td>−0.049</td>
<td>−0.042</td>
<td>−0.040</td>
</tr>
<tr>
<td>trans_by_distinct_nano × packs_per_stop</td>
<td>0</td>
<td>0.042</td>
<td>0.031</td>
<td>0.041</td>
<td>0.042</td>
<td>0.031</td>
</tr>
<tr>
<td>max_stop_totness × distinct_nano</td>
<td>0.041</td>
<td>0.021</td>
<td>0.028</td>
<td>0.026</td>
<td>0.029</td>
<td>0.029</td>
</tr>
<tr>
<td>trans_by_distinct_micro × distinct_macro</td>
<td>0.028</td>
<td>0.023</td>
<td>0.028</td>
<td>0.013</td>
<td>0.029</td>
<td>0.024</td>
</tr>
<tr>
<td>p_out_arrivals × p_tws</td>
<td>0.037</td>
<td>0.017</td>
<td>0.036</td>
<td>0</td>
<td>0</td>
<td>0.018</td>
</tr>
<tr>
<td>trans_by_distinct_nano × distinct_nano</td>
<td>0.022</td>
<td>0</td>
<td>0.022</td>
<td>0.021</td>
<td>0.021</td>
<td>0.017</td>
</tr>
<tr>
<td>DAU1 × r_duration</td>
<td>0.017</td>
<td>0.016</td>
<td>0.014</td>
<td>0.012</td>
<td>0.019</td>
<td>0.016</td>
</tr>
<tr>
<td>sim_nano × service_time</td>
<td>−0.017</td>
<td>−0.010</td>
<td>−0.015</td>
<td>−0.014</td>
<td>−0.017</td>
<td>−0.014</td>
</tr>
<tr>
<td>p_out_arrivals × n_tws</td>
<td>0</td>
<td>0.015</td>
<td>0</td>
<td>0.025</td>
<td>0.022</td>
<td>0.012</td>
</tr>
<tr>
<td>trans_by_distinct_nano</td>
<td>0.029</td>
<td>0.013</td>
<td>0.011</td>
<td>0</td>
<td>0.009</td>
<td>0.012</td>
</tr>
<tr>
<td>sim_nano × n_stops</td>
<td>−0.020</td>
<td>0</td>
<td>−0.021</td>
<td>−0.017</td>
<td>0</td>
<td>−0.011</td>
</tr>
<tr>
<td>sim_by_trans_nano × n_stops</td>
<td>0</td>
<td>−0.019</td>
<td>−0.015</td>
<td>−0.007</td>
<td>−0.016</td>
<td>−0.011</td>
</tr>
<tr>
<td>trans_by_distinct_nano × p_strict_tws</td>
<td>0</td>
<td>0.013</td>
<td>0.009</td>
<td>0.011</td>
<td>0.016</td>
<td>0.010</td>
</tr>
<tr>
<td>max_stop_lateness × distinct_nano</td>
<td>0</td>
<td>0.016</td>
<td>0.001</td>
<td>0.018</td>
<td>0.012</td>
<td>0.010</td>
</tr>
</tbody>
</table>

Table C.6: Coefficients of the most relevant features for predicting the score of a sequence
Figure C.1: Score distribution by station

Figure C.2: Score distribution by number of stops with time window
References


Florio, da Costa and Özark XII.17


Article XIII

A Deep Bi-directional Long-Short Term Memory Neural Network-Based Solution for the Vehicle Sequencing Problem

Amr Eltawil and Sara Atef

Source code is available on GitHub:
https://github.com/saraatef/RC-CLI-KingTUT-Team
Abstract

Both the vehicle routing and sequencing problems are very similar in their nature. However, the vehicle routing problem mainly focus on obtaining a good route that minimizes/maximizes a certain objective, while the vehicle sequencing problem uses historical data to learn how a good route should be in the future time steps. Recently, many reinforcement learning-based approaches have been proposed to tackle the vehicle routing problem. However, there is an obvious research gap in using reinforcement learning based approaches for solving the vehicle sequencing problem. In this paper, a bi-directional long-short term memory network prediction model was used to solve the Vehicle sequencing Problem. The results shown the capability of the proposed learning model to develop high score routes based on historical data.

1. Introduction

The classical Vehicle Routing Problem (VRP) was first studied in literature by Dantzig, Fulkerson and Johnson [1] in 1954, where the authors studied a relatively large-scale travelling salesman problem (TSP). However, Clarke and Wright [2] in 1964, first incorporated more than one vehicle in the problem formulation. The VRP research accelerated since the 1980s, researchers developed more complex VRP problems to capture real-life operations and solve; the capacitated VRP (CVRP), Time constrained VRP (VRPTW), Pickup and delivery problem (PDP), Multi-depot VRP, and the Fleet size and mix VRP (FSMVRP). The typical objectives of the VRP are either to minimize the travelled distance or the traveling cost by finding the optimum route starting from a depot and visiting several stops, each with a known demand.
The Vehicle Sequencing Problem, on the other hand, uses historical data to learn how a good route should be in the future time steps [3]. Many RL-based approaches have been proposed to tackle the VRP [4]-[6]. However, there is an obvious research gap in using RL-based approaches for solving the VSP.

Building Vehicle Sequencing Problem (VSP) predictive models mainly depends on detecting the nonlinear nature of the corresponding variables that affect the deliveries. To handle the corresponding massive amount of data from the existing systems, more elaborate techniques than the traditional data analysis ones are on demand. For instance, every stop selection done by a certain vehicle generates data that would indicate the associated future delivery patterns. In order to address this issue, data-driven prediction approaches have been proposed. Instead of performing complicated analysis on the detailed data, data-driven techniques automatically learn the nature of the data from historical observations [7]. Recently, Artificial Intelligence (AI)-based techniques have been widely used for various forecasting problems as they enable machines to detect the human behavior. Machine Learning (ML) is one subset of the AI technique that can improve the machine performance by using experience obtained from previous actions [8]. The basic learning scenarios of ML includes supervised, unsupervised, and Reinforcement Learning (RL). These scenarios differ in the type of training and testing data and the mechanism of learning. In supervised learning, the learner receives labelled data to make prediction for all unseen points. This supervised learning scenario is commonly used for classification, regression, and ranking problems. However, in the unsupervised learning, the training data are unlabeled, and the learner is requested to make prediction for all unseen points. It is often used for clustering and dimensionality reduction problems. Nevertheless, in contrast with supervised and unsupervised scenarios, RL is an on-line approach in which the learner actively interacts with the environment and receives an immediate reward for each action. RL has many applications in solving the Vehicle Routing Problem (VRP) [4-6].

Among such recent approaches, using Artificial Intelligence (AI)-based techniques, especially the Long Short-Term Memory (LSTM) networks, which is one of the most recently introduced Deep Learning (DL) networks, has shown good prediction results in many forecasting problems. However, this paper is the first work, to the best of our knowledge, that considers solving the vehicle sequencing problem using an LSTM DL-based prediction model.

2. Literature Review

Recurrent Neural Networks (RNN) is one type of Artificial Neural Networks (ANN) that has a better performance on sequential data types, such as translation, text and speech recognition, and time-series data [9]. RNNs have the same characteristics as traditional ANNs. Additionally, RNNs consider a recurrent loop for each hidden state that uses a state vector to retain all the information about the past elements of this state.
Therefore, RNNs can gather the previous data along with the present task within a deep learning network [10]. Unfortunately, having an infinite dynamic loop in RNNs can result in many challenges related to vanishing or exploding of the gradients during the backpropagation process. In order to overcome this problem, an extended version of RNNs known as Long-Short Term Memory (LSTM) networks has been proposed [11]. LSTM networks have the advantage of being able to learn long-term dependencies. Unlike the RNNs, in which the neurons apply an activation function on a linear set of the network inputs, LSTM networks use a memory cell that contains input, forget, and output gates. Using those memory cells allows the network to eliminate the gradient vanishing/exploding problem. Several studies have been conducted using deep learning LSTM networks for prediction problems using time series forecasting, e.g., financial market [12], petroleum production [13], wind speed [14]–[16], photovoltaic power [17], [18], natural gas consumption [19], and electricity price prediction problems [20]. Generally, the main advantage of RNNs, especially LSTM ones, in comparison to the traditional ANNs, is the ability to remember historical information related to previous time steps, which is an essential feature for sequential prediction. The basic version of LSTM networks is known as the Unidirectional network, which learns the current state through the past states’ information only. However, getting benefits from future time sequences is an extra advantage of the Bi-directional LSTM (Bi-LSTM) networks. On the other hand, there is limited literature considering the performance of the Bi-LSTM networks in forecasting problems. Wang et al. [21] examined a Bi-LSTM based model with an attention mechanism for load prediction. The proposed model was compared with three extensions of traditional Uni-LSTMs and the Support Vector Regression (SVR) model and concluded that Bi-LSTM is an effective and feasible predictive model. Furthermore, an experimental study was introduced in [22] to assess the impact of using deep-stacked layers of both Uni-LSTM and Bi-LSTM predictive models for STLF. A hyperparameter tuning process was applied for the examined predictive models. The study showed that using deep-stacked layers did not improve the forecasting model accuracy and also required higher computational specifications. Besides, the proposed Bi-LSTM outperformed both Uni-LSTM and SVR models.

3. Methodology

As the solution efficiency is a prime target, we selected a progressive approach to solve the problem. The approach relies on using the minimum number of input features to solve the problem, then adding more input features progressively and observe the improvement that is taking place. Another aspect of the proposed approach is to use the LSTM neural network hyperparameter tuning in order to find the optimum settings of the hyperparameters used in the solution. In this paper, the input features were limited to the zone information, latitude, and longitude. In future work, the travel time and hyperparameter tuning will be implemented.

The proposed approach predicts the vehicle route sequences through three main data fields associated with each stop: zone information, latitude, and longitude, denoted as: ‘zone_id’, ‘lat’, and ‘lng’, respectively. These
data fields have been identified to be the simplest set that would possibly lead to a complete solution in a reasonable time given the problem size. Also, the latitude and longitude are both numerical values that intuitively lead to identify the spatial location of the stop and its proximity to the other stops. It is planned to extend this work to include more features that would improve the solution. A possible approach is to use feature selection optimization [23]. The travel time was not initially considered as it would be quite challenging because it will increase the solution set exponentially and will negatively impact the solution time. In future work, the travel time should be included and the results should be compared to identify the optimum set of features to be used to get the best solution in the available time. Therefore, the predictive model has three input features that would be used to predict the associated sequence of each stop in a route as the model output feature.

The input data are mapped into a 3D array with the shape of (sample: number of routes, time step; number of stops in each route; features: three input features of each stop). As the main objective of the RC-CLI challenge is to predict the route sequences associated with the high score routes, only the routes of “High Score” have been considered during the training process. The dataset is partitioned into 90% for training and 10% for testing. In addition, 10% of the training dataset has been utilized for validating the training process. The main task of the validation set is to examine the network performance during the training process. Therefore, for the proposed model, a total of 2446 routes were firstly used through the training process and 245 routes of the training dataset were applied to validate the developed model. Finally, 272 routes were used during the testing process by feeding them to the trained network and then calculating the corresponding predicted output. To measure the prediction performance accuracy, the predicted route sequences, obtained from the testing process, are compared with the actual values using the Mean Square Error (MSE) metric which can be calculated as follows.

\[
MSE = \frac{\sum_{i=1}^{z}(x_i - y_i)^2}{z}
\]  

(1)

Where, which \(x_i\) and \(y_i\) are the actual and predicted sequence values, respectively, and \(Z\) is the total number of observations.

All the input parameters have been normalized to reduce data redundancy. The normalization process is performed using the minimum and maximum scaler for all the training, validation, and testing datasets. However, the output feature has not been normalized to avoid negative values during performance evaluation.

The proposed LSTM deep learning predictive network sequential model structure is shown in Figure 1. The first layer is a sequence input layer, in which the input features are inserted in the deep learning network. The second layer represents the LSTM network layer configuration, which can efficiently remember long term
dependencies. There are two types of LSTMs, Unidirectional (Uni-LSTM) and Bidirectional (Bi-LSTM) networks. A fully connected layer is introduced after that to connect the LSTM layer outputs into one response that represents the predicted stop sequence. The last layer is a regression layer, which calculates the half-mean-squared-error loss for the sequence data.

In deep learning, the model is developed through the training process, and then it is evaluated using the testing process. In the training process, the proposed predictive model tries to fit the best values using an objective function, which minimizes the error between the actual and predicted data. The real data represent the label of the training dataset, while the predicted one describes the output obtained by the model resulting from the training process. To execute this training process, the proposed predictive model attempts to learn the model parameters at each iteration. An initial value is set for each of the model parameters, weights, and biases at the first iteration. After that, the improvement of these parameters is required. Thus, an optimization algorithm, called Adaptive moment estimation (Adam) [24] is used for updating the network adaptive learning rates during the training process. Adam can be considered as an extension for the classical stochastic gradient decent algorithm which maintains a fixed learning rates for all parameters during the training process based on the average first moment. However, in Adam, individual learning rates for various parameters are computed based on an estimation of both the first and second gradients moments. In specific, an exponential moving average for both the gradient and squared gradient is calculated by Adam, then the decay rates are controlled through beta parameters. The values of moving averages and beta parameters are initialized. Therefore, Adam is used to compute a gradient vector for each weight vector in the proposed LSTM based model. This gradient vector can deduce the error difference in case the weight vector has been adjusted by a particular value. This process is known as "backpropagation." Regarding the computed gradients, the weights are updated in the opposite direction, and the new error is calculated.
Figure 1 The proposed deep learning predictive network model structure

In traditional feedforward neural networks, each hidden neuron performs some calculations using inputs to obtain the corresponding outputs. However, Recurrent Neural Networks (RNN) apply the same process in combination with reusing the previous output. The input data for the next step are represented by the previous output and the new input. The power of this mechanism lies in its ability to keep a short-term memory about past information. In LSTM, an internal state is added to the recurrent hidden neuron. Thus, at each time step, the LSTM models use the input value, previous output, and the internal state. This internal state can keep, control, and update all the historical information of the earlier sequences related to the current state. In particular, the internal state has three gates, input (\(i_t\)), forget (\(f_t\)), and output (\(o_t\)) gates. The input gate determines whether the memory cell is updated or not, while the forget gate controls keeping or removing the memory cell. Finally, the output gate adjusts the current information visibility [11]. All those gates are using a sigmoid function, which is a non-linear activation function used mostly in feedforward neural networks and has non-negative derivatives (ranges from 0 to 1) [25], and they can be described as;
\[ i_t = \sigma(W_i [h_{t-1}, v_t] + b_i) \] (2)
\[ f_t = \sigma(W_f [h_{t-1}, v_t] + b_f) \] (3)
\[ o_t = \sigma(W_o [h_{t-1}, v_t] + b_o) \] (4)

where \( \sigma \) is the sigmoid activation function; \( W_i, W_o \) and \( W_o \) represent the weight matrix corresponding to each gate, respectively; \( h_{t-1} \) is the hidden state at the previous time step; \( v_t \) is the input value at time \( t \); \( b_i, b_o \) and \( b_o \) are the bias value of each gate, respectively.

A cell candidate (\( \hat{c}_t \)) is calculated at each time step to overwrite the memory cell. It is computed using the hyperbolic tangent activation function (tanh), which gives a better training performance than other activation functions for multi-layer neural networks [89]. It can be described as follows.

\[ \hat{c}_t = \tanh(W_c [h_{t-1}, v_t] + b_c) \] (5)

Finally, the current memory cell (\( c_t \)) and the current hidden state (\( h_t \)) are expressed as follows.

\[ c_t = f_t * c_{t-1} + i_t * \hat{c}_t \] (6)
\[ h_t = o_t * \tanh(c_t) \] (7)

It can be observed from Figure 1 that the Uni-LSTM represents a traditional LSTM layer, which learns to predict the current state using all the information acquired from previous states. On the other hand, a Bi-LSTM network can use historical data; also, it considers future data to predict the current state. In particular, at each time step \( t \), the Bi-LSTM network uses the previous hidden states history, computed within the forward flow, in addition to the information of the future hidden states, obtained through the backward flow. The Bi-LSTM network performs more complex calculations in comparison to the Uni-LSTM network and needs the entire sequence of data in advance.

The proposed Bi-LSTM network predictive model has the same model structure as the proposed Uni-LSTM model except for the current hidden state equation. As in the bidirectional network, both forward and backward LSTMs are considered. Therefore, the Bi-LSTM hidden state (\( H_t \)), which includes both forward (\( \overrightarrow{h}_t \)) and backward (\( \overleftarrow{h}_t \)) hidden states can be expressed as:

\[ \overrightarrow{h}_t = \text{LSTM}(h_{t-1}, v_t, c_{t-1}), t \in [1, T] \] (8)
\[ \overleftarrow{h}_t = \text{LSTM}(h_{t-1}, v_t, c_{t-1}), t \in [T, 1] \] (9)
\[ H_t = [\overrightarrow{h}_t, \overleftarrow{h}_t] \] (10)
Figure 2 demonstrates the pseudo code of the proposed input feature selection methodology illustrating the implementation steps of performing the proposed deep-learning prediction-based methodology using the proposed Bi-LSTM network. The proposed algorithm uses the historical location information dataset as the input and produces as outputs the optimized predicted sequence associated with each stop. First, the dataset is prepared as a set of input features (I) and their corresponding actual sequence output (O). Moreover, the dataset is partitioned into training and testing sets. Before starting the training process, the learnable parameter values are initialized and then the weight matrix of the training data is randomly initialized. Subsequently, the training process starts with computing both the forward and backward hidden states of the Bi-LSTM layer using equations (2)-(10). The hidden layers outputs are then compiled through the fully connected layer into a single vector; and the related mean-squared-error loss is computed through the regression layer. Based on this loss, the gradients of the weights are computed, and both the weights and biases are updated through backpropagation for the next training iteration. After completing the training iteration, the trained network is tested by using it for predicting the output of the testing data. The prediction error is then calculated as the MSE of the predicted values and the actual sequences. After reaching the maximum number of epochs (N), the optimized prediction model is obtained.

4. Results and Conclusions

The proposed Bi-LSTM network predictive model has been evaluated through three stages. First, during the training process, the proposed model has been tested using the tested dataset. The Mean Square Error (MSE) loss function has been optimized through 1000 epochs that represent the number of the training iterations, as shown in Figure 3. Also, the figure shows the model ability of well-fitting without any observed over-fitting nor under-fitting.

Second, the model has been tested through the model-apply stage by predicting the proposed sequences of 13 routes. The results indicated the feasible solution obtained by the proposed model represented by a sequence starting from each route depot and passing through all route stops.

Third, the proposed model has been implemented considering two main cases. First case: considering all 6112 routes provided; Second case: considering only the “High Score” routes which equal to 2718 routes.

Figure 3 illustrates the well-fitting performance of the proposed model during the training process for two considered cases. In addition, Table 2 illustrates the significant impact of considering only the “High Score” routes as in the second case regarding the average obtained score of (0.5292) in comparison with a higher average score of (0.7044) obtained from the first case as illustrated in Table 1.
Figure 2 The pseudo code of the proposed input feature selection methodology

Therefore, the final submission of the proposed model has considered only the “High Score” routes. Further model development can include considering more input features such as the travel time data which can significantly improve the prediction accuracy. Moreover, investigating the impact of hyperparameter optimization process is recommended for future work.
First Case (a)  

Second Case (b)  

Figure 3 loss estimation through the training process

<table>
<thead>
<tr>
<th>Route ID</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>RouteID_15baae2d-bf07-4967-956a-173d4036613f</td>
<td>0.9845935063751703</td>
</tr>
<tr>
<td>RouteID_3f166f0e-fd2e-47ab-96a0-6cbc99cc6eef</td>
<td>0.5933318644840936</td>
</tr>
<tr>
<td>RouteID_5486294a-503f-4346-b8a9-862e988cbe7c</td>
<td>1.008064295888041</td>
</tr>
<tr>
<td>RouteID_693060a6-88bb-4324-9e9c-925d5240263c</td>
<td>0.44993204796815894</td>
</tr>
<tr>
<td>RouteID_7f5d87f0-c39f-434f-bf3f-b159ef321909</td>
<td>0.6740654472869116</td>
</tr>
<tr>
<td>Average score</td>
<td>0.7044243023873401</td>
</tr>
</tbody>
</table>

Table 2 Model score of the Second Case

<table>
<thead>
<tr>
<th>Route ID</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>RouteID_15baae2d-bf07-4967-956a-173d4036613f</td>
<td>0.9064604111010467</td>
</tr>
<tr>
<td>RouteID_3f166f0e-fd2e-47ab-96a0-6cbc99cc6eef</td>
<td>0.47322758790472924</td>
</tr>
</tbody>
</table>
RouteID_5486294a-503f-4346-b8a9-862e988cbe7c & 0.45643391228260244 \\
RouteID_693060a6-88bb-4324-9e9c-925d5240263c & 0.37477212503841306 \\
RouteID_7f5d87f0-c39f-434f-bf3f-b159ef321909 & 0.549442439194562 \\
Average score & 0.5292130456326196 \\

References


Article XIV

Learning Routing Models with Cluster Precedence Constraints

Ivan Contreras and Dang Le

Source code is available on GitHub:
https://github.com/DangLe1996/AmazonLastmileChallenge
Learning Routing Models with Cluster Precedence Constraints

Ivan Contreras, Dang Le
Concordia University and Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), Montreal, Canada H3G 1M8

Abstract

We present a data-driven routing algorithm for the Amazon Last Mile Routing Research Challenge. To determine the stop sequencing decisions, we consider a clustered asymmetric traveling salesman model with precedence constraints. Delivery stops are assumed to be partitioned into a set of mutually exclusive clusters and the goal is to find a minimum cost Hamiltonian circuit in which all stops of each cluster are visited contiguously. Clusters have to satisfy a predefined set of precedence constraints learned during the model build phase using decision trees. To generate routes during the model apply phase, we develop a fast two-stage heuristic algorithm. The first stage focuses on determining cluster sequencing decisions in a way that learned precedence constraints are respected. We use a constructive heuristic to generate a set of promising cluster sequences satisfying precedence constraints. The second stage deals with the stop sequencing decisions considering as input the cluster sequencing decisions from the first stage. To do so, we use a greedy constructive heuristic and local search procedure that explores feasible 3-opt and 4-opt moves. The proposed heuristic provides, in less than five minutes of computational time, an average route score of 0.0293 using a six-fold cross validation scheme with all 6,010 high and medium quality routes given in the historical data.

1. Introduction

During the course of this research challenge, we studied and analyzed several strategies that can leverage tacit knowledge given in known delivery data to develop a data-driven routing algorithm capable of generating high quality routes. Our selected method is inspired by the novel deep inverse optimization framework by Tan et al. [2019]. Inverse optimization (IO) seeks to learn unknown parameters of optimization models from observed optimal solutions [Ahuja and Orlin 2001]. IO is receiving increasing attention as an effective data-driven methodology for inferring input parameters of optimization models and can be seen as a form of deep learning [Tan et al. 2019, Tan 2021]. IO is a powerful tool that can also be used to learn optimization models from optimal decisions [Tan et al. 2020].
From a machine learning (ML) perspective, our considered IO approach can be stated as follows. Given a set of features (e.g., travel times, date, depot, vehicle capacity, package dimensions, planned service time) and a set of decision targets (e.g., observed routes) determined by an unknown optimization process, we seek to learn a family of routing models that provides near-optimal decisions for the set of high quality observed routes. In order to infer a routing model from historical data, we may define a suitable hypothesis space parameterized by a set of features. Even for a given hypothesis point, learning a model that makes the solutions feasible and near-optimal is a complex integer bilevel problem that poses several methodological and computational challenges. Given the time limitations of this competition, we use as a base model a clustered routing model. Our goal is to learn the objective function coefficients and constraints to provide near-optimal routes by exploiting the underlying structure of the observed routes.

The remainder of this paper is as follows. Section 2 provides a succinct literature review on related work. Section 3 describes the proposed methodology and overall routing algorithm used for the final submission. Finally, Section 4 provide the results of computational experiments and highlight future research directions.

2. Literature Review

This challenge focuses on a complex routing problem. Generally speaking, given a delivery vehicle and a set of stops that need to be served, the problem consists of finding a Hamiltonian circuit visiting each stop exactly once. When the quality of the route is defined only by its cost (or length) and such cost is assumed to be symmetric, the problem corresponds to the well-known symmetric traveling salesman problem (TSP). [Applegate et al., 2006] provides a description of the state-of-the-art exact TSP solver - the Concorde algorithm. It is a branch-and-cut algorithm capable of solving very large-scale instances to proven optimality. [Helsgaun, 2000] and [Taillard and Helsgaun, 2019] describe some of the most sophisticated implementations and extensions of Lin-Kernighan-based heuristics for finding high quality solutions for very large-scale instances, whereas [Nagata and Kobayashi, 2013] presents a powerful genetic algorithm that does not rely on a Lin-Kernighan-based local search. If the costs are asymmetric, as it is the case for the travel times given on this challenge, the problem correspond to the so-called asymmetric TSP (ATSP). [Kanellakis and Papadimitriou, 1980] present an effective adaptation of the Lin-Kernighan TSP heuristic for the ATSP. [Roberti and Toth, 2012] surveys the most effective models and exact algorithms for solving the ATSP. We note that any instance of the ATSP can be transformed into an instance of the STP by using a 2-node transformation [see, Jonker and Volgenant, 1983].

When the cost of travel between stops depends on the time at which vehicles arrives at each stop (or their position in the sequence), the problem generalizes to the time-dependent TSP [Vander Wiel and Sahinidis, 1995]. If stops have to be visited within a specified time window, the problem correspond to the TSP with time windows [Baldacci et al., 2012] and to the even more general time-dependent TSP with time windows [Vu et al., 2020]. However, none of these problems are capable of capturing the underlying structure of delivery routes given in the historical data. Moreover, taking into account Amazon’s real-life operations...
described in this competition, the quality of a route should not only be defined by its cost but also consider experience-based expectations of dynamic traffic conditions, availability of safe and convenient parking, and customer availability. Therefore, more complex functions are needed to better approximate observed routes.

A close look at the sequencing structure of historical routes revealed that the vast majority follow the following pattern. If a vehicle enters to a given zone, it tends to visit all stops of that zone consecutively before moving to the next zone. This sequencing structure is captured by a TSP variant known as the clustered TSP (CTSP) [Chisman, 1975, Laporte and Palekar, 2002], in which delivery stops are assumed to be partitioned into a set of mutually exclusive clusters. The CTSP consists of determining a minimum cost Hamiltonian cycle (or circuit) in which the stops of any cluster are contiguous. The CTSP can be transformed into a TSP by adding an arbitrarily large constant $M$ to inter-cluster costs, or by subtracting $M$ from intra-cluster costs. Lu et al. [2020] provide a computational study to compare several state-of-the-art TSP solvers with adhoc heuristic algorithms specifically designed for the CTSP. The results on a set of benchmark instances show that the genetic algorithm of Nagata and Kobayashi [2013] and the LK-based heuristic of Helsgaun [2000] are capable of producing better solutions with shorter computational times as compared to specialized CTSP heuristics. Our proposed solution algorithm does not use any of these algorithms given that they are not released under the licensing agreements considered in this competition. Instead, we describe in the next section our own constructive and local search heuristics for an extension of the CTSP.

In the CTSP, it is assumed that there are no predefined sequencing constraints among different clusters. However, a deep analysis of the zone sequencing structure in observed routes revealed that the majority of routes followed specific zone sequencing patterns that could not be modeled with the CTSP. As a consequence, in this paper we study an extension of the CTSP in which precedence constraints are incorporated into the CTSP. We refer to this problem as the clustered ATSP with precedence constraints (CTSPP). To the best of our knowledge, this specific TSP extension has not been studied in the literature.

Regarding the considered solution method, we next highlight how our work differs with respect to the recent work in IO. In particular, Tan et al. [2019] presents an iterative optimization process that use back-propagation through an interior point algorithm that can learn the coefficients of the cost vector and constraints for both parametric and non-parametric linear programs. In our case, the models we are interested in learning are combinatorial rather than continuous. As a consequence, we cannot directly use such deep inverse optimization framework in our work. Tan et al. [2020] propose a flexible gradient-based framework for learning linear programs from optimal decisions. The authors propose a bilevel formulation of the considered inverse linear optimization problem which is then approximately solved using sequential quadratic programming techniques. During extensive preliminary analysis and experimentation, we developed a similar bilevel formulation for our combinatorial inverse optimization model and provided several single-level linear reformulations. Our goal was to learn the cost coefficients of the CTSPP. Unfortunately, this line of research requires an in-depth analysis of the hypothesis space and potential families of cost coefficient functions that exceeds the time given for this competition. As a consequence, for the final submission we
focused on learning precedence constraints using other more tractable ML techniques such as decision trees [Hastie et al., 2009]. Finally, instead of using loss functions commonly used in IO such as decision or objective errors [Babier et al., 2020], we use the route score function provided in this challenge.

3. Methodology

Similar to other ML frameworks, our IO approach requires two main phases: model build and model apply. During the model build phase, our algorithm aims to learn zone precedence constraints, some of them involving disjunctive conditions. During the model apply phase, we then incorporate the learned precedence constraints to the CTSPP to generate sequencing decisions that mimic observed patterns of historical routes. To approximately solve the CTSPP, we develop a fast two-stage heuristic algorithm.

This section is structured as follows. We first present a summary of our analysis of the zone sequencing structure of observed routes. We then provide the details of the model build phase, including a data cleaning step and the considered decisions treess for learning cluster precedence constraints. Finally, we provide the details of the model apply phase, including a formal definition and a mathematical programming formulation for the CTSPP and the proposed constructive and local search heuristics.

3.1. An Analysis of the Sequencing Structure of Observed Routes

There are a total of 6,112 historical routes given in the competition. These can be classified into high, medium and low quality routes. Taking into account their coordinates and zoneID’s, we identified super-clusters of stops associated with five different major urban agglomerations in the contiguous United States. Table A.1 provides descriptive information on the historical delivery routes per urban agglomeration, each of them identified with its main city from now on.

After plotting several observed routes, we realized Zone IDs encode an underlying hierarchical structure of the geographical planning areas considered by Amazon in each city. We conjecture that cities are partitioned into several Regions, each of them further partitioned into several Areas. Each area within each region contains exactly three Zones. Figure 1 describes how regions, areas and zones unique identifiers are encoded in Zone IDs. For example, given zone ID H-18.1D, the first three alphanumerical characters (i.e., H-18) provide the region ID, the last alphabetical character (i.e., D) provides the area ID, and the second to last numerical character (i.e., 1) provides the actual zone ID. Moreover, we note that zones of the same area are contiguous as well as area pairs of the same region. Delivery stops always have a zone ID associated with them and each zone ID can be served by one or more depots from the same city.

![H-18.1D](Figure 1: Identification of Regions, Areas, and Zones in Amazon zoning.)
The following observations summarize our findings on sequencing patterns at different levels. They were obtained after performing an in-depth analysis of the sequencing structure of observed routes using data visualization tools available in open-source Python packages and Google maps.

1. In the majority of routes, when a vehicle enters a zone it tends to visit all stops of that zone consecutively before serving stops of different zones. There are clearly cases in which a vehicle enters and leaves multiple times the same zone. However, the proportion of zones which are served with almost perfect zone clusters is sufficiently large to justify the use of this hypothesis in our model.

2. When a delivery vehicle enters a given area, it tends to visit all three zones of such area before moving to another area. Moreover, for a significant number of routes the zone sequencing decisions within an area tend to follow an increasing (or decreasing) numerical order. For example, if a vehicle enters area B, it will likely visit the zones in either the order 1B → 2B → 3B, or 3B → 2B → 1B.

3. When a vehicle enters a given region, it tends to visit all areas of such region before moving to another region (if any). Moreover, for a significant number of routes the area sequencing decisions inside a region tend to follow a increasing (or decreasing) alphabetical order. For example, if a vehicle enters region A-4 and has to visit stops in areas: B, C, D, E, G, H, it will likely visit the areas in either the order B → C → D → E → G → H, or H → G → E → D → C → B.

4. Moreover, when a vehicle transitions from one area to the next, it tends to reverse the numerical ordering when visiting the zones. For example, if a vehicle transitions from area B to area C, it will likely visit the zones of these two areas in either the order 1B → 2B → 3B → 3C → 2C → 1C, or 3B → 2B → 1B → 1C → 2C → 3C.

5. If a vehicle has to visit two or more regions in the same route, it tends to visit all areas and zones of one region consecutively before serving stops of different regions. Moreover, for a significant number of routes, the region sequencing decisions tend to follow an increasing (or decreasing) alphanumerical order which depends on the associated depot. For example, if a vehicle has to visit A-4 and A-3 regions, it will either likely visit the regions in either the order A-4 → A-3, or A-3 → A-4.

Figure 2 depicts an example of the observed zone sequencing decisions for route RouteID_84f72f6e-3b04-4883-8e14-632fba58a7f0 associated with depot DLA5 in Los Angeles, CA. To simplify the illustration of the above sequencing patterns, the number of stops as well as the size, shape and spatial interaction of the regions, areas and zones depicted in this figure do not match the actual ones. There are more visited areas in the actual route, however, this figure only show a subset of areas and zone that the driver visited for this route. We also want to highlight that the driver most of the time only visit subset of areas and zone in any given region and thus this is the behavior we aim for.

We stress that the main objective of the above identified patterns is to provide some base guidelines on how zone sequencing decision should be made. There are numerous observed routes in which a subset (or all) of these patterns do not apply. We noted that one common reason on why these patterns would not apply
is when there exist a significant number of stops with time windows constraints. Another common reason is when serving densely populated and commercial areas. Finally, we conjecture that all above identified sequencing patterns are likely a direct consequence of the base algorithms used by Amazon to partition stops that need to be served each day to create a set of vehicle routes.

In the next section, we describe how ML techniques can be used during the model build phase to identify and learn these sequencing patterns from observed routes. These learned patterns are used to derive precedence constraints that are then used during the model apply phase.

3.2. Model Build Phase

The goal of our model build phase is to identify a set of region and area precedence constraints for each depot observed in the historical data. Given that our model build and apply phases heavily rely on the zone IDs given in the input files, we need to make sure all stops have zone IDs that conform the template described in the previous section. We note that a very small portion of stops in the input file have either missing or non-conforming zone IDs. Therefore, we use a $k$-nearest neighbor model with the stop’s longitude and latitude values as features. In particular, when there exist a stop $s$ without a correct zone ID, we find the 5-nearest neighbors with respect to the Euclidean distance metric and assign $s$ to the most dominant zone ID among these neighbors. This step employs the $KNeighborsClassifier$ implemented in scikit-learn [Pedregosa et al., 2011]. In what follows, we describe the considered decision tree learning models.

Decision tree learning is a method commonly used in data mining [Rokach and Maimon, 2008]. The goal is to create a model that predicts a target value based on several input variables. We use scikit-learn optimized version of the classification and regression trees (CART) algorithm to produce the classification
trees [Pedregosa et al., 2011]. One shortcoming of the current scikit-learn implementation is the lack of support for categorical variables. To circumvent this limitation, we use the Python built-in hash function to convert string variables into integers.

During the model build phase, we build two decision tree classifiers for each observed station code (i.e., depot). The first classifier stores the region precedence constraints and the second one stores the area precedence constraints. Algorithm 1 provides a pseudo-code of the logic behind building the region classifier. For each input route, the model takes the first and last stop of the observed sequence denoted as src and dst, respectively. If src and dst belong to the same region code, denoted by src[0] and dst[0], then the model compares the two string values using a natural order and record the observed direction and region ID. We only record the first character of the zone ID to make the classifier more general. We note this does not affect the actual performance of the classifier.

For example, if a route start at stop with zone ID R-5.1G and finish at zone R-6.2C, then the region classifier would receive one observation that in region R, this route uses an ascending order. This observation can then be used during the model apply phase to predict interactions that we have not seen before during the model train phase. For instance, if a previously unseen route serves regions R-7 and R-8, we can predict that the route should visit R-7 and then R-8.

Algorithm 2 provides a pseudo-code for our area classifier model. Similar to Algorithm 1, this model also uses src as the zone ID of the first stop of the observed sequence. However, the dst variable is the last consecutive stop belonging to the same region and area as src. For example, if src zone ID is H-3.3G, then dst zone ID is the last consecutive stop with a zone ID that matches the pattern H-3.*G. Then, the model records the zone ID direction and uses it to construct the classifier.

3.3. Model Apply Phase

In this section, we first formally define the proposed routing model and provide a mathematical programming formulation. We then describe the two-stage heuristic algorithm used to generate feasible routes during model apply phase.

3.3.1. The Clustered Asymmetric Traveling Salesman Problem with Precedence Constraints (CTSPP)

Let $D = (V, A)$ be a complete directed graph where $V = \{0, v_1, \cdots, v_n\}$ is a node set partitioned into $m$ clusters $V_0, V_1, \cdots, V_m$, where $V_0$ is a singleton corresponding to the depot (node 0), and $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is an arc set. We denote as $C = \{1, \cdots, m\}$ to the cluster set and $T = \{0, \cdots, m\}$ to the set of positions in the cluster sequence, where the 0 position is always reserved for the depot. For each $(i, j) \in A$, we define an (asymmetric) arc cost $c_{ij}(u, w)$, that depends on historical delivery data and on new delivery data encoded via a vector of features $u$ and associated weights $w$. For each $k \in C$ and $t \in T$, we also define a sequence-dependent cluster cost $f_{kt}(u, w)$. The CTSPP seeks to determine a minimum cost Hamiltonian circuit on $D$ in which the nodes of any cluster $V_k$ are contiguous and cluster precedence constraints are satisfied. The main difficulty of the CTSPP stems from the inherent interrelation between
two level of the decision process. The first level considers the cluster sequencing decisions, whereas the second level deals with the node sequencing decisions.

For each \((i, j) \in A\), we define the binary variable \(x_{ij}\) equal to one if and only if the vehicle goes directly from node \(i\) to node \(j\). For each \(k \in C\) and \(t \in T\), we define the binary variable \(y_{kt}\) equal to one if and only if cluster \(k\) is assigned to position \(t\) at the cluster sequence level. Let \(E(S) = \{(i, j) \in A : i, j \in S\}\) denote the set of arcs incident to node set \(S\). The CTSPP can be stated with the following integer nonlinear program:

\begin{align*}
(\text{INLP}) & \text{ minimize } \sum_{(i, j) \in A} c_{ij}(u, w)x_{ij} + \sum_{k \in C} \sum_{t \in T} f_{kt}(u, w)y_{kt} \\
\text{subject to } & \sum_{(i, j) \in \delta(i)^+} x_{ij} = 1 \quad i \in V \\
& \sum_{(i, j) \in \delta(j)^-} x_{ij} = 1 \quad j \in V \\
& \sum_{(i, j) \in E(S)} x_{ij} \leq |S| - 1 \quad \forall S \subset V, 2 \leq |S| \leq n - 1 \\
& \sum_{(i, j) \in E(V_k)} x_{ij} = |V_k| - 1 \quad \forall k \in C, |V_k| \geq 2 \\
& \sum_{t \in T} y_{kt} = 1 \quad k \in C \\
& \sum_{k \in C} y_{kt} = 1 \quad t \in T \\
& \sum_{i \in V_k} \sum_{j \in V_t} x_{ij} = \sum_{t \in T} y_{kt-1}y_{kt} \quad \forall k, \ell \in C, k \neq \ell \\
& \text{Precedence constraints for all cluster pairs} \\
& x_{ij}, y_{kt} \in \{0, 1\} \quad (i, j) \in A, k \in C, t \in T.
\end{align*}

The objective minimizes the sum of the arc and cluster costs. Constraints (1)–(3) are usual constraints needed to model the ATSP, whereas constraints (4) ensure that nodes in the same cluster are visited contiguously. Constraints (5)–(6) are assignment constraints at the cluster-level sequence. They ensure each cluster is assigned to exactly one position and that each position contains exactly one cluster. Constraints (7) are (nonlinear) linking constraints to ensure the node sequencing decisions are aligned with the cluster sequencing decisions. Constraints (8) represent the precedence constraints for all cluster pairs obtained from the learned sequencing patterns during the model build phase. Finally, constraints (9) are the usual integrality conditions. During the first submission, we used a relaxation of the CTSPP in which constraints (5)–(8) and \(y_{kt}\) variables were not taken into account. Moreover, constraints (4) were indirectly considered with a intra-cluster penalization feature. Other features involving time windows constraints and other info given in historical data were used to define several linear families of arc-cost functions \(c_{ij}(u, w)\). However, experiments performed after the first submission revealed that direct incorporation of constraints (4), in combination with cluster precedence constraints (8), significantly outperformed the model used during the first submission.
3.3.2. A Two-stage Heuristic for the CTSPP

We next describe our two-stage heuristic algorithm to solve the CTSPP. It uses a sequential approach to construct feasible delivery routes by decomposing the cluster and node sequencing decisions. The first stage uses the learned region and area precedence constraints from the model build phase to determine cluster sequencing decisions. Given a fixed cluster sequence, the second stage uses a greedy deterministic constructive heuristic and a local search to optimize node sequencing decisions.

Algorithm 3 provides a pseudo-code for the first stage. The input consists of the region ordering given by the classifier and initial area and zone orderings for the first region that needs to be visited. The algorithm starts by clustering the stops of the input route with the same zone ID (excluding the depot). For each region, the algorithm then takes all area IDs present in the given route and orders them according to the current area order \( AO \). Finally, the ordering of zones within each area is done according to the current zone order \( ZO \). Following the patterns described in Section 3.2, the area and zone orders are reversed from ascending to descending (and vice-versa) when moving between different regions and areas, respectively.

The input of the second stage consists of the cluster sequencing decisions obtained from the first stage. During the constructive step of the second stage, we obtain an initial feasible node sequence by starting the route from the depot and visiting all nodes from the first cluster before moving to the next one. We use a closest insertion mechanism in which at every iteration, we identify the closest unvisited node within the current cluster with respect to the current node and visit it. Once we finish visiting all nodes of the current cluster, we move to the next cluster by identifying again the closest node that belongs to the next cluster. Once we finish visiting all nodes of all clusters, we finish the route by going back to the depot. This provides an initial solution which is then improved with a local search. We consider 3-opt and 4-opt neighborhoods and explore them sequentially. At every iteration of the local search, 3-opt and 4-opt neighborhoods consider only feasible routes (with respect to clustering constraints) which differ with respect to the current one by adding/subtracting exactly three and four arcs, respectively. We use a best improvement strategy and we only explore 4-opt moves when the current solution cannot be improved with a 3-opt move.

The overall two-stage heuristic is depicted in Algorithm 4. Given an input route \( r \), along with the region and area classifiers from the model build phase, the algorithm starts by determining the region ordering (RO) and area ordering (AO) of the first region that needs to be visited. Preliminary computational experiments showed our algorithm works best with the region ordering applied as a hard constraint, i.e., the region ordering should never change with respect to what is predicted by the region classifier. However, area ordering should be applied as a soft constraint. That is, although the area classifier identifies the order in which a set of areas of the same region should be served, we still check if it would be better to follow the opposite ordering. In order to override the classifier’s choice, we add a condition stating the improvement on the routing cost should be at least 2% as compared to the cost of the classifier’s ordering.
4. Results and Conclusions

Our data-driven algorithm was coded in a combination of Python and C and run on an Intel Xeon E5-2687W v3 processor at 3.10 GHz with a limit of 64 GB of RAM under Linux environment. To speed up the performance of our algorithm, we used the multi-core programming tools of Python Dask package. We evaluated the performance of the algorithm using a k-fold cross validation scheme with \( k = 6 \). During our computational experiments, we used all 6,010 high and medium quality routes provided in the competition. We observed there is no significant conflict in terms of precedence constraints in these two sets of routes. Each fold took about five minutes to run, including model build and model apply phases.

Table A.2 gives the results of our computational experiments using the six-fold cross validation, where the model is trained using about 5,000 routes and is then tested with about 1,000 routes. The minimum, average, and maximum obtained average route scores are 0.0278, 0.0293, and 0.0302, respectively. Figure A.3 shows the obtained route score histogram for the first-fold. From this figure, we note our algorithm performs very well on most of the test routes, with around half of the test routes having scores smaller than 0.018. We noted that in most of the routes were our algorithm performed very well, they conform the patterns described in Section 3.2. In the worst-case scenario, the score is as high as 0.32, which clearly indicates a rather poor performance on some of the more challenging routes. We noted that in most routes were our algorithm performed poorly, the observed routes either do not conform the predicted cluster patterns or there were several time windows constraints violations. In order to assess possible over-fitting issues, we run an experiment where we trained our algorithm on all medium and high quality routes and tested it in all high quality routes. The results are given in Figure A.4 and show that even if all high quality routes are used for both training and testing, the results are still very similar to the six-fold cross validation, with an average route score of 0.0278.

In order to assess the potential of our proposed model (i.e., the CTSPP), we run an additional set of experiments in which we assume we know the optimal clustering decisions. For each high quality route, we analyzed and recorded the observed cluster sequences. Whenever a cluster is visited more than once in the cluster sequence, we only keep the position in the cluster sequence for the largest block of stops of such cluster. We use this observed cluster sequence in the CTSPP to fix the associated \( y_{kt} \) variables to one (or zero) and solve the remaining subproblem (on the \( x_{ij} \) variables). We developed an exact branch-and-cut algorithm and implemented it using the Callable Library of CPLEX 20.1. In our opinion, the obtained results are remarkable. The average route score for all high quality routes is only 0.0065 when using the branch-and-cut algorithm to optimize the node sequencing decisions, and 0.0074 when using Algorithm 4 instead. This means that if we find an effective way to improve the cluster sequencing decisions, specially for routes in which our algorithm performs poorly, there is a potential to significantly improve our current best obtained results. On this line, we developed (and experimented) with a third neighborhood in our local search to modify the cluster sequencing decisions. However, a composite function that uses the total route cost and a penalization term for time windows constraints violations is not enough to effectively guide the
search to consistently improve route scores. We conjecture that more sophisticated composite functions are needed to significantly improve our results.

A potential line of research we are currently investigating is to identify a family of features to define arc-cost functions $c_{ij}(u,w)$ and sequence-dependent costs functions $f_{kt}(u,w)$ that can provide a better approximation for the observed high quality routes. Having access to expert-knowledge of drivers, would be highly beneficial for this endeavor. Another line we are interested in is to study how our approach can be extended to train using instances on a set of depots and test it on unseen depots. Our current learning mechanism relies on the assumption that the testing set will contain routes associated with regions and areas seen previously during training.
Appendix A. Appendix

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

Table A.1: Descriptive information on historical delivery routes per city

<table>
<thead>
<tr>
<th>k</th>
<th>Score</th>
<th>Test routes</th>
<th>Train routes</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0287</td>
<td>1.002</td>
<td>5.008</td>
<td>6.010</td>
</tr>
<tr>
<td>2</td>
<td>0.0278</td>
<td>1.002</td>
<td>5.008</td>
<td>6.010</td>
</tr>
<tr>
<td>3</td>
<td>0.0302</td>
<td>1.002</td>
<td>5.008</td>
<td>6.010</td>
</tr>
<tr>
<td>4</td>
<td>0.0297</td>
<td>1.002</td>
<td>5.008</td>
<td>6.010</td>
</tr>
<tr>
<td>5</td>
<td>0.0300</td>
<td>1.001</td>
<td>5.009</td>
<td>6.010</td>
</tr>
<tr>
<td>6</td>
<td>0.0297</td>
<td>1.001</td>
<td>5.009</td>
<td>6.010</td>
</tr>
</tbody>
</table>

Table A.2: Computational result using a six-fold cross validation
Algorithm 1: RCLF

**Result:** Region Decision Tree Classifier for depot D

**Input:** D - The station code, L - List of routes starting at depot D;

\( X \leftarrow \) empty list, \( y \leftarrow \) empty list;

**foreach** route \( r \) **in** \( L \) **do**

\( Q^* \leftarrow \) zone IDs of optimal sequence, \( src \leftarrow Q^*_1, dst \leftarrow Q^*_{-1}; \)

**if** \( src[0] == dst[0] \) **then**

**if** \( src \leq dst \) **then**

\( direction \leftarrow True \)

**else**

\( direction \leftarrow False \)

**end**

\( X += src[0] \)

\( y += direction \)

**else**

\( 1 \) **Continue**

**end**

**end**

\( clf \leftarrow \) DecisionTreeClassifier(classweight='balanced');

\( clf.fit(X,y); \)

Algorithm 2: ACLF

**Result:** Area Decision Tree Classifier for depot D

**Input:** D - The station code, L - List of routes starting at depot D;

\( X \leftarrow \) empty list, \( y \leftarrow \) empty list;

**foreach** route \( r \) **in** \( L \) **do**

\( Q^* \leftarrow \) zone IDs of optimal sequence, \( src \leftarrow Q^*_1; \)

\( idx \leftarrow \) last index of consecutive stops with same region and area \( src \), \( dst \leftarrow Q^*_{idx}; \)

**if** \( src \leq dst \) **then**

\( direction \leftarrow True \)

**else**

\( direction \leftarrow False \)

**end**

\( X += hash(src)[1:5] \)

\( y += direction \)

**end**

\( clf \leftarrow \) DecisionTreeClassifier(classweight='balanced');

\( clf.fit(X,y); \)
Algorithm 3: GetSequence

Result: Cluster sequence

Input: \( s \) - input route, \( RO \) - region order, \( AO \) - area order, \( ZO \) - zone order

\( S \leftarrow s.stops, \ zoneIDs \leftarrow s.zoneID, \ RegionCodes \leftarrow s.RegionCode, \ zoneIDList \leftarrow \) empty list;

Sort \( RegionCodes \) according to order \( RO \);

\hspace{1em} \textbf{foreach} region \( r \) in \( RegionCodes \) \textbf{do}

\hspace{2em} \( Q \leftarrow \) all areas in region \( r \);

\hspace{3em} Sort \( Q \) according to order \( AO \);

\hspace{4em} \textbf{foreach} area \( a \) in \( Q \) \textbf{do}

\hspace{5em} \( Z \leftarrow \) all zoneIDs in area \( a \);

\hspace{6em} Sort \( Z \) according to order \( ZO \);

\hspace{7em} \text{zoneIDList.append}(Z);

\hspace{7em} Reverse zone order \( ZO \);

\hspace{5em} end

\hspace{4em} end

\hspace{3em} Reverse area order \( AO \);

\hspace{1em} end

Algorithm 4: Model Apply

Result: Proposed stop sequence

Input: \( r \) - input route, \( RCLF \) - region classifier, \( ACLF \) - area classifier

\( RO \leftarrow RCLF.predict(r), \ AO \leftarrow ACLF.predict(r); \)

\( bestDist \leftarrow \infty, \ bestSequence \leftarrow \) empty list;

\hspace{1em} \textbf{foreach} initial area ordering \( ao \) in [acending, decending] \textbf{do}

\hspace{2em} \textbf{foreach} initial zone ordering \( zo \) in [acending, decending] \textbf{do}

\hspace{3em} \( ClusterSequence \leftarrow \) GetSequence\((r, RO, ao, zo)\);

\hspace{4em} \text{currentDist, StopSequence} \leftarrow \) SolveFixedCTSPP\((r, ClusterSequence)\);

\hspace{4em} \text{if} \ ao = AO \text{ then}

\hspace{5em} \text{currentDist} \leftarrow \text{currentDist} \ast 0.98;

\hspace{5em} \text{end}

\hspace{4em} \text{if} \ currentDist < bestDist \text{ then}

\hspace{5em} bestDist \leftarrow \text{currentDist};

\hspace{5em} bestStopSequence \leftarrow \text{StopSequence};

\hspace{5em} \text{end}

\hspace{3em} \text{end}

\hspace{2em} \text{end}

\hspace{1em} \text{Return} \ bestStopSequence
Figure A.3: One-fold route score histogram

Figure A.4: High quality route score histogram
References


Article XV

Distance Matrix Adjustment for Routing Problems

Mayukh Ghosh, A.V. Mahes and Donato Maragno

Source code is available on GitHub:
https://github.com/Roshanmahes/Amazon/MIT_LMRRC
Distance Matrix Adjustment for Routing Problems

Mayukh Ghosh, A.V. Mahes, Donato Maragno

Amsterdam Business School, Faculty of Economics and Business, University of Amsterdam
Plantage Muidergracht 12, 1018 TV Amsterdam, the Netherlands

Abstract

Different variants of the Traveling Salesperson Problem (TSP) and the Vehicle Routing Problem (VRP) are of interest to researchers for solving last mile delivery problems. But these approaches neither leverage historical data of routes nor include the driver behavior into the planning of future routes. In this work, a heuristic-based algorithm is proposed which leverages historical information from previously traveled routes to predict future routes. In the first stage the algorithm solves data-driven TSP to optimize the sequence of some predetermined geographical planning areas called zones, where the cost matrix in the objective incorporates the drivers’ previous decisions using historical routes. Then, again by solving TSP, the order of stops within each zone is optimized to get the final predicted route. Multiple simulations with varying sizes of training and test data set are performed to get an estimate of a reliable submission score. The algorithm gives a mean score of 0.0178 with the validation set provided by the organisers. Also, a mean score of 0.0581 was achieved when the training data provided is split into 1,000 routes for validation while the remaining is used to train the model.

1. Introduction

In the field of combinatorial optimization, the Traveling Salesperson Problem (TSP) (Johnson & Pilcher (1988)) and its generalization Vehicle Routing Problem (VRP) (Dantzig & Ramser (1959)) are two of the most well-known problems. Although many different variants of TSP and VRP have been stated (see e.g. Orman & Williams (2007)), the primary goal of the problem is to find the optimal routes for a fleet of vehicles to reach a given set of customers. As such, these types of problems are highly relevant to companies who work in the delivery services sector.

In real-world scenarios, the optimal route in terms of distance may not always be the optimal decision to take. Drivers, who mostly drive the same route, may have tacit information about the conditions at a ground level. Hence, drivers often deviate from the initial proposed route suggested by the company. Leveraging
this tacit information from historical routes can provide better route predictions to the drivers in the future. The goal lies in line with the idea of moving from just optimization to data driven predictive prescriptions as suggested by Bertsimas & Kallus (2020). To this end, we propose a simple heuristic-based solution to predict future routes using both past information and the estimated costs (distances and/or expected travel times).

Certain technical constraints were specified regarding the training time, inference time, and computational capabilities that can be used for this problem. We realized that open source solvers based on Python are not able to compute the optimal routes within a reasonable amount of time, and hence, have not been directly used for predictions. This is a problem faced in real-life execution of such models when routes need to be predicted (nearly) in real time. As such, we decided to break the problem down into two parts: the prediction of the sequence of zones and the prediction of the sequence of stops within each zone. Splitting the inference process into two parts allows us to leverage historical data within the optimization without affecting the TSP structure. In contrast with the stops, the zones are generally visited in multiple routes. Therefore, our aim is to learn from historical information at zone level. We use both distance- and travel time-based cost matrices. In order to take the effect of the historical data in our model, we update the cost matrices using the information available in the data.

The remainder of this document is structured as follows. In the next section, we provide a small survey of relevant literature considered for this challenge. The methodology section discusses in detail the approach we used along with the intuition behind our approach. We formalize the algorithm in the form of pseudocode in the same section. In the last section, we present the preliminary results followed by our conclusions and a short discussion.

2. Literature Review

Both TSP and VRP are NP-hard problems that have seen many different integer programming formulations (Orman & Williams (2007)), heuristics and meta-heuristics (Speranza & Archetti (2014); Purkayastha et al. (2020)) in the last decades. Also, the number of VRP variants and applications is increasing. Lin et al. (2014) presented a literature review of green routing problems developed considering economical and environmental aspects. Montoya-Torres et al. (2015) reported a survey on VRPs with multiple depots with variants that include time windows, split delivery, heterogeneous fleet, periodic deliveries, and pickup and delivery. Nowadays, another important trend is the integration of driver behavior into the VRP formulation. In this perspective, Srinivas & Gajanand (2017) study the potential benefits of incorporating driver behavior into the optimization process. They also present a generic version of the objective function for VRP which includes driver behavior. Namely, the cost function is composed of two terms: the expected total cost and the variance of the cost for each arc. The second term is multiplied by a constant that increases or decreases the importance of drives autonomy. Similarly, Canoy & Guns (2019) construct a transition probability matrix using Markov models developed from historical data. The matrix is then used in the objective function of
VRP models.

In this paper, we adopt the Miller-Tucker-Zemlin (MTZ) formulation to solve the TSP, introduced for the first time by Miller et al. (1960). The MTZ formulation has the advantage of being easy to understand and implement, with the number of variables relatively small. As a downside, it has been proven to lead to a weak relaxation, see e.g. Pataki (2003). We emphasize that any other TSP formulation can be adopted and could be investigated as an alternative to the MTZ formulation. We embed driver behavior directly into the cost function like the aforementioned literature. On the other hand, instead of using a probability based matrix as cost matrix we adjust the distance (or travel time) based cost values using historical information. Our approach, therefore, leads to a deterministic optimization.

3. Methodology

This competition concerns the door-to-door delivery of parcels from a delivery service company. This is a complex, yet, crucial part of the supply chain, where better planning of the operation can lead to both driver and customer satisfaction. Given the stop information, i.e., the geographical locations of the parcels to
deliver, the objective is to devise the route the driver will take for the delivery. To this end, a training data set containing 6,112 historical routes is provided. Each of the routes starts at one of 17 stations across the United States of America. For each stop, the data we have at our disposal consists of the GPS coordinates, the (expected) travel time to any other stop within the route, the dimensions of the packages to deliver, and an identifier of the zone, i.e., the geographical planning area the stop falls into, called the "zone id." The zone ids are consistent within the set of routes that share the same station; if the station is different, the same zone id might refer to two different geographical regions. After delivering all parcels, the driver ends the route at the original station again. All routes have one single station and are performed by one single vehicle (uncapacitated routing problem). An example of a route is given in Figure 1, where different colors are associated with different zone ids (the station is left out of the route). The figure exemplifies how routes are divided into zones, and that each zone consists of multiple stops (indicated by numbers).

Our algorithm consists of two parts: the model build phase and the model apply phase. In the first phase, the historical routes are used to embed driver behaviors into the cost matrix. This matrix is then fed as input into the second phase, where a route for a new instance of stops is devised. A general overview of the entire framework is provided in Figure 2.

3.1. Model Build Phase

The first phase, i.e., the model build phase, is described in Algorithm 1. The model we build relies only on historical data corresponding to the same station. The inputs of this stage are the route data, i.e., the geographical information on the stops, referring to a specific station $s$ ($D_s$), and the actually driven routes ($StopSeqs$). Our aim is to convert this actual sequence of stops into a much smaller sequence of zones for each route. This is done using the function ToZoneSeq, where we assume that drivers never go back to an already visited zone. This function performs two main tasks:

1) Assign a zone id to stops for which no information about the zone is available. This issue is resolved by considering the closest stop with a zone id in terms of the Euclidean distances. Haversine distance could also be used as alternative to Euclidean distance; however, as the distances are not such that...
the curvature of the earth comes into play, the difference between the two proposed metrics can be considered as negligible.

2) Convert the sequence of stops into a sequence of zones. Given the order of the stops, for a specific route, and the corresponding sequence of zones, we create a (smaller) sequence of unique zones, which is what we consider to be the order of visited zones for that route. The position of a zone is defined by looking at the longest sequence of stops belonging to the same zone. We clarify the procedure of converting the visited zone sequence into a sequence of unique zones with the following two examples:

\[(Z_C, Z_A, Z_A, Z_B, Z_C, Z_C) \rightarrow (Z_A, Z_B, Z_C),\]
\[(Z_A, Z_C, Z_A, Z_B, Z_B) \rightarrow (Z_A, Z_C, Z_B),\]

where in the first case zone \(C\) is visited last since it presents the longest sequence just after zone \(B\). In the second case the longest sequence of zone \(A\) is one, therefore its position will be given by its first occurrence.

**Algorithm 1** Model Build Phase

**Input:** \(D_s\): route data set corresponding to station \(s\)

**Input:** \(\text{StopSeqs}\): actual sequence of stops for each route in \(D_s\)

**Output:** \(\text{ZoneCenters} \in \mathbb{R}^{m \times 2}\): (estimated) zone centers for each zone in \(D_s\)

**Output:** \(\text{CostMatrix} \in \mathbb{R}^{m \times m}\): costs of traveling between the \(m\) zones

1. \(\text{ZoneSequences} \leftarrow \emptyset\)
2. \(\text{AllZones} \leftarrow \emptyset\)
3. for route in \(D_s\) do
4. \(\text{ZoneSequences[route]} \leftarrow \text{ToZoneSeq(StopSeqs[route])}\)
5. \(\text{AllZones} \leftarrow \text{AllZones} \cup \{\text{ZoneSequences[route]}\}\)
6. end for
7. \(\text{ZoneCenters} \leftarrow \emptyset\)
8. for zone in \(\text{AllZones}\) do
9. \(\text{ZoneCenters[zone]} \leftarrow \text{EstimateZoneCenter(zone, D_s)}\)
10. end for
11. \(\text{DistanceMatrix} \leftarrow \text{ComputeDistMatrix(AllZones, ZoneCenters)}\)
12. \(\text{CountMatrix} \leftarrow \text{ComputeCountMatrix(AllZones, ZoneSequences)}\)
13. \(\text{CostMatrix} \leftarrow \text{ComputeCostMatrix(DistanceMatrix, CountMatrix)}\)

The set \(\text{AllZones}\) consists of all zones that have been visited at least once from station \(s\). Let \(m\) be the total number of zones within this set. The function \(\text{EstimateZoneCenter}\) is used to define the center of each zone as the mean longitude and mean latitude. It is performed considering all the stops belonging to the same zone. The described elements are used in the computation of three matrices: the distance matrix \(D\), the count matrix \(N\), and the cost matrix \(C\). The matrix \(D\) is symmetric and contains the Euclidean distances between all the zone centers in \(\text{AllZones}\). The element \(N_{ij}\) in the asymmetric matrix \(N\) represents the number of times that any driver went from the \(j\)-th to the \(i\)-th zone. Also, in this case the columns and rows are represented by \(\text{AllZones}\). The cost matrix \(C\) is the most important element of our framework. Not
only does it incorporate the distance matrix $D$ as cost of traveling, but also it uses the count matrix $N$ to learn from the historical routes. The cost of traveling from zone $i$ to zone $j$ for $i, j = 1, \ldots, m$ (where $m$ is the number of zones in AllZones) is defined in the following way:

$$C_{ij} = \frac{D_{ij}}{1 + N_{ij}}. \quad (1)$$

The zone centers and the cost matrix $C$ are the output of the model build phase and will be used as input of the model apply phase.

### 3.2. Model Apply Phase

Algorithm 2 describes the model apply phase. For a list of stops to visit, the goal of this second phase is to return a route that best approximates the decisions taken by the driver. The inputs of this phase are the cost matrix and the information about the stops that describe the new instance. The cost matrix, which is an output of the model build phase, might need to be adjusted. This happens when there are stops belonging to new zones, i.e., zones that have never been seen in the train data. This is performed as a preprocessing part, and it follows a three-step routine:

1) If no zone information is available for a particular stop, we assign the closest zone id in terms of the Euclidean distances.

2) The centers of the unvisited zones need to be estimated. We use all stops in the test set that belong to the same zone and we find the mean latitude and longitude.

3) We update the cost matrix using the function UPDATECOSTMATRIX which uses the Euclidean distances between the new zone center and all other zone centers to update the matrix.

**Algorithm 2 Model Apply Phase**

**Input:** CostMatrix: preprocessed output of MODELBUILD  
**Input:** AllZones, AllStops: all zones and stops characterizing the route  
**Input:** station, TravelTimes  
**Output:** PredictedStops: predicted sequence of stops

1. PredictedZones ← TSP(station, AllZones, CostMatrix)
2. PredictedStops ← [station]
3. for zone in AllZones do
4.   PrevStop ← PredictedStops[-1]  \> Last element of PredictedStops
5.   FirstStop ← FINDCLOSESTSTOP(zone, PrevStop)
6.   PredictedStops ← [PredictedStops, OTSP(FirstStop, AllStops[zone], TravelTimes)]
7. end for

The information on the stops is characterized by the list of zones (AllZones) and stops (AllStops), and by the travel times to any other stop (TravelTimes) within the route. Algorithm 2 first converts the information about the stops to a sequence of zones by using the cost matrix. It is performed using a TSP model which takes as input the origin, i.e., the station, the list of zones and the cost matrix. The TSP is solved using the
open source modeler PuLP (Mitchell et al. (2011)) with the Coin-or Branch and Cut (CBC) solver (Lougee (2003)). If the optimization process reaches a predefined time limit (20 seconds), we solve the TSP with OR-Tools, an open source software suite for optimization provided by Google (Perron & Furnon (2019)). The problem is solved using the default settings with a heuristic approach. The output of the TSP is, therefore, a sequence of zones that takes not only the preferences of drivers into account but also the geographical distances between zones.

At this point we have a sequence of zones that has to be transformed into a sequence of stops. In order to do so, we solve an open traveling salesperson problem (OTSP) for each zone. This is accomplished following the order of zones that we got from the previous step. For each OTSP the function \texttt{FindClosestStop} sets the first stop to be the closest (in terms of travel time) to the last stop in the previously visited zone. For the first zone, the last stop is considered to be the station. The TSP is open since we do not need to close the loop connecting the last stop with the first one within the same zone. Also, in this case we use PuLP-CBC for the optimization, with the support of OR-tools if the optimization time limit has been reached (7 seconds). The final output is a sequence of stops, which is the estimated route the driver has taken.

4. Results and Conclusions

During the course of this competition various information regarding 6,112 actual routes were provided for the training part, while the final models will be scored on around 3,050 new routes to check the performance.

In the following subsection, we present the results of our model from the simulations that we had run. We conclude with a discussion about possible future improvements than can be incorporated in our algorithm.

4.1. Results

The organisers have provided us with a validation set of 13 routes to assess the performance of our submissions. Since the number of routes in the test set for the final evaluation is much higher compared to the validation set, the performance of the model based on the validation set may not be a good representation of the actual score based on the test set. To this end, we further split the training set into parts based on different criteria to estimate the performance of our models.

Each predicted route will be scored on sequence deviation (SD) and normalized edit distance with real penalty (ERP). The final score will be the mean of the scores for all the routes in the test set. Assuming \( I = \{1, 2, ..., n\} \) to be the set of routes in the final test set, the route score and submission score can be defined as below:

\[
\text{route\_score}_i = \frac{SD(A, B) \cdot ERP_{norm}(A, B)}{ERP_e(A, B)}, \quad (2)
\]

\[
\text{submission\_score} = \frac{1}{n} \sum_{i=1}^{n} \text{route\_score}_i, \quad (3)
\]
where $A$ and $B$ are the actual and the predicted sequences of stops, respectively.

Initially, we train our algorithm on the entire training set of 6,112 routes and test its performance based on the provided validation set. On a computer with an Intel Core i7-1165G7 (8x2.8 GHz) processor and 32 GB of RAM, the training time is approximately 15 minutes, whereas the validation takes approximately 1 minute. Since the number of routes in the validation set is relatively low, we cannot judge the performance of our algorithm based on the corresponding submission score. Hence, we split our training data to create a validation set of 1,000 routes. Also, in order to check the variability in the route score for the validation set, we decide to look into the standard deviation ($std$) of the route scores. Finally, it is desired to estimate the score in cases where we may encounter both new zones and new stations. Hence, we create a training and a validation data set by randomly sampling 50 routes from the training data. We compare the performance of the above mentioned subcases in the table below.

<table>
<thead>
<tr>
<th>Training Size</th>
<th>Validation Size</th>
<th>Submission Score</th>
<th>std(Route Score)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6,112</td>
<td>13</td>
<td>0.0178</td>
<td>0.0182</td>
</tr>
<tr>
<td>5,112</td>
<td>1,000</td>
<td>0.0581</td>
<td>0.0483</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>0.0684</td>
<td>0.0487</td>
</tr>
</tbody>
</table>

Table 1: Performance comparison for different subcases.

We plot the probability distribution of the route scores for the second subcase, with 1,000 routes in Figure 3. It is observed that the 85.7% route scores lies below 0.1 while 96.5% route scores lies below 0.15.

4.2. Concluding Remarks

The key advantages of our model are its simplicity and its high performance on the test set. The choice of determining the optimal zone sequence before determining the stop sequence not only makes it possible to
incorporate the history into our model in a simple way, but also makes our method computationally easier. Also, since the training time is relatively low, our algorithm can be trained in real time by using the newly realized historical routes of the same day. There are some areas of improvement for future works. The chosen cost matrix is an adapted version of the distance matrix between the estimated zone centers where we currently use the Euclidean distances between the zone centers as the metric. However, it does not take road structure into account. This could be solved by using the provided travel times in the train data. Also, other types of distances may be compared to see if they have an effect on the score. The relation between the distance matrix and count matrix in equation (1) is a heuristic that we proposed. The method we propose uses a linear relationship between the distance and count matrix. Instead, the use of other types of relations, e.g. logarithmic or exponential, could be investigated. By modeling our environment as a Markov decision process (MDP) (Sutton & Barto (2018)), the cost matrix might be learned itself. Also, our solution assumes that the influence of package information can be considered as negligible. We might find a way to infer useful package information that will later affect the optimization process. Finally, we have observed that solving the TSP with other commercial solvers, such as Gurobi (Gurobi Optimization, LLC (2021)), could lead to optimal results in a much smaller time.
References


Article XVI

Tabu Search-Based Two-Stage Stop Sequence Optimization with Zone-Travel Weight Information Learning from Historical Route Data

Zhongyuan Lyu and Xuewu Chen

Source code is available on GitHub:
https://github.com/Xrysnow/LastMileRoutingResearchChallenge
Tabu search based two-stage stop sequence optimization with zone-travel weight information learning from historical route data

Zhongyuan Lyu  
Department of Industrial and Manufacturing Systems Engineering, The University of Hong Kong, Hong Kong, China

Xuewu Chen  
The College of Information Science and Engineering, Northeastern University, Shenyang, 110004, P. R. China

Abstract

In this algorithm, we try to learn a weight matrix from historical data to modify the zone travel matrix of a given instance. Then, we develop a Tabu search-based two-stage stop sequence optimization algorithm to derive the travelling route in a computationally efficient manner. In the first stage, we try to optimize the zone visitation sequence. In the second stage, we employ a rolling horizon strategy to calculate the stop travel sequence in several selected zones based on the tabu search heuristic.

1. Introduction

Based on observation of historical data, we find that, in the associated actual route sequence of each historical instance, the stations belonging to the same zone are usually travelled in succession. Therefore, we try to learn the zone travel sequence from historical data. We have tried several methods, including the optimization-based local search using the PuLP-python library, approximate dynamic programming (ADP) and various heuristics, to generate the stop travel sequence. Through our experiments, the two-stage tabu search based heuristic gives the best scores.

2. Literature Review

Our method relates to the research that learns from historical routing data with machine learning-based approaches. Krumm (2016) tried to learn from the historical GPS data and predict the future road segment. Ye et al. (2015) presented a driving route recommendation system. A prediction module is developed in the system to predict an entire route based on Hidden Markov Models (HMM) and the driver’s history routes. Wang et al. (2015) and Canoy and Guns (2019, September) tried to derive a probability transition matrix based on the first-order Markov model and the driver’s preferred links and routes. However, in the routing challenge problem, the Markov model that assumes the independence of previous travelling status is unsuitable for combining the package information and the
given new traveling distance matrix with the historical route data. Instead of directly deriving the travel stop recommendation, we try to generate a preference weight matrix from historical route data to modify the distance matrix.

We also review the literature related to the two-stage heuristic algorithm. Tuzun et al. (1999) first developed a two-phase tabu search approach for the location routing problem (LRP). In the first phase, they tried to determine the facility location decision. In the second phase, they make the routing decision under the given configuration. The numerical experiments show that their proposed algorithm can achieve better performance than other LRP heuristics. Subsequently, the two-phase heuristic has been widely used to solve various optimization problems (see, for example, Wong et al., 2014, Zhang et al., 2015, Yao et al., 2016). In our algorithm, we also try to optimize the stop travel sequence with a two-stage tabu search-based heuristic.

3. Methodology

In this section, we introduce our proposed algorithm step by step. The flow chart of our algorithm is shown in Figure 1. We generate a zone sequence set based on the historical route data and incorporate it into each instance's travel sequence solving procedure. In step 0, we input all required data to the algorithm. In step 1, given a new instance data, we generate a zone travel matrix representing the travel cost from one zone to another. In step 2, we propose a method to give a modified zone travel matrix based on historical data. Step 3 and Step 4 are the steps for the tabu search-based two-stage stop sequence optimization. In step 3, we use the tabu search heuristic to generate the zone visitation sequence. In step 4, we employ a rolling horizon strategy to calculate the stop travel sequence in several selected zones based on the tabu search heuristic. Then, we introduce the details of each step.
Figure 1. The flow chart of the proposed algorithm.

**Step 0: Input the instance data and historical zone sequence set**

In this step, we generate the zone sequence set. In a historical instance \( s \), let \( z \) represent the zone and \( i \) represent the stop. For each stop \( i \in [1, ..., I] \), let \( k_i \) represent the order in which stop \( i \) appear in the route sequence of the historical instance. Then, we denote the set of stops in each zone by \( s_z, \forall z \in [1, ..., Z] \). For each zone, compute its priority in the instance by

\[
p_z = \frac{\sum_{i\in s_z} k_i}{|s_z|}
\]  

(1)

Then, let \( Seq_z \) represent the list of zones and sort \( Seq_z \) in ascending order of value \( p_z \). Then we have \( Seq_z = \{z_1, z_2, ..., z_Z\} \) and \( p_1 < p_2 < ... < p_Z \).

**Step 1: Generate the zone travel matrix**

In this step, we use the route data to generate the zone travel matrix. For each zone \( z \in Z \), compute the longitude and latitude coordinates of its center point

\[
c_z = (\text{lon}_z, \text{lat}_z) = \left( \frac{\sum_{i\in s_z} \text{lon}_i}{|s_z|}, \frac{\sum_{i\in s_z} \text{lat}_i}{|s_z|} \right)
\]  

(2)

As the coordinates values of zones are very close in the historical data, we use the following equation to calculate the zone travel matrix \( D \)

\[
D_{z_1, z_2} = \begin{cases} +\infty & \text{if } z_1 = z_2 \\ \sqrt{(\text{lon}_{z_1} - \text{lon}_{z_2})^2 + (\text{lat}_{z_1} - \text{lat}_{z_2})^2} & \text{otherwise} \
\end{cases}
\]  

(3)

Here Euclidean distance is not a good approximation to real distance. To further improve the performance, Haversine distance should be deployed

\[
D_{z_1, z_2} = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\text{lat}_{z_1} - \text{lat}_{z_2}}{2}\right) + \cos(\text{lat}_{z_1}) \cos(\text{lat}_{z_2}) \sin^2\left(\frac{\text{lon}_{z_1} - \text{lon}_{z_2}}{2}\right)}\right)
\]  

(4)

Where \( r \) is the radius of the earth.

**Step 2: Calculate the weight matrix and modify the zone travel matrix**

In this step, we calculate history frequency matrix \( N \) corresponding to distance matrix \( D \), where \( N_{z_1, z_2} \) represents the frequency of travel \( z_1 \rightarrow z_2 \) occurs in the zone sequence set. In our proposed implementation, it has the following procedures: Firstly, we collect all history routes \( R \) which has more than two zones of the given new
instance. Let $s_0$ represent the new instance. Secondly, let $t_{s_1,s_2}$ represent the similarity of the given new instance and historical instance. For each $s \in R$, calculate

$$t_{s_1,s_2} = \begin{cases} 
\alpha \cdot \frac{n_c^2}{n_0 n_s} & \text{if } (z_1, z_2) \text{ in } Seq_s, \forall z_1, z_2 \in s_0 \\
0 & \text{otherwise}
\end{cases}$$

(5)

It should be mentioned that when $z_1, z_2$ are not listed next to each other in $Seq_s$, the value is set to be zero. $n_0$ means number of zones of $s_0$. $n_s$ means the number of zones of $s$. $n_c$ represents the number of common zones in both new instance $s_0$ and historical instance $s$. $\alpha$ is a factor depending on the initial station. $\alpha$ is set to 1 when they have the same initial station, and 0.6 otherwise. This parameter setting gives the best scores through the numerical experiment.

Thirdly, calculate history frequency matrix $N$ by

$$N_{z_i,z_j} = \frac{1}{|R|} \sum_{s \in R} t_{s_i,s_j}, \forall z_i, z_j \in r_0$$

(6)

Next, we modify the zone travel matrix with the history frequency matrix $N$. Calculate weight matrix $W$ by

$$W_{z_i,z_j} = F\left(N_{z_i,z_j}\right), \forall z_i, z_j \in r_0$$

(7)

Where function $F$ is a function to be defined in the model. In our proposed implementation, $F$ is in the form

$$F(x) = \begin{cases} 
\alpha \cdot e^{-\frac{x}{b}} & x > 0 \\
1 & x = 0
\end{cases}$$

(8)

By validating on the given data, we set $\alpha$ to 0.5 and $b$ to $8 \times 10^{-4}$. Finally, multiply $D$ with $W$ to get adjusted distance matrix $\tilde{D}$.

**Step 3: Generate the zone visitation sequence**

In step 3, we implement the first stage of the heuristic to optimize the zone sequence $Seq_0$ with the modified zone travel matrix $\tilde{D}$. In our proposed tabu search heuristic, there are the following procedures:

1) Generate an initial path $x_{now}$, set $x_{best} = x_{now}$ and set an empty tabu list $T$. Mark length of path as $L$ and tabu list size as $n_t$.

2) Choose two random integers $m, n \in [1, L]$ to represent two positions in the sequence $x_{now}$. Then inverse subsequence between $m, n$ of $x_{now}$ to generate a neighborhood.

3) Repeat 2) $n_b$ times to get neighborhoods and record corresponding operations $(m, n)$ as $h_i$ into list $H$.

4) Compute the travel distances of neighborhoods using $\tilde{D}$ and sort neighborhoods as $X_n$ in ascending order of their travel distances.

5) For $x_i \in X_n$,
   a) If the travel distance of $x_i$ is less than $x_{best}$, set $x_{best} = x_i$ and $x_{now} = x_i$. Add the corresponding operation $h_i$ into $T$. Break.
b) If \( h_i \) is not in \( T \), set \( x_{\text{new}} = x_i \) and add \( h_i \) into \( T \). Break.

6) If the size of \( T \) is greater than \( n_t \), remove previous elements from \( T \) to make it have the size \( n_t \).

7) Repeat the steps from 2) to 6) for \( n_i \) iterations.

8) Output \( x_{\text{best}} \).

In this step, parameter \( n_t, n_b, n_i \) are set to 5, 400 and 150, which gives the best scores through the numerical experiment.

**Step 4: Generation the stop travel sequence**

In step 4, we implement the second stage of the heuristic with the zone sequence \( \text{Seq}_{s_0} \) derived from step 3. We divide the \( \text{Seq}_{s_0} \) into subsequences and make sure that there are no more than \( C \) associated stations in each subsequence. Then, for each zone subsequence, we use the same tabu search procedures in step 3 to optimize station sequence under the given stop travel matrix from the instance. The last station of the former result will be the start station at each time. Finally, all notations are summarized in Table 1. In this step, parameter \( C, n_t, n_b, n_i \) are set to 15, 5, 400 and 100, which gives the best scores.

**Table 1. Summary of notations**

<table>
<thead>
<tr>
<th>Indices</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s )</td>
<td>index of historical instance, ( s \in [1, \ldots, S] )</td>
</tr>
<tr>
<td>( z )</td>
<td>index of zones, ( z \in [1, \ldots, Z] )</td>
</tr>
<tr>
<td>( i )</td>
<td>index of stops, ( i \in [1, \ldots, I] )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{lon} )</td>
<td>The longitude coordinate</td>
</tr>
<tr>
<td>( \text{lat} )</td>
<td>The latitude coordinate</td>
</tr>
<tr>
<td>( c_z )</td>
<td>The coordinate of zone ( z )</td>
</tr>
<tr>
<td>( D )</td>
<td>The zone travel matrix</td>
</tr>
<tr>
<td>( N )</td>
<td>The history frequency matrix</td>
</tr>
<tr>
<td>( s_0 )</td>
<td>The new given instance</td>
</tr>
<tr>
<td>( t_{x_1,z_2,s} )</td>
<td>The similarity of the given new instance and historical instance</td>
</tr>
<tr>
<td>( n_0 )</td>
<td>The number of zones in ( s_0 )</td>
</tr>
<tr>
<td>( n_s )</td>
<td>The number of zones in historical instance ( s )</td>
</tr>
<tr>
<td>( n_c )</td>
<td>The number of common zones in both new instance ( s_0 ) and historical route ( s )</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>The factor depending on the initial station</td>
</tr>
<tr>
<td>( F )</td>
<td>A function to change the historical frequency value to the weight</td>
</tr>
<tr>
<td>( \tilde{D} )</td>
<td>The modified zone travel matrix ( \tilde{D} )</td>
</tr>
<tr>
<td>( C )</td>
<td>A fixed value</td>
</tr>
</tbody>
</table>

**4. Results and Conclusions**

To evaluate our model, we use 5000 routes of the given data as history routes and 1000 routes as evaluation set without distinctions between routes of different quality. We also evaluated scores with the unmodified zone travel distance matrix \( D \) to make a comparison. The result is shown in Table 2.
Table 2. Numerical results

<table>
<thead>
<tr>
<th>Score</th>
<th>Average</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modified matrix $\tilde{D}$</td>
<td>0.0595</td>
<td>0.0451</td>
</tr>
<tr>
<td>Original matrix $D$</td>
<td>0.0697</td>
<td>0.0462</td>
</tr>
</tbody>
</table>

According to the numerical results, we conclude that information from history routes benefits the model performance. Besides, we observed that the derived score performance is in a long-tailed distribution. Some routes have a poor score in our model. It can be illustrated that we only use zone sequence information from history but ignored time limitations and other information in delivery.

There are several feasible approaches to improve the algorithm: 1) We can incorporate time limitations in the optimization. 2) The weight function in our algorithm can be further improved. 3) The heuristic route optimization algorithm can be refined. 4) To set different weights to routes of different quality. 5) Haversine distance should be used in step 2 to calculate the travel distance more accurately.

References


Article XVII

Inverse Reinforcement Learning for Learning Driver Utility Function for Package Delivery Routing Problems

Jianhan Song, Ashutosh Shukla and Josiah Coad

Source code is available upon request from the authors.
Article XVII

Inverse Reinforcement Learning for Learning Driver Utility Function for Package Delivery Routing Problems

Jianhan Song

Department of Electrical and Computer Engineering, University of Texas at Austin

Ashutosh Shukla

Graduate program in Operations Research and Industrial Engineering, University of Texas at Austin

Josiah Coad

Department of Computer Science, Texas A & M University

Abstract

In this article, we present a framework that Amazon.com, Inc can use to optimize their last mile delivery operations by recommending drivers route for package delivery. Our approach uses past good quality routes taken by drivers as demonstrations to learn the driver’s utility function (which is approximated using a neural network) using inverse reinforcement learning. The utility function’s output are used as parameters to solve a mixed integer programming (MIP) formulation of a soft time window constraint based vehicle routing problem (VRPTW). The mathematical program thus obtained is solved heuristically using an open-source solver (Google OR Tools). Using the proposed approach, we observe near 30 percent improvement in the quality of solutions obtained from our approach when compared to solving a plain vanilla VRPTW.

Acknowledgements

We would like to thank our advisor, Shubhankar Agarwal, Department of Electrical and Computer Engineering, University of Texas at Austin, for his insight and guidance in creating this work.

1. Introduction

In the past two decades, retail industries across the globe have undergone massive digital transformation by creatively bringing in their entire marketplace to small hand-held devices. This has enabled customers

\[1\] In order to determine the quality of solutions, we use the scoring criteria as given in the routing challenge.
to choose from hundreds of thousands of products through simple gestures and get them delivered at any location with just a few clicks. This smooth process which ensures a delightful shopping experience is powered by intelligent algorithms. It is these algorithms that help organizations decide what products to show, when and how so as to carefully manage customer expectations. Furthermore, once an order has been placed, it is again a set of clever algorithms that help organizations decide how should they move the desired product from one of their warehouses located possibly anywhere in the world to the customer so that it is delivered in timely manner and undamaged state. Ensuring this introduces many service level constraints. To meet them, companies have developed sophisticated decision support systems that can leverage their past operations data for demand forecasting, inventory management, transportation planning etc. In addition to these measures which are already in place, organizations have now turned their attention to their last mile operations planning.

One fairly common problem in last mile operations is of routing the sequence of deliveries. Given a set of stops, the capacity of the delivery vehicle and time window constraints on stops if any, the routing problem can be modeled as a capacitated vehicle routing problem with time-windows. The formulation typically reduces to a MIP. This reduced mathematical program thus obtained can then be passed to any state-of-the-art MIP solvers to get final solutions. This approach however has several limitations. Firstly, such mathematical formulations usually place deterministic assumptions on travel times between nodes and service time at stops. One way to alleviate this can be to solve a stochastic version of the problem with distributional assumptions on travel and service times. Such an approach brings in new sets of challenges in terms of how should one infer the probability distribution over travel and service times. A middle ground to approach such problems could be based on distributionally robust formulations by imposing restrictions on first and second moments of the distributions but such approaches usually lead to very complex formulations. Besides, even the deterministic version of the problem falls within the class of NP-hard problems and depending upon an instance, the solve time can become unmanageable for optimization over large scale networks. Another major setback with such mathematical programming based formulations is that it assumes that driver’s utility function that he is trying to maximize is consistent with the objective function of the formulation which may not be justifiable. Also it implicitly assumes that all drivers have same utility function for all types of deliveries which may not be true in real life.

To overcome the challenges of computational intractability and a better representation of driver’s utility function, we propose an integrated learning and optimization based framework. In our proposed approach, we first model the driver’s utility function using an inverse reinforcement learning based approach. Here we use the past high-quality routes (routes taken by drivers who did well on some success criteria as measured by the organization) to demonstrate driver’s policy. Taking these routes as demonstrations, we try to learn

---

2The journey of ordered products from source to destination usually has several intermediate stops. The set of decisions that are made to move the products from it’s last stop to customers is referred to as last mile planning.
the reward function that the driver is trying to optimize. This approach of ours is based on the premise that, drivers usually being from the city, have a deep level knowledge on traffic conditions, idea on where they could find parking easily, where can they take that unexpected short cut etc. Once we have learnt the reward function that maps the given stop inputs (like travel times, service times, number of packages to deliver at a particular stop, size of packages etc.) to some the utility function that the driver is trying to optimize, we apply this function over any fresh input data. The transformed inputs are later fed to a heuristic based routing solver, which solves the to provide high-quality solutions in no times. The details pertaining to same are discussed in the methods section.

2. Related Work

The VRPTW has been quite popular in academic communities since past few decades. Santillán et al. (2012) have discussed it’s various applications in inventory and logistics management. Similarly Niroomand et al. (2014) used it to solve a network design problem for a postal service company. The model has also been used for allocation of resources in electric power distribution systems by Agudelo Zapata et al. (2019). Being an NP-hard problem, the focus of vast majority of literature has been on exploiting structural properties of the problem and developing approximation algorithms to solve the problem in a heuristic manner. As a consequence of such research efforts, we have zeroed down to a set of heuristics that are applied to solve new applications of the problem.

However, there is no reason to believe that one heuristic can generalize to all problems. Rather the problem can be modelled such that for any new application we can learn the heuristic for a specific class of problems by leveraging past data. This has motivated some recent efforts from reinforcement learning community. For example, Neural Combinatorial Optimization, by Bello et al. (2016), trains a pointer network with a recurrent neural network (RNN) encoder to predict a distribution over permutations of routes. The authors use the total route length as the cost signal in the reinforcement learning algorithm. They achieve near optimal results on routes up to 100 nodes. Another similar approach by Nazari et al. (2018) uses reinforcement learning to find a stochastic policy to solve the vehicle routing problem. The trained model produces a solution as a sequence of consecutive actions. Their model is a simplified version of a pointer network along with an attention layer. In contrast to Neural Combinatorial Optimization, they omit the RNN as an encoder since they note that there is no meaningful information in the order of the input information about all the stops. They are able to solve routes sized up to 100 nodes with near optimal solutions and comparable computation time to google’s OR-tools. In yet another attempt by Kool et al. (2019), an attention based model is introduced, trained with the REINFORCE algorithm. The authors achieve comparable results to highly optimized and specialized algorithms. They get close to optimal results for problems up to 100 node.

The underlying theme of approaches discussed above is common. They aim to come up with policies which are competent or better than traditional heuristics which are common in literature. However, these
approaches are plagued by large computational overheads first for training the model using millions of instances and then sampling hundreds of thousands of routes, evaluating each of them and choosing best amongst them. Furthermore, these approaches are applied to simplest version of VRPTW, a TSP which consists of just one driver and no time window constraint. The performance of this approach when benchmarked against google or tools as compared in Bello et al. (2016) is comparable but like mentioned comes with large computational burden. Lastly, the paper does not mention any details on how the approach does with larger networks or for problems with more complicated constraints.

Our approach, makes an attempt to integrate learning and optimization but in a way slightly different that what has been the goal of above discussed work. We try to use traditional heuristic techniques developed in operations research literature for the optimization part. But in order to determine the parameters which go as input to optimization framework, we rely on learning utility functions (which reflect objective function) using techniques developed in reinforcement learning (particularly inverse reinforcement learning). A creative integration of the two has helped has get the best of both worlds. By learning the driver’s utility function, we have achieved over 30 percent improvements as compared to solving plain vanilla VRPTW.

3. Methodology

3.1. Problem Formulation and Notations

The training dataset consists of last mile logistics information for nearly six thousand real-world Amazon delivery tasks. Let $\mathcal{T}$ denote the set of delivery tasks in the training set. Each task $t \in \mathcal{T}$ is associated with a set of $n(t)$ stops, denoted by $\mathcal{V}^{(t)} = \{v_1^{(t)}, \ldots, v_{n(t)}^{(t)}\}$. Without loss of generality, let $v_1^{(t)}$ be the depot and other stops denote the package drop-off locations. The logistic data provided for each task includes geographic information on stops and information of packages that need to be delivered to each stop. Please refer to the specification of the dataset for full details. By proper pre-processing, the data associated with a task $t$ can be categorized into three types:

1) Task features, denoted by vector $x^{(t)}$: features defined upon the whole task $t$, e.g., the depot location, the total number of stops, the type of the delivery vehicle, etc.

2) Stop (vertex) features, denoted by vector $y_i^{(t)}$ for each stop $v_i^{(t)} \in \mathcal{V}^{(t)}$: features associated with each stop, including the number of packages, the geographic location, time window, etc. To be precise, the time window of a stop $v_i^{(t)}$ is defined by the overlapping of windows of all the packages it associates with, denoted by the interval $(w_{i,1}^{(t)}, w_{i,2}^{(t)})$. Let $W^{(t)} := (w_{i,1}^{(t)}, w_{i,2}^{(t)})_{i=1,2,\ldots,|\mathcal{V}^{(t)}|}$ be the list of time windows.

3) Link (edge) features, denoted by vector $z_{i,j}^{(t)}$ for each directed link $(v_i^{(t)}, v_j^{(t)}) \in \mathcal{V}^{(t)} \times \mathcal{V}^{(t)}$: features associated with each edge, e.g., the (estimated) travelling time from $v_i^{(t)}$ to $v_j^{(t)}$. We denote the travelling time of link $(v_i^{(t)}, v_j^{(t)})$ as $d_{i,j}^{(t)}$. Let $D^{(t)} := (d_{i,j}^{(t)})_{i,j \in \{1,2,\ldots,|\mathcal{V}^{(t)}|\}}$ be the travelling time matrix.

Let $I^{(t)} := (x^{(t)}, (y_i^{(t)})_{i \in \{1,2,\ldots,|\mathcal{V}^{(t)}|\}}, (z_{i,j}^{(t)})_{i,j \in \{1,2,\ldots,|\mathcal{V}^{(t)}|\}})$ be the overall data associated with task $t$. Aside from the above features, in the training set, the actual route sequences generated by Amazon drivers
are provided as “labels”. We denote the actual sequence of task $t$ by a binary matrix $S^{(t)}$, such that $S^{(t)} = (s^{(t)}_{i,j})_{i,j \in \{1,2,\ldots,|V^{(t)}|\}}$ where $s^{(t)}_{i,j} = 1$ if and only if $(v^{(t)}_i, v^{(t)}_j)$ is in the actual sequence.

Note that the dataset contains tasks finished by drivers with different experience level (High, Medium and Low). Let $T = T_H \cup T_M \cup T_L$, with each subset containing tasks finished by High, Medium and Low drivers respectively.

3.2. IRL Approach

The underlying intuition of our main approach can be summarized as follows: We believe a driver’s goal is to minimize his effort/cost when designing the route sequence. However, how an experienced driver “measures” the cost driving between any two stops is not obvious (e.g., simply using the travelling distance as a surrogate). For instance, drivers might prefer serving customers block by block, thus travelling across blocks should indeed induce additional cost even if two stops (of separate blocks) are close geographically. Therefore, the focus of our approach is to learn the implicit cost, or the abstract distance as we call it, of each link based on all the available features, and then the problem can be modeled as a TSP instance.

Furthermore, since meeting the time constraints of packages is an important factor (but not a strict requirement) affecting route decisions of a delivery task, we believe drivers do respect the time windows to some extent. Thus, instead of a vanilla TSP model, we assume drivers indeed try to solve TSP instances with soft time window constraints.

The learning framework is exhibited in Figure 1. In the forward pass, for each task we first use a neural network $f(\cdot)$ to approximate the abstract distance matrix from the available data $I^{(t)}$. The design of the neural network will be discussed later. The generated abstract distance matrix $F^{(t)} := f(I^{(t)})$, and

![Figure 1: Prediction Framework.](image-url)
a trainable soft-time-constraint coefficient $\lambda$ (along with the travelling time matrix $D^{(t)}$ and time window constraints $W^{(t)}$), is given to the TSP solver. The solver generates a predicted sequence $S_{f,\lambda}^{(t)}$ such that $S_{f,\lambda}^{(t)} = \arg\min_S \sum_{(i,j):S_{i,j}=1} F_{i,j}^{(t)} + \sum_{v \in V^{(t)}} \lambda \Delta_v^{(t)}(S)$, where $\Delta_v^{(t)}(S)$ is the time violation of stop $v$ given sequence $S$, travelling time matrix $D^{(t)}$ and time windows $W^{(t)}$.

The loss function is defined as follows

$$L(f, \lambda) = \sum_{t \in T_H} L_{f,\lambda}^{(t)} + \beta_1 \sum_{t \in T_M} L_{f,\lambda}^{(t)} + \beta_2 \sum_{t \in T_L} L_{f,\lambda}^{(t)}$$

where

$$L_{f,\lambda}^{(t)} = \log \frac{C_{f,\lambda}^{(t)}(S^{(t)})}{C_{f,\lambda}^{(t)}(S_{f,\lambda}^{(t)})}, \quad C_{f,\lambda}^{(t)}(S) := \sum_{(i,j):S_{i,j}=1} F_{i,j}^{(t)} + \sum_{v \in V^{(t)}} \lambda \Delta_v^{(t)}(S),$$

and $\beta_1 \leq 0, \beta_2 \leq 0$ are hyper-parameters. To interpret this loss function design, first consider $\beta_1 = \beta_2 = 0$, i.e., suppose we only use tasks in $T_H$ for training. For any $t \in T_H$, we wish that the total cost of the predicted sequence is close to that of the actual sequence (i.e., $C_{f,\lambda}^{(t)}(S^{(t)}) \approx C_{f,\lambda}^{(t)}(S_{f,\lambda}^{(t)})$), and it is reasonable to assume that the closeness of total costs implies closeness of sequences themselves. Note that since $S_{f,\lambda}^{(t)}$ is ideally the optimal solution for the VRPTW instance defined by $(F^{(t)}, \lambda, D^{(t)}, W^{(t)})$, we have $C_{f,\lambda}^{(t)}(S^{(t)}) \geq C_{f,\lambda}^{(t)}(S_{f,\lambda}^{(t)})$, and thus $L_{f,\lambda}^{(t)}$ is used to measure their similarity. When $T_M$ and $T_L$ are considered, we set $\beta_1$ and $\beta_2$ to be negative since we want to avoid mimicking behaviors of inexperienced drivers.

In the back propagation, a major issue is that the TSP solver is non-differentiable. Thereby, we fix the predicted sequence $S_{f,\lambda}^{(t)}$ once the forward pass is done, and then the gradient update can be applied to the neural network $f$ and $\lambda$ with respect to the loss function.

**Design of the Neural Network:** To avoid large networks which demand high computation cost, we indeed do not build a network with $I^{(t)}$ as input and outputting $F^{(t)}$ directly. Instead, we assume the abstract distance of each link is only determined by its local information (through some common transformation), and a small network can be used to model this relation. To be specific, for each link $(v_i^{(t)}, v_j^{(t)})$, we concatenate $\tilde{I}_{i,j} := (x_i^{(t)}, y_i^{(t)}, y_j^{(t)}, z_{i,j}^{(t)})$ as the input to the network, which outputs $\tilde{F}_{i,j}^{(t)}$ as the corresponding predicted abstract distance. The matrix $F^{(t)}$ is constructed as $(\tilde{F}_{i,j}^{(t)})_{i,j}$.

### 3.3. TSP Solver

Once the inputs are transformed to abstract distance matrix, we pass this as our distance matrix with soft time window constraints. The penalty to impose when the constraints are violated is a parameter that we have learnt using the training data. The mathematical program this obtained is a NP-Hard problem and is approximately solved using a heuristic based optimization solver Google OR Tools [Perron & Furnon, 2019]. The solver takes as input the distance matrix between the nodes, the time window at each node. The model also expects us the specify a first solution strategy heuristic (used to generate an initial feasible solution) and a metaheuristic that affects how new solutions are generated from already existing set of feasible solutions. In this study, we have used path cheapest arc heuristic for first solution strategy with default metaheuristic.
4. Experimental Setup and Results

In this section, we describe the experimental setups that we use for testing our approaches. We mainly relied on AMZ score function provided by the organizers to validate our approaches. AMZ Score function was treated as a black box which provided a continuous metric of how different two routes are. Exactly same routes (following the stops in exactly same order as the actual sequence) were given a score of 0.0.

For our approaches we used the features provided in the dataset. As explained Section 3.1 divided our features into link and route feature sets.

- **Task Features**, denoted by vector $x^{(t)}$: These features represented data for the whole route and were same for all the links. We had in total 6 route features. Specifically the route features we used are
  - vehicle capacity: The capacity of the vehicle carrying packages.
  - station code: One hot encoding representing the city the route belonged to.
  - Package features - Largest package and avg package volume of the route.

- **Stop Features**, denoted by vector $y^{(t)}_i$: These features represented data for a stop $i$. We had in total 3 stop features. Specifically the stop features we used are
  - number of packages, service time at the stop and one hot encoding of the Zone.

- **Link Features**, denoted by vector $z^{(t)}_{i,j}$. These features represented data between a stop $i$ and a stop $j$. We had in total 7 link features. Specifically the link features we used are
  - travel times: travelling time between stop $i$ and stop $j$.
  - zone crossing: 1 if the link between stop $i$ and stop $j$ is crossing a zone and 0 otherwise.
  - geographic distance: Distance between geographic locations (lat, long) of the stops.

4.1. Neural Network Training Setup

For the neural network based approach we had used 3 fully connected layers with Relu functions at the end of each layer. Experiments showed that bigger networks did not help in our approach. We performed standard data normalization techniques on our dataset. Additionally, we used a replay buffer to stabilize our training. Replay Buffer was 5 times the size of the dataset. Batch size 256 was used for training. Our model is trained for 5-10 epochs. Figure 2 shows the training loss and score plots for both linear and NN models. NN model is able to achieve lower loss and therefore a better score. It is interesting to observe that even though the loss is decreasing for NN model, the score does not get better. This behavior was the result of our loss function, which encourages the NN to increase the norm of the output to decrease the loss.
Table 1: Comparison of different approaches on 1000 test routes. Lower score is better.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Avg Score of Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR Tools</td>
<td>0.105</td>
</tr>
<tr>
<td>Linear</td>
<td>0.0852</td>
</tr>
<tr>
<td>NN</td>
<td><strong>0.0592</strong></td>
</tr>
</tbody>
</table>

Figure 2: Training Plots. **Left** figure shows the loss of linear model and NN model for each epoch. **Right** figure shows the score of linear model and NN model for each epoch.

4.2. Results

We trained our approaches on 2000 routes and tested on 1000 routes. Table 1 clearly shows that the NN model outperforms the OR Tools and the linear model, achieving significantly lower score on the test set. OR Tools score is calculated with just using travel times as abstract distance.
Figure 3: Comparison between an actual route and route predicted by our NN approach. The number of each pin denote the order of the stop in the route and color represents the zone.
4.3. Qualitative Results

The Figure 3 shows that the NN predicts similar sequence within each zone, but is unable to accurately predict the sequence of the zones in the route. This is due to the fact that our model only has information of the specific link when predicting abstract utility cost. NN Model’s output are only locally consistent therefore the model is not able to learn the sequence of zones. We discuss some ways to tackle this in the Section 5.

5. Conclusion

We presented a very simple yet powerful approach for learning the Amazon driver utility function. Our model has less than 50 learnable parameters and still is able to perform very robustly. Our model is quickly trained in 5-10 epochs. Additionally, our model is easily parallelizable and is able to predict a route within 2-3 secs with significant (90%) amount of time taken by the Google OR Tools solver. Finally, with the help of Google OR tools solver and learning the time constraints penalty term we are able to make sure we do not violate any delivery time constraints.

Even though our model performs strongly as it is, there are many further improvements we can introduce to further improve the performance. One of the key weaknesses of our approach is that our NN model does not have any information about surrounding stops. Our NN model predicts the abstract utility cost of a link only by looking at the features of that specific link. This is a major flaw for a problem which is highly correlated between links. We can address this flaw two ways:

- We can provide $n$ neighbor link features to the model, instead of just providing features of the specific link. This way our model will be able predict abstract utility cost for a link by also accounting for neighboring links.
- We can also use a separate network to learn soft time constraints for each stop in the route and provide those to OR Tools solver, instead of providing raw time constraints. This network will be provided features of all the links will be able to learn the global structure of the route, i.e. Zone A comes before Zone B.

6. Acknowledgements

We would like to thank Professor Guni Sharon (Department of Computer Science, Texas A & M University) and Professor Ngoc Tran (Department of Mathematics, University of Texas at Austin) for their highly valuable advice, suggestions and mentorship. Additionally, we would like to thank Amazon Routing Challenge for providing data and platform for us to tackle the challenging problem. Lastly, we acknowledge VITA Lab’s (Department of Electrical and Computer Engineering, UT Austin) support for providing us with needed computing resources.
References


Article XVIII

A Perspective on Driver’s Preferences for Route Planning

Shanshan Huang, Rayner Rebello and Qiuyu Zhu

Source code is available on GitHub:
https://github.com/r-rb/routingchallenge/tree/main
Abstract

In order to model the choice of which stop a driver will go to next, we use a discrete choice model that augments a Multi-nomial Logit model with a neural network when computing the utility of a particular stop. We are able to achieve an accuracy of 77% when predicting the next stop in a route. When applied to generating new route we achieve an average score of 0.0627 over all the provided routes.

1. Introduction

Our team’s approach to modelling the problem is to split the decision of choosing a route for a driver into a series of sequential decision problems. We model the driver as an agent who picks one stop at a time from a set of stops that are left to go. To model the driver’s decision-making for picking which stop to go to next, we use a discrete choice model. To start, we use a Multi-nomial Logit model which uses a linear function to prescribe a score or utility for every available stop. This does perform relatively well, but we improve the accuracy slightly by augmenting the linear utility function with a neural network. This joint choice model manages to pick the correct stop 77% of the time with local testing.

The data given to us was leveraged by first turning the observed routes that drivers took into input data for our discrete choice model. To do this, we treat an observed route as many observations of a driver choosing a stop to travel to next, given whatever stops that are left. For example, a route that has 100 stops
provides 99 observations of where a driver has to pick between multiple stops. Each stop in each observation is a possible choice for the driver to go to, and the actual stop the driver picks is the target (dependent) variable. For each option, we extract features that may explain why a driver would choose to go that stop and input it into the discrete choice model we fit.

2. Literature Review

Discrete Choice Model (DCM) is widely used to model choices by an agent (e.g. decision-maker) over a set of alternatives (choice set). Each alternative gives the agent a different utility which consists of a representative utility modelled by the researcher and an unknown random utility. Faced with a set of alternatives, the agent chooses the one with the highest utility. The probability of choosing one alternative will be the probability that its utility is the highest. Among the class of DCM, one of the most widely used models is the Multinomial Logit Model (MNL) which we use to model a driver’s choices over feasible stops in a network. For more details about the DCM and MNL, please refer to [Train (2009)].

Many studies have applied DCM in transportation problems. There is a stream of papers using DCM to model a driver’s choices of paths in the route choice problem (RCP) ([Zimmermann & Frejinger (2020), Broach et al. (2012), Bekhor et al. (2006)]). Their choice set is a set of feasible paths. Since there is an exponential number of possible paths in a network, they consider a subset of feasible paths which is generated between the origin and destination of each observation of path by solving versions of the shortest path problem ([Bekhor et al. (2006)]). However, it is impossible to consider all feasible paths in our problem and our destination is not determined in the network. Therefore, algorithms for RCP are not applicable to our problem. Since the goal is to come up with a sequence of stops, we propose to use DCM to model choices of a driver over feasible stops in the network where drivers are assumed to be homogeneous.

Moreover, our problem differs from the Travelling Salesman Problem (TSP) ([Flood (1956)]) in which the objective is to minimise the cost of distance. However, the objective is not clear in our problem which should be learned from the data. One may propose to use Inverse Reinforcement Learning (IRL) where the reward function is also learned from the data ([Ravichandiran (2018)]). However, our method is much easier to implement.

3. Methodology

Our team chose to split the problem of choosing a route given a set of stops into a sequence of choices of stops the driver should visit. The choice of stop is modelled with choice models where every alternative is assigned a utility or score. We assume that a delivery person first considers the set of stops that are left to be completed and picks one which provides the largest utility. After every stop, the delivery person
repeats the procedure without the stop they are currently at until the route is completed. So every route in our dataset will provide several observations to our choice model. For the purposes of our description of the choice model, route and delivery person are interchangeable, as we have no information specific to the driver.

3.1. Multi-nomial Logit Model

We choose to use a multi-nomial logit (MNL) model to understand a delivery person’s decision of which stop to go to next. The MNL model has a closed form for the choice probabilities and is convenient to implement into Keras as the soft-max layer is already written. Other DCM such as the probit model or the nested MNL model would require us to estimate a covariance matrix or group the stops into different hierarchies.

Let $S_i$ denote the set of stops that need to be visited in route $i$. Let $S^n_i$ denote the set of stops that need to be visited in route $i$ where there are $n$ stops already sequenced. Let $X^{ni}(s)$ be a vector of features that are induced from information about a stop $s \in S^n_i$ and also route level information about the route $i$. When a delivery person on route $i$ picks a stop they consider a utility of picking stop $s$ to go to next:

$$U(s) = \theta^T X^{ni}(s) + \epsilon, \quad \forall s \in S^n_i$$

where $\epsilon \sim EV_1(0, 1)$, a standard type-1 extreme value distribution and $\theta$ is the parameters we wish to learn. Let $y^{(s)} = 1$ if $s \in S^n_i$ is the $(n + 1)$-th scheduled stop and 0 otherwise. The prediction probability of a stop $s \in S^n_i$ is:

$$\Pr[y^{(s)} = 1] = \frac{\exp(U(s))}{\sum_{s' \in S^n_i} \exp(U(s'))} \quad (1)$$

3.2. Neural Network Augmentation

The MNL model uses the linear utility function which may not be sufficient to capture the possible dependencies between the features of the stop which may be non-linear in reality. We propose to use Neural Network to capture the non-linear relation between the features and the stops utilities. The utility of the stops in DCM will be modeled as the linear utility $\theta^T X^{ni}(s)$ added by the non-linear utility given by the Neural Network. Let $V$ be a function denoting the output of a multilayered neural network which takes in the features of a stop and returns an utility. The joint utility can be written as:

$$U(s) = \theta^T X^{ni}(s) + V(X^{ni}(s)) + \epsilon_i, \quad \forall s \in S^n_i \quad (2)$$

After obtaining the utilities $U(s)$ from (2), we can obtain the choice probabilities by Logit formula (1). Then we can calibrate this model using historical choices of stops by drivers. Specifically, the parameters of Linear utility model $\theta$ and the Neural Network are jointly estimated.
3.3. Features

From each route in the dataset, we are given multiple observations in which a driver sees a list of stops that are yet to be made, and then picks one. We chose to generate 16 features about every stop considered in each of the observations. The features can be split into three broad categories. The first is features that describe the characteristics of the stop available, such as the distance from the current stop the driver is at. The second category is forward-looking features which are based on the characteristics of the available stops if the considered stop is chosen. The third category is the network features, such as PageRank and closeness. We use the networkx to calculate the network features. Here is a list of all the features we used to train our model.

- Characteristics of the stop:
  1. \textit{dist}: The travel time from the stop the driver is currently at to the stop in seconds.
  2. \textit{service time}: The service time required at the stop in seconds.
  3. \textit{same zone}: Indicator variable if the stop has the same exact zone id as the current stop.
  4. \textit{same outer zone}: Indicator variable if the stop has the same outer zone as the current stop. e.g "A-2.1A" and "A-3.2B" have the same outer zone.
  5. \textit{second zone diff}: Absolute difference of numerical values in the zone ids of the drivers current and considered stop. "A-2.1A" and "A-3.2B" have a difference of 1.1.

- Forward-looking features:
  1. \textit{avg dist to closest stops}: Average distance to closest 5 stops, if the considered stop would be chosen, in seconds.
  2. \textit{avg dist to other stops}: Average distance to all other remaining stops, if the considered stop would be chosen, in seconds.
  3. \textit{avg dist to same zone}: Average distance to all other remaining stops with the exact same zone id as the considered, if the considered stop would be chosen, in seconds.
  4. \textit{min dist to other stops}: Distance to closest stop, if the considered stop would be chosen, in seconds.
  5. \textit{avg closest diff zone}: Average distance to stops in the closest different zone as the considered stop, if the considered stop would be chosen, in seconds.
  6. \textit{min closest diff zone}, Minimum distance to stops in the closest different zone as the considered stop, if the considered stop would be chosen, in seconds.
  7. \textit{sum closest diff zone}: Sum of all the distances to stops in the closest different zone as the considered stop, if the considered stop would be chosen, in seconds.

- Network features:
Figure 1: Diagram showing how the utility for all available stops are calculated. The input is a row of the feature matrix $X(s)$ associated with stop $s$ in the set of stops that are yet to be sequenced. The output is the utility of that stop $s$. The model is applied to every row of the feature matrix giving a utility vector.

1. pagerank: Pagerank of the considered stop.
2. betweenness: Normalized betweenness centrality of the considered stop.
3. eigenvector: Normalized eigenvector centrality of the considered stop.
4. closeness: Normalized closeness centrality of the considered stop.

3.4. Implementation and Training

There are a total of 6112 routes, of which 2718 are high quality. We choose to only generate training data using high-quality routes. For each of these routes, we treat every choice of stop as observation into the choice model resulting in approximately 370,000 training examples. Each training example creates a feature matrix of $238 \times 16$, where 238 is the maximum number of stops in any single route and 16 is the number of features we extract from a stop as detailed in the previous subsection. This provides an upper bound of the number of choices in the choice model when being fit. Many of the observations will correspond to when there are much fewer than 238 stops that the delivery person has yet to visit. To handle this, we fill in "dummy" stops which are just stops with features of a randomly chosen "real" stop. These dummy stops are added to the end, and as such can never be picked. The softmax layer we use will predict the next stop based on which has the highest utility and in the case of a tie chooses the stop associated with the smaller index. We apply this to generate new routes by using the trained neural network and coefficients of the linear model to compute the utility of any stop, thereby generalising to routes with any number of stops.

The implementation of the MNL and the neural network is in TensorFlow. The MNL where the utility function is linear has 16 parameters and is implemented as a custom Keras layer. This linear model computes the utility of every one of the 238 options by computing the dot product with the parameters and the features.
of that stop. The output is then put into a softmax layer. The loss function was set to cross-categorical cross-entropy. The linear model is trained with 90% of the training data and 10% left as validation. The parameters which achieved the best validation classification accuracy are picked.

The linear model is trained first, and then is set to be untrainable - we want to use the implied utility as a starting value to train our neural network. The neural network loosely speaking is correcting the utility of the linear model. We do this by creating another model which uses the pre-trained linear model and also a custom Keras where the utility of each option is computed with a neural network instead. This neural network has an input layer of size 16 with 4 hidden layers of size 12 with ReLU activations all densely connected. In total there are 913 trainable parameters. The utility vector of the neural is added to the utility vector generated by the linear layer. The combined utility vector is then fed through a softmax layer. We train this joint model with 90% of the training data and 10% left as validation. The parameters which achieved the best validation classification accuracy is picked.

We only need to save the linear model parameters and the neural network inside the custom layers to be used when generating the actual routes. To generate a new route, we generate features for all the remaining stops and compute the utility for each with the saved models. The figure below shows that the features for each stop left are put into the joint choice model which returns a combined utility.

The stop with the highest combined utility is set as the next stop in the sequence. This process is

![Histogram of scores for all routes](image)

*Figure 2: Histogram of scores when testing our trained choice model over all 6112 routes provided in the dataset.*
repeated until all the stops are sequenced.

4. Results and Conclusions

The performance of the choice models we train is evaluated by generating new routes for all the routes in the data given to us. The trained joint choice model achieves a validation accuracy of 77% and when used to generate proposed routes, achieves an average score of 0.0627. The maximum and minimum score is 0.43 and 0.003 respectively. Figure 2 on page 6 shows a histogram of the scores. The average score on only "High" quality routes is 0.0611.

When our method scores poorly, this is often due to picking the wrong first stop. Once picked, it usually decided on a similar sequence to that in the actual sequence. It means that the model is able to capture the local behaviour of drivers effectively. However, a mistake in the first stop creates a relatively large deviation in the score because the station is far away, despite the subsequent stops in the proposed route matching the actual one.

A possible avenue for improvement would be introducing heterogeneity to the model so that stop choice can be parameterized with more information about the route. For example, drivers may trade off features differently near the beginning of their route than towards the end. Other factors such as if the departure time is during peak hours or if the city is urban/suburban could influence the utility drivers perceive a stop to have. This model heterogeneity can be implemented by training different models by partitioning the dataset or adding another layer that takes in route/drive level features to contextualize the utility.

References


Article XIX

Simple Heuristic Algorithm to Solve the TSP With Spatial Structure and Time Window Adherence

Ha Dao, Konstantin Geier and Bernhard Hilfer

Source code is available on GitHub:
https://github.com/hadoop211/routing-challenge
Article XIX

Simple heuristic algorithm to solve TSP with spatial structure and time window adherence

Ha Dao, Konstantin Geier, Bernhard Hilfer
*TUM School of Management, Technical University of Munich*
80333 Munich, Germany

Abstract

For the routing challenge, we propose a two-step heuristic algorithm which builds a meta-tour of zones on the high level and an intra-zone tour of stops as the final route. In each step, we solve a multiple-objective problem, i.e., distance optimization and zonal structure adherence for the meta-tour and distance optimization and time window adherence for the intra-zone tour. With this approach, we contribute a method to elevate traditional basic (heuristic) algorithms (Nearest Neighbor, 2-Opt, Iterated Local Search, Farthest Insertion) to deal with a Traveling Salesman Problem (TSP) in a practical scenario. This algorithm obtains the score of 0.05101 on 2718 high-score routes in the training set with a run time of less than 30 minutes.\(^1\)

1. Introduction

While the traditional TSP mainly focuses on optimizing traveling distance or time (hereafter, both mentioned as distance), in this challenge, we attempt to mimic the decisions made by drivers without prior knowledge of how they do so, i.e., which criteria they use. Therefore, while building the algorithm to solve this challenge, we pay attention to the following points:

- The decision is made by humans. As a consequence, we want the criteria or rules behind our algorithms to be easily understandable.
- A human should be able to apply the whole decision process in a quick and easy manner.

We decompose the problem into smaller sub-problems. Firstly, we define a high-level meta-tour of zones and then define the tour in each zone as well as connections between zones. In more detail, for the meta-tour, our approach is to integrate spatial structure information into traditional distance optimization methods.

\(^1\)Running multi-processing on a computer with 3.47GHz x 4 cores
Specifically, we obtain the spatial structure by utilizing the zone id data field provided for each stop. Because these zone IDs are defined by Amazon, they should be constructed in a way which is meaningful and specific to the company’s operations. Moreover, in Amazon’s operation process, drivers are given zone codes instead of the explicit addresses in the manifest (Blank (2021)), hence it is reasonable to assume that drivers are familiar with these codes. For the tour within each zone, we combine time window information with distance in the objective function. In short, the whole process can be described as a two-step heuristic algorithm:

- **Step 1**: Build the meta-tour of zones using Nearest Neighbor (NN) and iterative 2-Opt with a multi-criteria objective function considering distance optimization and spatial structure adherence.
- **Step 2**: Build intra-zone tour using Farthest Insertion (FI) with a multi-criteria objective function considering distance optimization and time window adherence.

2. Literature Review

The literature about solving the TSP can be roughly partitioned into three approaches:

1. Exact Modeling and Optimization
2. Meta-Heuristics
3. Artificial Intelligence and Machine Learning

For the first approach there exist different model formulations, however, they all suffer from their NP-hardness and cannot be solved efficiently within a short time even though the computational power has shown a huge increase in the past (Schrijver (2005)). As one of the premises of the Amazon routing challenge is to be able to solve many routes within a short time, we avoid exact modelling approaches and try to focus on the other fields. Moreover, the driver has a major impact on the route sequence as he may use his knowledge of the environment. However, we believe that the observational and computational capabilities of the drivers are limited as it is hardly possible to grasp all the information about possible traffic jams at different times, roads that are hard to navigate or other factors like time window adherence. We therefore believe that an approach that mostly utilizes heuristics with parameters that we learn from the data is the most suitable for this problem as it comes with many advantages. Most of the heuristics are proven in literature to have a very good computation speed to performance ratio compared to approaches like exact modeling. Additionally, they are easy to understand and often follow a comprehensible approach that makes sense to a human like visiting the nearest neighbor of the last node. Following this idea, we base our approach on the concept that the drivers are using easy decision rules in the form of heuristics for their routing choices. Our goal with this approach is to mimic the decision that the drivers do as best as possible and check whether these heuristics somewhat emulate the thought process of a human. In the following we name some of the most known methods that solve the TSP (Cook et al. (2007)).

1. Constructing heuristic:
2. Exchange / Improvement methods:

(a) K-Opt
(b) Genetic Algorithm
(c) Simulated Annealing
(d) Tabu-Search
(e) A-Star search
(f) Ant-Colony-Optimization

In our approach we combine Nearest Neighbor, Farthest Insertion and 2-Opt. Although the other methods can theoretically yield better results, they also come with higher computation times which is a very limiting factor. Also, these methods often conduct changes that are not comprehensible at the first look and therefore they might not reflect the thinking process of the drivers. Moreover, we do not only consider distance in our methodology. Instead, we analyze the data and look for the most standing-out attributes a driver would consider when making his routing choices. We then use these attributes and put them into a weighted objective function that we describe in the following chapters. Subsequently, we use an iterative learning process to find reasonable weights for the different attributes.

3. Methodology

We base our methodology on the concept of decomposing a large complex problem into smaller less-complex sub-problems. By merging the solutions of these sub-problems, we generate a solution for the original problem. Accordingly, our approach first determines a meta-tour, that is, the sequence in which a delivery driver visits the zones of a problem instance. In a second step, our approach computes a delivery order of customers for every zone in the meta-tour and concatenates all delivery orders to generate a solution for the original problem instance. With this zonal approach, we reduce the amount of backtracings, which we define as the number of times a delivery driver returns to already visited zones. Furthermore, we reduce the computational complexity of the problem enabling us to generate solutions within a relatively short time.

This section is structured as follows: Section 3.1 defines the notation used for describing our approach. Section 3.2 documents the preprocessing necessary for our methodology. Sections 3.3 and 3.4 introduce the algorithms used for computing the meta-tour and the customer delivery orders, respectively. Section 3.5 explains our process of parameter tuning.

3.1. Notation

A problem instance consists of a set of customers $I$. Each customer $i \in I$, in turn, can order several parcels $P_i = \{p_{i,0}, \ldots, p_{i,n}\}$ with different time windows $tw_p = [a_p, b_p]$ and estimated service times $s_p$ per
parcel \( p \in P_i \). Furthermore, we denote the estimated travel time between two customers \( i \) and \( j \) by \( c_{i,j} \).

To evaluate a certain generated tour \( S \), we define \( dist_S \) as the total distance and \( v_S \) as the spatial structure adherence score of the considered tour \( S \). The spatial structure adherence score is calculated indirectly through \( v_S^{sp} \) and \( v_S^{sb} \), denoting the number of times the spatial rules are violated in tour \( S \). On the other hand, variables \( t_i \) and \( t_{final} \) represent the estimated point in time at which the driver finishes serving customer \( i \) and the last customer in the tour, respectively. In order to measure delayed delivery, we introduce the metric \( d_{total} \), which indicates the estimated total delay in seconds over all customers of a certain delivery tour. Finally, we use the weights \( w_1 \), \( w_2 \), and \( w_3 \), corresponding to \( v_S^{sp} \), \( v_S^{sb} \) and \( t_{final} \) respectively, to balance the different criteria in our objective functions.

3.2. Preprocessing

Given that a delivery tour must visit each customer only once, we aggregate all parcel time windows \( tw_{p_i,0}, \ldots, tw_{p_i,n} \) to a customer time window \( tw_i \) according to (1).

\[
tw_i = [\max(a_{p_i,0}, \ldots, a_{p_i,n}), \min(b_{p_i,0}, \ldots, b_{p_i,n})]
\]  

(1)

The motivation behind this approach is to set the most restrictive time window per customer. In this way, it is ensured that a driver meets the time windows \( tw_p \) of all parcels \( p \in P_i \) if he meets \( tw_i \).

We use the data field \textit{time window} to model \( tw_p \). Accordingly, \textit{start time utc} and \textit{end time utc} determine the lower \( a_p \) and upper bound \( b_p \) of a time window, respectively. In order to transform the UTC time strings of \textit{start time utc} and \textit{end time utc} into a processable format, we convert the 'HH:MM:SS' component of the time string into the corresponding amount of seconds and assign it to \( a_p \) and \( b_p \), respectively. If time windows stretch until the next day, we consider these daybreaks by adding \( 24 \cdot 60 \cdot 60 \) seconds.

Furthermore, we compute the estimated service time \( s_i \) at customer \( i \) by summing over the service times \( s_p \) of all parcels \( p \in P_i \). As a consequence, \( s_i \) is given by (2).

\[
s_i = \sum_{p \in P_i} s_p
\]

(2)

Finally, we impute potential NaN-values in the \textit{zone id} data field of the stops. The imputation logic depends on the \textit{type}-attribute of the stop with \( \text{zone id} = \text{NaN} \):

- If \textit{type} = \text{Station}, replace \text{NaN} by an artificial 'Start' zone ID
- If \textit{type} = \text{Dropoff}, replace \text{NaN} by the zone ID of the closest stop with \( \text{zone id} \neq \text{NaN} \)

In this way, we maintain the existing zonal structure of the problem instance.

3.3. Meta-tour

3.3.1. Nearest Neighbor

To construct an initial meta-tour, we use the Nearest Neighbor (NN) heuristic. The NN heuristic constructs a tour by finding the next unassigned node which is closest to the previous assigned node. We start

\textit{Dao, Geier and Hilfer}
the search at the depot. It terminates when all nodes are assigned. The nodes in our case are the individual zones. Since there is no data for the distances between zones as they are a set of different customers, we use the single-linkage distance. This distance measure depicts the minimum distance between all pairs of customers of two zones. Using this heuristic, we try to achieve a good initial solution that we can improve with the methods described in subsequent sections.

3.3.2. Iterative 2-Opt

2-Opt: We use 2-Opt to improve the meta-tour generated by the NN algorithm. For this specific challenge, we propose an objective/scoring function, which includes additional components to consider the spatial structure adherence on a zone-level. The general form of that objective function is defined in (3) with \( dist \) and \( v \) being the total distance and spatial structure adherence score and \( w \) being the corresponding weight.

\[
f = (1 - w) \cdot dist + w \cdot v
\]  

(3)

According to what we analyze from high-score routes, this high-level spatial structure can be described by the two following rules:

i. A meta-tour usually visits super-zones (which are represented by the part before the dot in the zone ID, Figure 1) in sequence, i.e., visiting all the zones having the same super-zone code before moving to other zones with other super-zone codes.

ii. Within a super-zone, the meta-tour usually visits zones with the same sub-zone (which are represented by the last character of the zone ID, Figure 1) one after the others.

![Figure 1: Super-zone, sub-zone definition](image)

To illustrate our observations, we look at the actual tour with the route ID RouteID_5c43005c-f08c-4bbf-ae5b-c018047ea384, for example. We can clearly see that it goes from H-10.** to H-9.** and within H-10, it goes from H-10.*H to H-10.*J.

H-10.2H → H-10.3H → H-10.3J → H-10.2J → H-10.1J → H-9.1J → H-9.2J → H-9.3J → ...

We incorporate these insights into (3) by using (4) as the objective function of our approach. For an initial tour \( S \) and tour \( S' \) created from \( S \) by a 2-Opt swap, the change in objective value from \( S \) to \( S' \) is denoted as \( \text{change}_{S,S'} \) and calculated as:

\[
\text{change}_{S,S'} = \begin{cases} 
(1 - w_1) \cdot (dist_{S'} - dist_S) + w_1 \cdot m \cdot (v_{S'}^{sp} - v_S^{sp}) & \text{if } v_{S'}^{sp} - v_S^{sp} \neq 0 \\
(1 - w_2) \cdot (dist_{S'} - dist_S) + w_2 \cdot m \cdot (v_{S'}^{sb} - v_S^{sb}) & \text{otherwise}
\end{cases}
\]  

(4)
where:

- $\text{dist}_{S'}$ and $\text{dist}_S$ are the total distances of tour $S'$ and $S$,
- $v_{S'}^{sp}$ and $v_S^{sp}$ are the number of times going to a zone with different super-zone code of tour $S'$ and $S$ (e.g., H-10.1J $\rightarrow$ H-9.1J is considered as one time),
- $v_{S'}^{sb}$ and $v_S^{sb}$ are the number of times going to a zone with different sub-zone code of tour $S'$ and $S$ (e.g., H-10.3H $\rightarrow$ H-10.3J is considered as one time),
- $w_1$ and $w_2$ are the weights representing the importance degree of spatial structure adherence components, which is set before running the algorithm and which will be tuned at the training step,
- $m$ is a normalized parameter to scale the change in number of times (i.e., $v_{S'}^{sp} - v_S^{sp}$ and $v_{S'}^{sb} - v_S^{sb}$) in the same range as the changing in distance. In the submitted code, it is set to a fixed value of 100 (based on the range of travel times, refer to Figure A.3 in the Appendix).

The remaining part of our 2-Opt works similarly to the original 2-Opt algorithm. If the change in objective value from $S$ to $S'$, or $\text{change}_{S,S'}$ calculated from (4), is negative and significant (i.e., its absolute value is bigger than a certain defined threshold), there is a significant improvement in objective by changing from $S$ to $S'$. We compare every possible valid swap and choose the one with the best improvement (i.e., minimum change). We run this process repeatedly until no improvement is found or a time limit is exceeded.

**Iterative 2-Opt:** In the next step, we propose an iterative 2-Opt process, which is based on the iterated local search algorithm. In each iteration, we perform an adjustment process (perturbation) with the aim to improve the adherence to spatial rules on the tour obtained from 2-Opt. After adjusting, we obtain a new tour which has three characteristics:

i. Zones with the same super-zones/sub-zones are visited consecutively.

ii. The order of super-zones to be visited remains the same as in the original tour.

iii. The order of sub-zones (in each super-zone) to be visited remains the same as in the original tour.

The adjusted tour is accepted if the increase in tour distance is less than a threshold $\text{max\ allowed}$, which will be tuned at the training step. Otherwise, the new tour will be used as new starting solution for the next 2-Opt iteration. In short, this iterative process tries to find a balance between distance optimization and spatial structure adherence.

The pseudo-codes of the adjustment process and iterative 2-Opt algorithm are presented in Algorithm 1 and Algorithm 2.

3.4. *Intra-zone Tour*

This subsection introduces the algorithms necessary for determining the intra-zone delivery order. Such an order sets the sequence, in which a delivery driver services the customers of a certain zone. The driver enters the zone, visits all corresponding customers and leaves for the next zone. Since the driver does not
Algorithm 1 Adjustment process

1: procedure Adjust(tour) ▷ not iterate over the Start zone
2: for i = 1 to len(tour) − 1 do ▷ (i) Case of violation to sub-zone rule
3:     if tour[i] and tour[i + 1] have same super-zone codes then
4:         if tour[i] and tour[i + 1] have different sub-zone codes then
5:             tour ← insert all zones, which are from the index i + 2 and have the same super-zone and
sub-zone codes as those of tour[i], after tour[i]
8:     else
7:         tour ← insert all zones, which are from the index i + 2 and have the same super-zone codes
as those of tour[i], after tour[i]
8: return tour

Algorithm 2 Iterative 2-Opt

1: procedure Iterative2Opt(starting_tour, w1, w2, max_allowed) ▷ starting_tour obtained from NN
2:     starting_tour ← 2opt(starting_tour, w1, w2) ▷ local search: 2-Opt using [4]
3:     while ¬ termination criteria do
4:         new_tour ← adjust(starting_tour); ▷ improve spatial structure
5:         if distance(new_tour) - distance(starting_tour) ≤ max_allowed then ▷ new solution accepted
6:             starting_tour ← new_tour; changed ← FALSE; ▷ get the new tour and exit
7:         else
8:             new_tour ← 2opt(new_tour, w1, w2) ▷ local search with new starting solution
9:     return starting_tour

have to return to the first customer of the zone, the delivery order is modeled as a path instead of a round trip.
In order to compute these paths, we propose the Farthest Insertion (FI) algorithm due to its relatively good computational performance.
However, before FI can be applied, we need to determine a pair of start and end stops for each intra-zone path with more than one node. Since the meta-tour already sets the sequence in which a driver visits the zones, we aim to minimize the travel time between zones. Therefore, we iterate over the meta-tour and select the start stop of an intra-zone path z and the end stop of the preceding path z − 1 such that the travel time between those two stops is minimal. If this results in the same start and end stop for path z − 1, we must select a new end stop for z − 1 to prevent a stop from appearing twice in a delivery order. In such a case, our algorithm chooses the stop in z − 1 that is second closest to the start of z as z − 1’s new end stop.
For the last zone in the meta-tour, we introduce an artificial customer as end stop. This artificial customer can be reached within a travel time of 0 from any of the remaining stops. In this way, we do not fix the last customer of the overall delivery tour and, thus, leave this decision to the optimization algorithm.

3.4.1. Farthest Insertion

The FI algorithm consists of a selection and insertion step. Both steps are repeated until all stops are part of the path.

- **Selection**: The algorithm selects an unvisited stop that is farthest from any of the already visited stops.
- **Insertion**: The algorithm inserts the selected stop optimally into the current tour.

In order to evaluate potential insertion positions, FI employs an objective function that returns a score for a given tour. For the purpose of this routing challenge, we propose an objective function as in (5), which evaluates both tour efficiency and adherence to time windows:

\[
 f(t_{\text{final}}, d_{\text{total}}) = w_3 \cdot t_{\text{final}} + (1 - w_3) \cdot d_{\text{total}}
\]

with \( t_{\text{final}} \) denoting the last customer of the delivery tour and \( w_3 \) denoting the weight for balancing tour efficiency and time window violations. We learn \( w_3 \) during the model-build phase as explained in Section 3.5.

In our approach, a customer \( i \) qualifies as serviced as soon as a driver delivered all of \( i \)'s parcels. In addition, we forbid early deliveries since our experiments have shown that this yields improved scores. As a consequence, a driver must wait until the time window begins if he arrives early. Accordingly, we propose (6) for determining \( t_i \), the point in time at which we finish servicing customer \( i \):

\[
 t_i = \max(a_i, t_j + c_{j,i} + s_i)
\]

with \( j \) being the preceding customer in the delivery tour.

The metric \( d_{\text{total}} \) to measure delayed delivery, in turn, is given by (7):

\[
 d_{\text{total}} = \sum_{i \in I} \max(a_i - t_i, 0),
\]

The starting time of each intra-zone path depends on the point in time, at which a delivery driver finishes servicing the end stop of the preceding path, plus the travel time needed for traveling from that end stop to the start stop of the current path. The starting time of the first zone, the depot, is given by the starting time data field.

In order to generate an overall delivery tour, we iterate over the meta-tour and concatenate the paths of each zone. If a zone consists of one stop, we only append that single stop. If it consists of two customers, we first append the start stop and then the end stop. Given a zone consists of more than two customers, we append the path generated by the FI algorithm. After iterating over the whole meta-tour, we must remove the artificial stop at the end of the delivery tour to get a feasible solution.
3.5. Parameter Tuning

We use the building time to tune parameters $w_1, w_2, w_3,$ and $\text{max\_allowed}$. We run the algorithm with different values for the parameters on a set of only high-score routes (max. 3000 routes) to select values that give the best score. Two aspects should be noted here:

i. Parameters are selected sequentially because it takes less time than combinatorial tuning. Accordingly, we can search for a wider value range while not exceeding the building time.

ii. We divide the high-score routes set into 3 subsets based on the number of super-zones visited in each route, i.e. routes with zones belonging to 1, 2 or $\geq 3$ super-zones. The ratio in size between these subsets among the selected set for tuning is similar to the ratio in size between these subsets among the whole high-score route set provided. Then, parameters are selected differently for each subset.

We introduce these subsets since we observed that the importance of spatial structure adherence is inversely proportional to the number of super-zones visited in each route. It means drivers tend to drive back and forth between super-zones in the route with stops belonging to $\geq 3$ super-zones (refer to Figure A.4 in Appendix).

For each subset, the order and search range of parameter tuning is described as below:

(1) $w_1$: the search range is defined from 0.5 to 0.9 with step size of 0.1 because we want to put more importance on spatial structure adherence than on distance optimization. While tuning $w_1$, other parameters are fixed ($w_2 = 0, w_3 = 0, \text{max\_allowed} = 600$). After tuning, we obtain the best value of $w_1$, denoted as $w_1^\ast$.

(2) $w_2$: the search range is based on the best value of $w_1$ and defined as $[0, 0.5 \cdot w_1^\ast, w_1^\ast]$. Other parameters are fixed ($w_1 = w_1^\ast, w_3 = 0, \text{max\_allowed} = 600$). The best value of $w_2$ after tuning is $w_2^\ast$.

(3) $w_3$: the search range is defined from 0 to 0.5 with step size of 0.1 because we see that time window adherence has a weak effect compared to distance optimization on the intra-zone tour. Parameters $w_1, w_2, \text{max\_allowed}$ are fixed at $w_1^\ast, w_2^\ast, 600$ respectively. The best value of $w_3$ after tuning is $w_3^\ast$.

(4) $\text{max\_allowed}$: three values 150, 300, 600 are selected for tuning. Parameters $w_1, w_2, w_3$ are fixed at $w_1^\ast, w_2^\ast, w_3^\ast$, respectively.

4. Results and Conclusions

4.1. Results

On the 2718 high-score routes provided in the training data set, the best parameter set and the average score based on this set is reported in Table 1. The distribution of the route score of each subset is illustrated in Figure 2.

The route scores are distributed relatively similarly in subset 1 and subset 2. The worst score obtained in subset 1 is about 0.145 while that in subset 2 is about 0.317. The subset 3, which includes routes with at

Dao, Geier and Hilfer
Table 1: Average score and best parameter set on 2718 high-score routes

<table>
<thead>
<tr>
<th>Subset</th>
<th>No. of routes</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>max_allowed</th>
<th>Avg. score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subset 1 (1 super-zone)</td>
<td>497</td>
<td>0.9</td>
<td>0.9</td>
<td>0</td>
<td>300</td>
<td>0.049446</td>
</tr>
<tr>
<td>Subset 2 (2 super-zones)</td>
<td>1928</td>
<td>0.8</td>
<td>0.8</td>
<td>0</td>
<td>150</td>
<td>0.049655</td>
</tr>
<tr>
<td>Subset 3 (at least 3 super-zones)</td>
<td>293</td>
<td>0.8</td>
<td>0.4</td>
<td>0</td>
<td>300</td>
<td>0.062585</td>
</tr>
<tr>
<td>All high-score routes</td>
<td>2718</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.051010</td>
</tr>
</tbody>
</table>

The worst score obtained in this subset is 0.382. The explanation behind the worse performance in subset 3 is that as mentioned previously, the spatial structure adherence plays a less important role in defining the tour in these routes. Consequently, our algorithm for defining the meta-tour cannot work effectively in this case. The best weight for time window adherence in all subsets are 0, suggesting that the time windows may not affect the drivers’ decision.

4.2. Conclusion

We propose an elevated heuristic approach to the specific problem in this challenge. While this algorithm works well with the sets of routes where there are only one or two super-zones to be visited, it performs significantly worse with the routes with super-zones. In addition, there are some potential directions which can improve the current algorithm such as: splitting the current zones into smaller ones to model the inter-(super-)-zone visit (i.e., visiting back and forth one (super-)zone), or finding other factors affecting drivers to integrate into the objective function.
Appendix A. Supplementary figures

Figure A.3: Distribution of travel time in the high-score routes in the training data

Figure A.4: Difference in importance of super-zone rule adherence between route set with 2 and at least 3 super-zones to visit
References


Article XX

Combining Operational Research and Machine Learning to Solve Vehicle Routing Problems

Terence Ngo, Rathea Uth and Romain Loirs

Source code is available on GitHub:
https://github.com/Rathea0286/routiens-app.git
Article XX

Combining Operational Research and Machine Learning to solve Vehicle Routing Problems

Terence Ngo, Rathea Uth, Romain Loirs
École Nationale Supérieure d’Informatique pour l’Industrie et l’Entreprise (ENSIIE)
1 Place de la Résistance, 91000 Évry-Courcouronnes, France

Abstract

This short technical paper introduces an approach to find the routing sequences on the Last Mile Routing Research Challenge organised by Amazon and MIT Center for Transportation and Logistics. To solve the problem, we use an integer linear programming approach. However, it can only solve the problem with at most fifteen stops for an appropriate execution time of ten seconds for each route. Therefore, we added a Machine Learning approach along with clustering techniques to classify stops with the same zone id together. The ML approaches lead to find the first visited zone id in terms of clustering and also to detect and cut the directed links which are not relevant in the complete graph representing in the form of adjacency matrix and consequently reducing the run-times consumption. We mainly use only routes with label “High” when building our ML model. The next step consists of connecting the station to the sequence of each cluster of zone id using Linear Programming (LP) approach, begin with the first zone id to visit provided by ML model. Finally, we tested the performance of our model with all routing data set provided by Amazon and the results show that our model can perform quite well with the best route-level score of 0.003 and the empirical mean score of 0.06497 on 6112 routes versus 0.05942 on the testing set 13 routes.

1. Introduction

The Vehicle Routing Problem (VRP) is one of the most studied combinatorial problems. It consists in optimally serving from a fixed stop of depot a set of customers (stops) with known demand (travel times) while minimising the total cost. It was first introduced by Dantzig & Ramser (October 1959), who proposed a simple matching-based heuristic to route gasoline delivery trucks. Since then, a large part of the Operational Research community has been devoted to this problem which naturally arises in a large variety of practical applications. This trend has been reinforced by the explosion of consumer direct delivery, for example UPS ground delivered around 11.5 million packages in 1993 against around 5.5 billion packages in 2019. This article
aims at combining Optimization and Machine Learning techniques to tackle the Vehicle Routing Problem with Time Windows (VRPTW). A recent trend both in the OR and ML communities has been to apply ML to solve combinatorial optimization problems in order to find the routing sequence with a given data set. Most combinatorial optimization problems, including the VRPTW, are hard to solve, from both theoretical point of view (NP-hard) and practical point of view (computationally expensive techniques, increasing run-times exponentially). Therefore, ML is often seen as a good candidate to tackle these problems. Two main ways of doing so have been explored: using ML to detect and cut the directed links of travel times from stop to stop results in replacing heavy computation by an appropriate execution time, and using ML to find the first zone id to visit, i.e first cluster of zone id besides the depot. These two approaches relied on supervised learning and we also added ideas of clustering zone id using a clustering algorithm with some modifications which leverages the classification knowledge in order to have a homogeneous number of stops for each cluster. We also explore the importance of the training data of our ML model by using only those routes with label “High”, which are about 46% of the given training data set by Amazon. We extend a previous work by Romain (2008) of using the GLPK package to solve our routing problems.

The rest of the paper is organized as follows. In Sec. 2 we review some literature related to the Last Mile Routing problems. We describe our proposed methods in Sec. 3 and present the results in Sec. 4 before concluding.

2. Literature Review

In this section, we review the main articles relating to the methods to find a route that served delivery points for a vehicle. First, Pisinger (2007) presents a general definition of VRP (vehicle routing problem) which consists of seeking an optimal route for a fleet of vehicles, based in one or more depots, in order to serve a set of customers with known orders, and geographically dispersed. A tour refers to all customers visited by a vehicle that leaves and returns to the same depot. He also presents a unified heuristic, which is able to solve five different variants of the vehicle routing problem: the vehicle routing problem with time windows (VRPTW), the capacitated vehicle routing problem (CVRP), the multi-depot vehicle routing problem (MDVRP), the site dependent vehicle routing problem (SDVRP) and the open vehicle routing problem (OVRP). In the article from Mourid St-Pierre (2012), we are mainly interested in methods of exact resolution of VRPTW. In particular, we are interested in linear programs. In fact, this approach is widely used to solve this problem. Our LP model is largely inspired by linear programming models useful for finding a sequence of delivery points. However, we adapted them. Indeed, the main points of divergence of these models from ours are that there are only one delivery truck per road and a single loading stop.

We now present two articles related to the use of Machine Learning in order to complete the work of Operational research. The article Poullet & of Technology. Operations Research Center (2020) first presents a way to design a clustering algorithm leveraging Optimal Classification Trees (OCT), which aims at dividing customers into smaller subsets. Second, it presents an actor-critic Reinforcement Learning (RL) approach to
solve the VRPTW on these smaller customer clusters. Nalepa (2019) shows how to apply machine learning and advanced data analysis techniques to improve routing systems. However, classification methods used later are different from the methods presented in the article. In fact, the dataset of the challenge allow us to define our own method of classification that we will present later.
3. Methodology

3.1. General presentation

The different steps of the algorithm are shown in Fig. 1. The algorithm is composed of two major parts: i) grouping the stops by zone_id in order to find a road which connects all zone_id and ii) finding a road which connects all the stops for each zone_id.

Figure 1: Major steps of the algorithm
3.2. Defining a linear program (LP) to solve VRPTW

We decided to solve the VRPTW thanks to a linear program because, as already mentioned, this approach is often used.

The set of routes between the different delivery points of the network is represented by a connected directed graph $G = (V, E)$ where the set of vertices $V$ corresponds to the delivery points and the set of directed links $E$ corresponds to the paths between these points.

The variables of the problem are: the binary variable $x_{i,j}$ for any $(i, j)$ in $E$ is equal to 1 if the arc $(i, j)$ is chosen in the solution and 0 if not; the travel time of each arc is represented by $p_{i,j}$; the variable $t_i$ for each vertex $i$ in $V$ represents the time at which the consumer $i$ was delivered, where $t_i \in [a_i, b_i]$ which represents the time window in which the package must be delivered to the customer.

The starting vertex $v_0$ of the route is known as well as the final vertex $v_f$. The problem comes down to minimize the objective function

$$\sum_{(i,j) \in E} x_{i,j} \cdot p_{i,j}$$

subject to

\[
\begin{align*}
\sum_{(i,j) \in E} x_{i,j} &= 1 \quad \forall i \in V - \{v_f\} \quad (2a) \\
\sum_{(i,j) \in E} x_{i,j} &= 1 \quad \forall j \in V - \{v_0\} \quad (2b) \\
\sum_{(i,j) \in E} x_{i,j} &= \sum_{(i,j) \in E} x_{j,i} \quad \forall i \in V - \{v_0, v_f\} \quad (2c) \\
\sum_{(i,j) \in E} x_{i,j} - \sum_{(i,j) \in E} x_{j,i} &= 1 \quad i = v_0 \quad (2d) \\
\sum_{(i,j) \in E} x_{i,j} - \sum_{(i,j) \in E} x_{j,i} &= -1 \quad i = v_f \quad (2e) \\
\sum_{(i,j) \in E} x_{i,j} &= 0 \quad \forall i \in V \quad (2f) \\
t_i + p_{i,j} - t_j &\leq M \cdot (1 - x_{i,j}) \quad \forall (i,j) \in E \quad (2g) \\
a_i \leq t_i \leq b_i \quad \forall (i,j) \in E \quad (2h) \\
x_{i,j} \in \{0, 1\} \quad \forall (i,j) \in E \quad (2i)
\end{align*}
\]

The objective function (Eq. (1)) aims to minimise the total travel time of a road. In this problem there are ten constraints. Constraints (2a) and (2b) require each customer be served once and only once, Eqs (2c), (2d) and (2e) assure flow conservation. Eqs (2f) and (2g) ($M$ sufficiently large, meaning greater than the maximum of $p_{(i,j)} +$ maximum of $t_i$) impose on the problem of not having a cycle. These equation are part of the MTZ formulation(see also Miller et al. (1960)). This formulation is chosen among others because it does not require to introduce new variable. In fact, this formulation requires the use of time windows which are already integrated in the problem. Eq. (2h) imposes respect in times constraints. Finally, Eq. (2i) requires that the decision variable be a Boolean.
Then, a lower bound and an upper bound are defined. The lower bound is the cost of the sum of the edges of a spanning tree of minimal weight. The upper bound is the sum of the edges of a road built thanks to the spanning tree of minimal weight.

In order to implement this LP, Python has been used and more precisely the Pulp library (see Mitchell et al. (2011)) and the GLPK package (see Romain (2008)). As expected, this LP is able to solve problems independently from the number of delivery stops. However, the execution time grows exponentially with the number of delivery stops. In fact, less than ten seconds are needed to compute a ten stops problem while a twenty stops problem needs more than sixty seconds.

3.3. Reducing calculation time

3.3.1. Using segmentation to solve quickly sub-problems

As previously shown, the linear program is only able to solve a problem with fifteen stops (in less than 10 seconds) but all the roads contain more than hundreds stops.

In order to solve this problem, we make the hypothesis that a delivery man has to deliver all the packages of zone id before passing to another. In fact, for 63% of “High” labeled road, delivery men deliver all packages of a zone id before going to the next.

However, in the remaining 47%, delivery men pass also from a zone id to another but some packages of another delivery zone id are delivered during the passage in a zone id. In more than 50% of these exceptions, less than 10 stops are delivered when a zone id is currently traveled. Therefore, we have decided to group stops by zone id and to find a sequence in each zone id. Each zone id contains only at most 15 stops and is considered as a sub-problem to solve thanks to the linear program.

3.3.2. Using ML to reduce calculation time

The goal of Machine Learning is to guide and accelerate the search for sequences by Operational Research (OR). In short, we use a binary classification problem of relevant links and irrelevant links, each example processed by Machine Learning model corresponding to an directed link of two stops \( s \to s' \). To do so, we need to define some relevant explanatory variables for the model which are:

- **distance time**: time required to travel (the link) from stop \( s \) to stop \( s' \)
- **five closest**: average time taken from stop \( s \) to go to the 5 stops closest to it (close in the sense of distance time)
- **same zone**: boolean variable which is equal to 1 if stop \( s \) and \( s' \) are in the same zone id, 0 otherwise
- **planned time service**: time taken by the delivery man to deliver all the packages from the stop \( s' \)

The variable to predict is \( y \), which is equal to 1 if the link \( s \to s' \) is used by the sequence, 0 otherwise. We train our model on the route of training set which are in zone id of label “High” using a Logistic Regression model with a 10-fold cross validation (with penalty \( L_2 \)). This method is preferred to other models as SVM,
artificial neural networks, ... for its performances, run-times and its explainability. The goal is to build a model able to predict the value of the target variable $y$ given the values of 4 co-variables $X_1, \ldots, X_4$. The Logistic Regression model is defined by: $\log\left(\frac{p}{1-p}\right) = a_1 X_1 + \cdots + a_4 X_4$ where $a_i \in \mathbb{R}$ are the model parameters and $p = \text{Prob}(Y = 1|X = x)$. The parameters $a_i$ are estimated by the Maximum Likelihood Method with an historical data set (see also King (2008)).

On the testing set, this model gives a F1-score of 0.91 with a recall of 0.96 and a precision of 0.86 (see Fig. A.2 and Fig. A.3). This model is considered good enough to use for our prediction.

The next step consists of a process which is using this binary classification model to prune (cut) links $s \rightarrow s'$ in each cluster which will not be used for the route sequence. However, we need to calibrate the pruning tendency to make sure that there is still a path we can take after pruning the links. In that case, a sufficient condition which ensures the existence of such a path is given by Dirac’s theorem (see Li (2013)). We recall that a Hamiltonian cycle in a graph $G = (V, E)$ is a cycle that visits each vertex exactly once.

**Theorem 1.** (Dirac’s theorem) Let $G = (V, E)$ be a graph with $n$ vertices in which each vertex has degree at least $n/2$. Then $G$ has a Hamiltonian cycle.

We therefore used this condition for pruning. The idea is that, in each cluster of size $n$, we prune in such a way that the degree of each stop is at least $\frac{n}{2} + 1$ (degrees counted with undirected links). For a given link, the closer the model’s prediction is to 0, the lower the probability that the link is relevant, i.e. this link will not be used for the routing sequence, and we will therefore prune it.

**Algorithm 1:** Pruning links in a cluster

Result: A cluster in the form of a matrix by using the Machine Learning model

Input: road_id, ss_adj_matrix, model, scaler, stop_data, route_data, package_data

$n \leftarrow$ number of rows in the matrix;

for $s$ in stops of a cluster do

if degree of $s > n/2$ then

number of pruning arcs remaining $\leftarrow n/2 - \text{(degree of } s) - 1$;

find the list of stop (arc) $s'$ which is still linked to stop $s$;

calculate the sum of (model.predict(arc $s \rightarrow s') + \text{model.predict(arc } s' \rightarrow s)) \text{ and then store them in a list } l$;

sort elements in list $l$ in ascending order;

remove $((2 \times \text{number of pruning arcs remaining}) \text{ link}^{\text{a}} \text{ of the matrix which correspond to (number of pruning links remaining) first values of elements in list } l)$;

end

end

return An adjacency matrix

---

*aWe take into account 2 times because it is necessary to count the outgoing arcs but also the incoming links of $s$
However, since Dirac’s theorem only applies in the case of undirected graph, we must therefore automatically prune the link \( s' \to s \) if we know that the link \( s \to s' \) is pruned. The pseudo code used to prune links in a cluster is presented in the algorithm 1.

Generally by using this methodology, the majority of directed links in clusters are pruned between 30\% to 50\%. However, some clusters are not pruned; these correspond to very small clusters that contain the minority of stops. Technically, it can be noted that pruning arcs method often retains more than 60\% of the relevant arcs to use later on for the next step of Operational Research.

3.3.3. Using parallelization to reduce global time calculation

Previously, we have focused on methods to reduce computation time of a road. However, thanks to the computing capacity of the Amazon’s computer, it is possible to compute several roads at the same time. That is why parallelization is used and more precisely multi-threading. In the algorithm, ten threads are used to reduce computation time.

3.4. Grouping sub-problems to solve the problem.

3.4.1. Finding the first zone_id to visit

In order to find the first visited zone_id, a Machine Learning model is developed. Its goal is to predict the target variable: \texttt{is\_first\_zon} which is equal to 1 when the zone_id is the first visited and 0 otherwise. The goal is to build a model able to predict the value of the target variable \( y \) given the values of 5 co-variables \( X_1, X_2, X_3, X_4, X_5 \) which correspond to:

- \texttt{Nbcollis}: the number of packages to deliver in the zone_id. As shown in Fig. B.4, whatever the road, it would seem that, on average, stops with the maximum number of package are first delivered

- \texttt{Tps\_depot\_to\_first\_zon}: travel time between the station code and the barycenter of the zone_id (i.e the barycenter of all stops contained in the zone_id)

- \texttt{Km\_depot\_to\_first\_zon\_ind\_norm}: number between 0 and 1 where 0 means the barycenter of the zone_id is the closest from the station code and 1 means that the zone_id is the farthest from the station code. The histogram in Fig. B.5b shows that, in most cases, the first zone_id visited is either the closest or the farthest from the station.

- \texttt{distance\_barycentre\_ind\_norm}: number between 0 and 1 where 0 means the barycenter of the zone_id is the closest from the barycenter of the road (ie the barycenter of all the stops in the road) and 1 means that the zone_id is the farthest from this barycenter. Indeed, Fig. B.5b shows that the fist zone_id visited is the farthest from the barycenter of the road.

- \texttt{convex}: number of stops in the zone_id that are also on the border of the convex hull of the road (convex hull of all the stops in the roads). As the last two variables tend to show, the first zone_id
visited is at an extremity. To support this strong hypothesis, we decided to compute the convex hull for each road and create this feature.

Again, the model used is a logistic regression with $L_2$ penalty, trained on a data set composed of zone_id from “High” Label roads.

On the test set (also composed of zone_id from “High” Label roads), the model obtain a F1-score of 0.4 and a ROC AUC of 0.66. As the consequence, the model is not able to predict precisely the first zone_id. In fact, the figure [B.6] demonstrates that the model does not provide the expected first zone_id to visit in most of the cases. However, we have decided to use the model because it slightly improves the score (of 0.01).

3.4.2. Connecting zone_id

The next step is to find a road which connects every zone_id. For this purpose, the linear program (LP) is reused and the algorithm [2] is developed. Its goal is to give a sequence of ordered zone_id, given the first zone_id to visit and the zone_id where the station code is located.

**Algorithm 2: Ordering clusters**

**Data:** adjacency matrix $P$, list of zone_id list $\text{zi}$, first zone_id to visit $\text{first zi}$, zone_id of the station code $\text{sc zi}$

**Result:** An ordered sequence of zone_id

```
Sequence ← []; 
P' ← get_matrice_cluster_as_sommet(P,list_zone_id); 
n ← size of the matrix P'; 
if $n \geq 15$ then 
    cluster_zone_id ← same_size_cluster(P',list_\text{zi},15); 
P'' ← get_matrice_cluster_as_sommet(P',cluster_\text{zone id}); 
    Sequence_cluster ← Linear_programm(P'',\text{first_\text{zi},sc_\text{zi}}); 
    Sequence ← for each cluster of zone_id in Sequence_cluster find a path of zone_id; 
else 
    Sequence ← Linear_programm(P',\text{first_\text{zi},sc_\text{zi}}); 
end
return Sequence
```

Note that for the LP, we do not pass the entire graph as input but a graph $G' = (V', E')$ where the set of vertices $V'$ corresponds to the barycenters of the zone_ids and the set of directed links $E'$ corresponds to the travel time between these points. The latter is the average of the travel time between each couple of stops. The first stop is located in the departure zone_id and the second is located in the zone_id of arrival. The matrix of the problem is named $P'$ and is obtained from $P$ thanks to the `get_matrice_cluster_as_sommet` function presented in the algorithm [2] in the Appendix C.
However, if the number of zone_id is greater than fifteen, zones are grouped according to the travel time, meaning the two closest zone_id are grouped to form a cluster and if a third zone_id is connected to a zone_id of the cluster, the zone_id is added to this cluster. This is the purpose of the same_size_cluster function presented in the algorithm 5 also in the Appendix C. The function also ensures that the first zone_id to visit and the zone_id of the station code are not in the same cluster. If this were the case, it could be impossible to apply the linear program.

3.4.3. Connecting stops to find a road

Now that a road which connects the zone_id is found, the next step is to find a road which connects all the stops in a zone_id. All the steps are summarized in the Algorithm 3. Note that it is necessary for the linear program to know the first and last stop. Thus, for each zone_id, we determine the first stop and the last stop to visit.

Algorithm 3: Prediction

**Data:** adjacency matrix $P$, list of zone_id $Z$, the station code $pt$.

**Result:** An ordered sequence of stops.

1. Sequence $\leftarrow \emptyset$ (an empty list);
2. first_zone $\leftarrow$ Predicting the first zone_id to visit;
3. $Z_{ordonne} \leftarrow$ ranking zone_id thanks to Algorithm 2;
4. for each zone_id in $Z_{ordonne}$ do
   1. Obtaining the starting stop of the next zone_id to visit and finish stop of the current zone_id;
   2. $P'$ $\leftarrow$ the transformed adjacency matrix (Algorithm 1) where the vertices are the stops of the zone_id;
   3. sous_sequence $\leftarrow$ find a road in the zone_id given the first stop and the last stop thanks to the LP;
   4. Sequence $\leftarrow$ Sequence $\cup$ sous_sequence;
5. end

**Return** Sequence

The first zone_id visited contains the station code and it is the first stop of the sequence (for this cluster and the road). The delivery vehicle acquires all packages at the station and delivers them at subsequent drop-off locations. In order to find the station among all stops in a route, Amazon gave us information about the type of stop as a categorical variable whether it’s a station or a drop-off in the file `route_data.json`. This information leads us to easily indicate a stop which is the station for each route.

To find the first stop of the next zone_id and the last stop of the current zone_id, two consecutive zone_id are taken and the shortest travel time (in the adjacency matrix) between two stops of either sides define these two stops.
4. Results and Conclusions

We train our ML model by using only the training set of routes labelled “High”. In order to evaluate our final model before submitting it to the Amazon’s website, we had firstly tested it on the 13 routes of the testing set provided by Amazon in the model apply phase. In addition to that, we also create some partitions of data using the data set provided by Amazon in the model build phase, which divides three different sets of data regarding the label of routes. These three data sets are respectively composed of 2718 routes with label “High”, 3292 routes with label “Medium” and lastly 102 routes of label “Low”. The reason is that we want to analyse the performances of our model on different sets of routes.

Tab. 1 illustrates the distribution of the route-level scores obtained by our model as well as its average score (empirical mean) for each data set that is used for testing purposes for our local performance evaluation of our prediction model.

<table>
<thead>
<tr>
<th>Number of routes tested</th>
<th>Route-level score max</th>
<th>Route-level score min</th>
<th>Empirical mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>2718 high routes</td>
<td>0.3964</td>
<td>0.0030</td>
<td>0.06589</td>
</tr>
<tr>
<td>3292 medium routes</td>
<td>0.3691</td>
<td>0.0042</td>
<td>0.06965</td>
</tr>
<tr>
<td>102 low routes</td>
<td>0.1418</td>
<td>0.0058</td>
<td>0.05937</td>
</tr>
<tr>
<td>testing set 13 routes</td>
<td>0.1878</td>
<td>0.0134</td>
<td>0.05942</td>
</tr>
</tbody>
</table>

Table 1: The distribution of the route-level scores obtained by our model on different sets of routes

One can clearly see that the route-level scores obtained by our model on the 13 routes and 102 low routes seem to be pretty lower (implying that our model is working quite well) than the one that we tested on the training data set of medium and high routes. That is because the number of routes that we use to test the performance of our model are not on the same level. However, the empirical mean of the route-level scores for all routes that we had tested on 6112 routes is equal to 0.06497, the approximate value of average score we expected for our final submission.

Concerning the route-level scores obtained during the various tests on our model, the routes that happen to have a particularly poor score (as shown in Figs D.7, D.9 & D.11) mean that our model failed to predict the first zone_id to visit and that the delivery man also not delivered all the package in the same zone_id before passing to another, i.e that is the exception of our model. Besides, our model can predict fairly well (see examples in Figs D.8, D.10 & D.12) with the lowest route-level score of 0.003 as shown in Fig D.8.

Our model seems to work just fine with the data set provided by Amazon with an appropriate route-level score in empirical mean, but it would be better if we can improve the performance of our ML model that is used to find the first zone id for the sequence by adding more crucial features or new ML model. Furthermore, in order to improve the performance of our model i.e the route-level score, we ought to find a criterion more interesting than that of Dirac’s to allow the ML approach to remove more irrelevant links.
Appendix A. The performance of ML model on routes label “High”

--- TRAIN SET ---
Accuracy: 93.39%
Precision: 86.35%
Recall: 95.17%
F1: 90.54%
ROC AUC: 97.34%

![Figure A.2: The performance on training set (80% of routes label “High”)](image)

--- TEST SET ---
Accuracy: 94.02%
Precision: 86.70%
Recall: 96.39%
F1: 91.29%
ROC AUC: 97.32%

![Figure A.3: The performance on testing set (20% of routes label “High”)](image)
Appendix B. ML model to predict first zone_id - statistical analysis

![Graphs showing average of the maximum size of the deposited package at a stop according to the order of passage of different roads labeled 'Low', 'Medium', and 'High'.]

Figure B.4: Average of the maximum size of the deposited package at a stop according to the order of passage of this one for roads of different label

![Histograms showing distribution of explanatory variables such as distance barycentre norm and depot to first zone norm.]

Figure B.5: Histogram of some explanatory variables of the model to predict first zone_id

![Histograms showing normalized position (divided by the number of zone_id in the road) of the first visited zone_id in actual_sequence for roads with a label “High”.]

Figure B.6: Histograms normalized position (divided by the number of zone_id in the road) of the first visited zone_id in actual_sequence for the roads with a label “High”
Algorithm 4: get_matrice_cluster_as_sommet

Data: an adjacency matrix \( P \), a list of the zone id with the stops contains in each zone id list \( zi \)

Result: a new adjacency matrix

\[
\begin{align*}
&n \leftarrow \text{the number of zone id} ; \\
&\text{res} \leftarrow \text{an empty matrix } n \times n \text{ whose vertices correspond to a zone id} ; \\
&\text{for each zone id } (zi_1) \text{ in list } zi \text{ do} \\
&\hspace{1em} \text{for each zone id } (zi_2) \text{ in list } zi \text{ do} \\
&\hspace{2em} \text{res}[zi_1,zi_2] \leftarrow \text{extract a sub-matrix } P' \text{ from } P \text{ where the vertices of } P' \text{ are the stops in } zi_1 \text{ (for the rows) and } zi_2 \text{ (for the columns) and compute the average of all the coefficients of } P' ; \\
&\hspace{1em}\end{align*}
\]

end

\[\text{return } \text{res}\]

Algorithm 5: same_size_cluster

Data: an adjacency matrix \( P \), a list of the zone id with the stops contains in each zone id list \( zi \), the maximal number of cluster \( \text{max} \)

Result: a list of cluster of zone id

\[
\begin{align*}
&\text{coeff} \leftarrow \text{a decreasing list of the elements of } P ; \\
&\text{res} \leftarrow [] \text{ (an empty list)} ; \\
&\text{while the number of cluster is larger than max do} \\
&\hspace{1em} zi_{\text{start}},zi_{\text{arrival}} \leftarrow \text{the zone ids which correspond to the greater coefficient of the list coeff} ; \\
&\hspace{1em} \text{delete the greater coefficient of the list coeff}; \\
&\hspace{1em} \text{if } zi_{\text{start}} \text{ and } zi_{\text{arrival}} \text{ are not already in a cluster then} \\
&\hspace{2em} \text{adding a cluster formed of } zi_{\text{start}} \text{ and } zi_{\text{arrival}} \text{ in res} \\
&\hspace{1em} \text{else} \\
&\hspace{2em} \text{adding } zi_{\text{start}} \text{ to the cluster of } zi_{\text{arrival}} \text{ in res if } zi_{\text{arrival}} \text{ is already in a cluster and vice versa} \\
&\hspace{1em}\end{align*}
\]

end

\[\text{return } \text{res}\]
Appendix D. Routing sequences comparison with different route-level scores

Figure D.7: Example of a route “High” which is scored very poorly (route-level score = 0.3964) by our model

Figure D.8: Example of a route “High” which is scored very well (route-level score = 0.003) by our model

Figure D.9: Example of a route “Low” which is scored very poorly (route-level score = 0.1418) by our model
Figure D.10: Example of a route “Low” which is scored very well (route-level score = 0.0058) by our model

Figure D.11: Example of a route in testing set which is scored very poorly (route-level score = 0.1878) by our model

Figure D.12: Example of a route in testing set which is scored very well (route-level score = 0.0134) by our model
References


Article XXI

Local Search with Learned Constraints for Last Mile Routing

William Cook, Stephan Held and Keld Helsgaun

Source code is available on GitHub:
https://github.com/heldstephan/jpt-amz
Article XXI

Local Search with Learned Constraints for Last Mile Routing

William Cook
Combinatorics and Optimization, University of Waterloo
200 University Avenue West, Waterloo, ON, Canada

Stephan Held
Research Institute for Discrete Mathematics, University of Bonn
Lennéstr. 2, 53113 Bonn, Germany

Keld Helsgaun
Computer Science, Roskilde University
P.O. Box 260, DK-4000, Roskilde, Denmark

Abstract

We describe our submission to the Amazon Last Mile Routing Research Challenge. The optimization method we employ utilizes a simple and efficient penalty-based local-search algorithm, first developed by Helsgaun to extend the LKH traveling salesman problem code to general vehicle-routing models. We further develop his technique to handle combinations of routing constraints that are learned from an analysis of historical data. On a target set of 1,107 training instances, our submitted code achieves a mean score of 0.01989 and a median score of 0.00752. The simplicity of the method may make it suitable for applications where machine learning can discover rules that are expected (or desired) in high-quality solutions.

1. Introduction

Routing in the Amazon Last Mile Routing Research Challenge is at the level of an individual driver. The data provide a list of stops a driver must visit to deliver packages on a given day, together with point-to-point travel times. The travel times are asymmetric, that is, the time to travel from stop $x$ to stop $y$ may not be the same as the time to travel from $y$ back to $x$. The task of finding the quickest route for the driver is an instance of the asymmetric traveling salesman problem (ATSP).

Direct optimization of routes is not the focus of the challenge, however. Teams are asked to match, as best they can, the sequences of stops in actual routes driven by van drivers, who may have taken into consideration warehouse operations, current road conditions, types and sizes of packages to be delivered,
and other factors. In machine-learning fashion, models and algorithms developed with the help of provided training data are evaluated by their performance on an additional large set of routes that are kept hidden from competing teams.

1.1. Initial scores

As a first step towards a solution procedure, we consider optimal tours for the ATSP instances. The Concorde code \cite{Applegate2006} solves symmetric instances of the TSP, so we applied a standard transformation of the ATSP to the TSP, splitting each stop into a pair of nodes, one for incoming edges and the other for outgoing edges. The TSP instances thus obtained were solved with Concorde in an average of 142 seconds of computation time on a single core of a Linux workstation, equipped with an Intel Xeon Gold 6238 CPU @ 2.10GHz processor. This is not an efficient way to solve instances of the ATSP, but it allowed us to easily obtain optimal tours for the set of 1,107 High+Delivered training instances (those with driver routes that are classified as ‘High’ quality and have no undelivered packages; the target class for the final competition).

<table>
<thead>
<tr>
<th>Solution Set</th>
<th>Mean Travel Time</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver Tours</td>
<td>12250.0 sec</td>
<td>0.0</td>
</tr>
<tr>
<td>ATSP Tours</td>
<td>10853.3 sec</td>
<td>0.07030</td>
</tr>
</tbody>
</table>

The driver tours are on average 12.9% longer than routes that minimize travel time.

The optimal ATSP tours achieved a competition score of 0.07030. The score of a tour is a measure of its similarity with the corresponding driver tour. Details of the score computation can be found at the challenge GitHub site \url{https://github.com/MIT-CAVE/rc-cli/tree/main/scoring}. A lower score indicates a better match, the driver tour itself scoring a perfect 0.0.

The ATSP tours are only a weak approximation of the driver sequences. For example, the tour order does not reflect any conditions related to the warehouse operation or to the driver’s storage of packages in the van he/she will be using that day. Regarding this, the training data provide a zone ID for each stop, indicating a possible clustering of the route’s stops in a tour.

There are an average of 20 distinct zones represented in each of the High+Delivered routes, giving an average of approximately 7 stops per zone. The zones partition the set of nodes in the ATSP instance, where we create a special zone containing only the delivery station. (In cases where a stop in the training data does not have a zone ID, we assign to it the ID of the closest stop (in Euclidean distance) associated with the same station.) It is not difficult to modify an ATSP instance to force the stops in each of these zones to appear consecutively in any optimal tour. This can be achieved by adding a large constant $M$ to the travel time of any edge joining stops in distinct zones. With this modification, we again used the ATSP to TSP transformation and solved the resulting instances with Concorde.

<table>
<thead>
<tr>
<th>Solution Set</th>
<th>Mean Travel Time</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustered-ATSP Tours</td>
<td>11235.2 sec</td>
<td>0.04866</td>
</tr>
</tbody>
</table>
The optimal clustered-ATSP tours obtained were 3.5% longer than the tours without the cluster restrictions, but they reduced the similarity score to 0.04866.

1.2. Precedence constraints, tour neighbors, and super clusters

We view the clustered instances as the starting point for the Last Mile Challenge. Our research program aimed to obtain properties of driver tours that could restrict the clustered-ATSP instances to achieve a better match with the choices made by the driver and by the warehouse operators. The restrictions are obtained from analysis of the routes in the training data, making direct use of the zone-to-zone sequences in the driver tours, as well as the zone ID patterns these sequences create.

All new restrictions we consider are at the inter-zone level, while maintaining the clustering of stops within each zone. A natural constraint is to force a precedence between a pair of zones \((a, b)\) as they appear in a tour, that is, if we let \(\text{visit}(x)\) denote the position of zone \(x\) in the tour (with the delivery-station zone having position 0), then a precedence constraint for \((a, b)\) requires \(\text{visit}(a) < \text{visit}(b)\). A path constraint for \((a, b)\) is the stronger restriction that zone \(a\) immediately precedes zone \(b\) in the tour, that is, \(\text{visit}(a) = \text{visit}(b) - 1\). We also consider neighbor constraints, requiring that zone \(a\) is either immediately before or immediately after zone \(b\) in the tour, that is, \(|\text{visit}(a) - \text{visit}(b)| = 1\). Furthermore, we permit in our models logical combinations of these three types of constraints; particularly useful are binary disjunctions, stating that for two specified constraints \((A, B)\), a solution tour must satisfy either \(A\) or \(B\).

Together with constraints involving pairs of zones, we also create super clusters, forming a partition of the zone clusters, and super-super clusters, forming a partition of the super clusters. This hierarchy allows us to create precedence, path, and neighbor constraints for pairs of super clusters and pairs of super-super clusters, further guiding the algorithm-produced tours towards sequences followed by actual drivers.

1.3. Optimization engine

Super clusters, super-super clusters, and neighbor constraints (at all levels of the hierarchy) can be incorporated into an ATSP model by modifying travel costs, as we described in the clustered-ATSP case. The inclusion of precedence constraints, path constraints, and logical disjunctions make such a transformation much more difficult. We therefore do not rely on an ATSP solver in our work, developing instead a penalty-based local-search heuristic that can handle multi-level clustering and logical combinations of our three types of constraints. The new heuristic optimizer allows us to compute high-quality tours in computation time well under the limit set in the final competition.

2. Literature Review

The local-search paradigm is simple: starting with any candidate solution to a problem, repeatedly look for small alterations that can possibly lead to a better solution. The idea dates back to work on the TSP, starting with the 2-opt heuristic by Flood (1956), that was quickly extended to 3-opt by Bock (1958) and...
Croes (1958), and later to a variable $k$-opt algorithm by Lin & Kernighan (1973). Extensions of the basic paradigm are discussed in surveys by Lourenço et al. (2003) (iterated local search), Hoos & Stützle (2005) (stochastic local search), and Alsheddy et al. (2018) (guided local search), and in the book Aarts & Lenstra (2003), covering a range of applications in discrete optimization.

Of particular interest in our work is the use of local search in the area of constraint programming, where penalties are used to guide the heuristic towards feasible solutions; see Hoos & Tsang (2006) for a survey. Ideas from this constraint-programming work were adopted to handle time-window-constrained routing problems in López-Ibáñez & Blum (2010) and Nagata et al. (2010).

Helsgaun (2017) incorporated penalty-based search into his LKH-3 code for vehicle-routing models, greatly extending the reach of his well-known TSP code LKH (Helsgaun (2000)). We use in our work the simple and efficient method deployed in LKH-3, to measure simultaneously the length of the route and the penalties incurred when constraints are violated.

3. Methodology

Our approach adopts modest machine-learning techniques, but combines this with a powerful heuristic optimization method to exploit the discovered properties of driver routes.

3.1. LKH-AMZ

The tour-finding heuristic is called LKH-AMZ, acknowledging its origins in the LKH-3 code. LKH-AMZ streamlines LKH-3 by implementing a restricted set of $k$-opt moves, suitable for the short computation times called for in the Last Mile Challenge. At the same time, LKH-AMZ expands the family of constraints that can be handled, giving us great flexibility in adopting rules that are learned from training routes.

3.1.1. Penalizing constraint violations

LKH-AMZ treats the initial zone-clustering restrictions as hard constraints, adopting the travel-cost transformation described in Section 1.1. The remaining restrictions are treated as soft constraints, handled by penalizing any violations.

Cluster, super-cluster, and super-super-cluster penalties are evaluated with a simple $O(n)$ mechanism that loops through the stops in tour order, counting the numbers $e_c$, $e_{sc}$, and $e_{ssc}$ of tour edges entering different clusters, super clusters and super-super clusters, respectively. If we have $k_c$ clusters, $k_{sc}$ super clusters, and $k_{ssc}$ super-super clusters, then the assigned penalties are $10 * (e_c - k_c)$, $10 * (e_{sc} - k_{sc})$ and $10 * (e_{ssc} - k_{ssc})$, respectively. Note that cluster penalties are not strictly necessary, since they are handled by the cost transformation.

The precedence, path, and neighbor constraints are evaluated by examining the tour positions of the constrained pairs of clusters, super clusters, and super-super clusters. It is possible to set the penalties...
at different values for individual pairs, but to avoid over-fitting we adopt only two penalty levels. Cluster-precedence constraints and cluster-neighbor constraints receive a penalty of 1 when violated, and cluster-path constraints and all super-cluster and super-super-cluster constraints receive the penalty 1000.

Disjunctions of constraints are evaluated as the minimum penalty of each pair in the disjunction.

3.1.2. Tour representation

We make use of the standard ATSP to TSP transformation (see Jonker & Volgenant (1983)). With each node $i$ of the original problem there is associated a dummy node $i+n$ and a fixed edge $(i, i+n)$ having travel cost 0. LKH-AMZ organizes its tours so that every original node follows its dummy node in the tour orientation. A two-level list data structure (see Fredman et al. (1995)) is used to represent the tour, supporting fast $k$-opt operations.

3.1.3. Special 3-opt, 4-opt moves

We say a tour $T'$ improves a tour $T$ if at least one of its penalty or length is strictly less than that of $T$ and neither of these values has increased. Given a tour $T$, and a non-fixed edge $(t_1, t_2)$, the function SpecialMove attempts to find a 3-opt or 4-opt move that improves $T$.

Since every original node must follow its dummy node in the tour, a move is not allowed to reverse any segment of the tour. This leaves only two possible 3-opt and 4-opt moves.

Figure 1a shows the choice of nodes $t_1, t_2, t_3, t_4, t_5$, and $t_6$ for a 3-opt move. Note that node $t_5$ must lie between $t_2$ and $t_3$ in the tour’s orientation. Figure 1b shows the result of the move. Observe that no segments have been reversed.

Figure 2a shows the choice of nodes $t_1, t_2, t_3, t_4, t_5, t_6, t_7$, and $t_8$ for a 4-opt move. Note that again $t_5$ must lie between $t_2$ and $t_3$, and $t_7$ must be between $t_4$ and $t_1$. The resulting move is displayed in Figure 2b. To keep the time complexity low, only four possible combinations of $(t_5, t_6)$ and $(t_7, t_8)$ are tried (those nearest to $t_1, t_2$ and $t_3, t_4$, respectively).

After a move $T \Rightarrow T'$ has been made, it is tested whether $T'$ improves $T$. In that case, the current tour $T$ is set to $T'$. Otherwise, the move is retracted.
3.1.4. Candidate edges

LKH-AMZ restricts the search for moves by means of candidate edges. These are the edges that are considered for possible inclusion in the tour. To reduce the number of such edges, an edge-elimination procedure based on minimum spanning trees and subgradient optimization is applied (Helsgaun (2000)). The candidate edge set is further sparsified by eliminating those edges whose inclusion would violate any neighbor constraints. This latter step both speeds up the search for moves and guides the search towards solutions satisfying the constraints.

3.1.5. Iterated local search

A trial in LKH-AMZ is a sequence of 3-opt and 4-opt moves found by SpecialMove, where the search for these moves is terminated according to rules that trade off the speed of the computation with the likelihood of finding further improvements. (See Applegate et al. (2006) for a detailed discussion in the context of the pure TSP.) In each run of the code, \( n \) trials are executed, where \( n \) is the number of nodes in the input data.

The trials are organized as an iterated local search. The idea is to allow the code to quickly sample local minima in the neighborhood of the best tour \( T^* \) that has been found thus far. The initial trial begins with a pseudo-random tour. In each subsequent trial, the current \( T^* \) is “kicked” by performing a 4-opt move of the type indicated in Figure 2; the 4-opt kick is selected using the Rohe long-edge rule with a 50-step random walk, as described in Applegate et al. (2003). The kicked tour is re-optimized via a sequence of calls to SpecialMove. If this search produces a tour \( T \) that improves \( T^* \), then we replace \( T^* \) by \( T \).

3.1.6. Multiple runs

Until a specified time limit is reached, we carry out multiple runs of the full iterated local search. Each run is initiated with a pseudo-random tour. We evaluate the resulting tour \( T \) by the function \( v(T) = \text{PenaltyMultiplier} \times \text{Penalty}(T) + \text{Cost}(T) \), and record \( T \) as a new best tour if \( v(T) \) is less than the \( v(\cdot) \) value of our current best tour. The role of the scaling factor \( \text{PenaltyMultiplier} \) is to allow us to have both the penalty and cost of the tour play a role in the decision to accept the result of the run; it is set to 1500 in our tests, bringing the typical scaled \( \text{Penalty}(T) \) to a value comparable to \( \text{Cost}(T) \).
3.2. Zone precedence constraints from training data

To predict the order in which a driver traverses the zones in a new routing instance $r$, we compute a reference route $q^*$ from the training instances. We choose $q^*$ among all routes that start at the station of $r$, sharing the maximum (scaled) number of common zones with $r$. As we favor reference routes with a ‘High’ route score, we scale the number of common zones by a factor $2/1.5/1$ for ‘High’/‘Medium’/‘Low’ scored routes. We then extract zone-order constraints based on $q^*$ as follows.

During the model-build phase we compute a zone graph for every route. It contains a vertex for every visited zone and an edge $(a, b)$ if the driver served a package in zone $a$ immediately before a package in zone $b$. Contracting all its strongly connected components yields the component path. Its vertices consist of strongly connected components of zones. (See Tarjan (1972).)

In the apply phase, we create a precedence constraint $\text{visit}(a) < \text{visit}(b)$ for each pair $(a, b)$ of zones whose strongly connected components $A \ni a$ and $B \ni b$ are joined by an edge $(A, B)$ in the component path of the reference route.

Adding precedence constraints to the Clustered-ATSP instances reduces the score from 0.04866 to 0.03414, where the tours are obtained by running LKH-AMZ for 10 seconds per instance.

Alternatively, we use the transitive closure of all such precedence constraints. This reduces the score to 0.03157. However, in combination with constraints from Section 3.3, non-transitive constraints gave better results on average.

We also tried to extract zone-order constraints from multiple reference routes, but did not find a model that improves the single reference route approach.

3.3. Cluster rules from zone IDs

The zone IDs assigned to stops have the form $\Gamma-x.y\Delta$, where $\Gamma$ and $\Delta$ are capital letters and $x$ and $y$ are integers. In Table 1 we list these IDs in the order they appear in the driver tours for each of the first four High+Delivered routes. The lists have patterns, suggesting drivers are following higher-level clusters created by transitions in the four components of the zone IDs. In our build phase, we create such a clustering by selecting a subset $S$ of the symbols \{\(\Gamma\), $x$, $y$, $\Delta$\} and grouping all zone IDs that have matching values in the $S$ positions. Super clusters are determined by a selection $S$ of three symbols, super-super clusters are determined by a selection $T \subset S$ of two symbols, and a top-level clustering is determined by $U \subset T$ having a single symbol. For example, setting $S = \{\Gamma, x, \Delta\}$, $T = \{\Gamma, x\}$, and $U = \{\Gamma\}$ gives for route amz0002 from Table 1 the super clusters \{A-2.2E, A-2.1E\}, \{A-2.1D, A-2.2D, A-2.3D\}, \{A-2.3C, A-2.2C, A-2.1C\}, \{A-2.1B, A-2.2B\} and a single super-super cluster consisting of all of the zones. In our submission, the choices of $S$, $T$, and $U$ are made in a build-phase computation, minimizing the number of times the training tours cross the components of the partitions of zones at each clustering level.

Adding these super clusters and super-super clusters to our test instances improves the score to 0.02347 with 10-second runs of LKH-AMZ. Going further with the analysis, within each super cluster $C$ we sort by
Table 1: Sequences of zone IDs in driver tours

<table>
<thead>
<tr>
<th>amz0002</th>
<th>amz0003</th>
<th>amz0012</th>
<th>amz0019</th>
</tr>
</thead>
<tbody>
<tr>
<td>Station</td>
<td>Station</td>
<td>Station</td>
<td>Station</td>
</tr>
<tr>
<td>A-2.2E</td>
<td>B-11.1G</td>
<td>A-2.1E</td>
<td>M-10.3D</td>
</tr>
<tr>
<td>A-2.1E</td>
<td>B-11.1H</td>
<td>A-2.2D</td>
<td>M-10.3C</td>
</tr>
<tr>
<td>A-2.1D</td>
<td>B-11.2H</td>
<td>A-2.3D</td>
<td>M-10.2C</td>
</tr>
<tr>
<td>A-2.2D</td>
<td>B-12.1G</td>
<td>A-2.3C</td>
<td>M-10.1C</td>
</tr>
<tr>
<td>A-2.3D</td>
<td>B-12.3G</td>
<td>A-2.2C</td>
<td>M-10.1B</td>
</tr>
<tr>
<td>A-2.3C</td>
<td>B-12.2G</td>
<td>A-2.1D</td>
<td>M-10.2B</td>
</tr>
<tr>
<td>A-2.2C</td>
<td>B-12.3G</td>
<td>A-2.1C</td>
<td>M-10.3B</td>
</tr>
<tr>
<td>A-2.1C</td>
<td>B-12.1H</td>
<td>A-2.1B</td>
<td>M-10.3A</td>
</tr>
<tr>
<td>A-2.1B</td>
<td>B-12.2H</td>
<td>A-3.2A</td>
<td>M-10.2A</td>
</tr>
<tr>
<td>A-2.2B</td>
<td>B-12.3H</td>
<td>A-2.1A</td>
<td>M-10.1A</td>
</tr>
<tr>
<td></td>
<td>B-12.3J</td>
<td>A-2.2B</td>
<td>M-10.2A</td>
</tr>
<tr>
<td></td>
<td>B-12.2J</td>
<td>A-2.3B</td>
<td>P-10.1A</td>
</tr>
<tr>
<td></td>
<td>B-12.1J</td>
<td>A-2.3A</td>
<td>P-10.2A</td>
</tr>
<tr>
<td></td>
<td>B-11.1J</td>
<td>A-3.1A</td>
<td>P-10.3A</td>
</tr>
<tr>
<td></td>
<td>B-11.2J</td>
<td>A-2.2A</td>
<td>P-10.3B</td>
</tr>
<tr>
<td></td>
<td>B-11.3J</td>
<td>A-2.3A</td>
<td>P-10.2B</td>
</tr>
<tr>
<td></td>
<td>B-11.3H</td>
<td></td>
<td>P-10.1B</td>
</tr>
<tr>
<td></td>
<td>B-11.3J</td>
<td></td>
<td>P-10.1C</td>
</tr>
<tr>
<td></td>
<td>B-11.2G</td>
<td></td>
<td>P-10.2C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>P-10.3C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>P-10.3D</td>
</tr>
</tbody>
</table>

zone ID the clusters contained in \( C \), and add neighbor constraints for each adjacent pair of clusters in the sorted order. These constraints put the clusters within a super cluster in either sorted or reverse sorted order. To attempt to force our tours to match the driver’s choice between these two orderings, for neighboring pairs of super clusters \((C, D)\) we possibly add zone-level neighbor constraints for pairs of zones \((c, d)\), with \( c \) being either the first or last zone in the sorted order for \( C \) and \( d \) being either the first or last zone in the sorted order for \( D \). The neighbor constraint \((c, d)\) is added if these zones have matching values for the unique symbol \( u \) in \( \{\Gamma, x, y, \Delta\} \setminus S \). If there are two such pairs of zones for \((C, D)\), then we create a disjunction for the pair of neighbor constraints. Corresponding constraints are also added for sorted orders of super clusters within super-super clusters, and for sorted orders of super-super clusters within each top-level cluster. Adding the entire collection of new constraints leads to a score of 0.02146.

To complement the constraints created with zone ID patterns, we also add super-cluster path constraints derived from the set of training routes. Here we follow the idea from our work on cluster-level precedence. For a routing instance \( r \), we find a reference route \( q^* \) having the greatest number of super clusters in common with \( r \). We add a path constraint for pairs of super clusters \((C, D)\), such that \( C \) and \( D \) appear in both \( r \) and \( q^* \), the clusters \( C \) and \( D \) are each entered exactly twice in \( q^* \), and \( C \) and \( D \) appear consecutively in \( q^* \). These additional constraints improve the score to 0.02030 and we adopt them in our default code.

As an alternative, it is possible to add a super-cluster precedence constraint for \((C, D)\) rather than a path constraint, but this gives the slightly worse score of 0.02079.
3.4. Merging families of tours

LKH-AMZ is designed to very quickly find good tours satisfying all or most of the specified constraints. Indeed, it is not always productive to increase the LKH-AMZ time limit.

<table>
<thead>
<tr>
<th>Time per Route</th>
<th>10 sec</th>
<th>20 sec</th>
<th>30 sec</th>
<th>40 sec</th>
<th>50 sec</th>
<th>60 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>0.02030</td>
<td>0.02001</td>
<td>0.01997</td>
<td>0.01997</td>
<td>0.02003</td>
<td>0.02002</td>
</tr>
</tbody>
</table>

We instead use a portion of the computation time to run LKH-AMZ on an alternate set of constraints and then merge the two collections of tours. To merge, we compute the travel times $t_d$ and $t_a$ for a route’s default and alternate tours, then select the default if $t_d \leq \text{MergeFactor} \times t_a$, and otherwise select the alternate. Setting $\text{MergeFactor} = 1.01$ allows us to replace outlier tours in our collection.

In our submission, the alternate set of constraints is obtained from the default set by selecting instead the transitive-closure precedence constraints in Section 3.2 and the super-cluster precedence constraints in Section 3.3; we allocate twice as much running time for the default constraints.

<table>
<thead>
<tr>
<th>Time per Route</th>
<th>Default</th>
<th>Alternate</th>
<th>Merged</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 sec + 5 sec</td>
<td>0.02030</td>
<td>0.02176</td>
<td>0.02009</td>
</tr>
<tr>
<td>20 sec + 10 sec</td>
<td>0.02001</td>
<td>0.02116</td>
<td>0.01989</td>
</tr>
<tr>
<td>40 sec + 20 sec</td>
<td>0.01997</td>
<td>0.02113</td>
<td>0.01989</td>
</tr>
</tbody>
</table>

3.5. Computer implementation

The LKH-AMZ heuristic solver, written in the C programming language, consists of approximately 5,300 lines of code. Python scripts, totaling 2,600 lines, are used for analyzing zone information, extracting constraints, and visualizing tours. An additional 500 lines of C implement an internal scoring function, 300 lines of shell scripts control the LKH-AMZ solver and execute the merge routine, and 100 lines of scripts control the build and apply phases of the submission to the challenge. All codes and scripts are available under the MIT License at [https://github.com/heldstephan/jpt-amz](https://github.com/heldstephan/jpt-amz).

4. Results and Conclusions

Codes submitted to the Last Mile Challenge are evaluated on a set of 3,050 routes, running on an AWS EC2 m5.4xlarge server and a time limit of 12 hours for the build phase and 4 hours for the apply phase. The m5 server has an 8-core processor and supports 16 virtual cores. The results presented above were carried out on a linux server equipped with two 16-core Intel Xeon Gold 5218 CPU @ 2.30GHz processors, supporting a total of 64 virtual cores. The two machines have roughly similar single-core performance.

Our build-phase analysis completes in under 5 minutes on a single core of the linux server, so well within the competition time limit. To accommodate the 20s+10s merge runs in the 4-hour apply-phase limit, we execute in parallel the LKH-AMZ runs on individual routes. Using 64 threads on the linux server, the total wall-clock computation time for the 20s+10s merge was 617 seconds.
The model_apply script in our submitted code determines the amount of time to allocate to LKH-AMZ after the constrained ATSP instances are created, aiming for a total run time of 3.5 hours using 16 threads. In a test on an AWS EC2 m5.4xlarge server using 3,050 routes, the code adopted a 41s+20s merge.

4.1. Distribution of scores

We have thus far reported only the mean value of the 1,107 individual tour scores for the High+Delivered routes, which is the measure used to rank the submissions in the Last Mile Challenge. More detail can be seen in the histogram of the scores given in Figure 3. The mean score is greatly impacted by a small number of poor results. Indeed, the median score of 0.00752 is well under the 0.01989 mean.

4.2. Final remarks

- We were not successful in learning useful constraints for individual stops, due in part to the sparse coverage of the training routes when observed at the stop level. We expect that more sophisticated learning approaches developed by other teams could be combined with our optimization engine.

- LKH-AMZ can very quickly, in a small number of seconds, obtain low travel-time tours that achieve good competition scores. This speed may make the code suitable for real-time optimization, where updated travel times could be uploaded and a new tour computed.

- LKH-AMZ is set to handle time-window constraints, but we did not utilize this in our submission. We found that the small number of such constraints did not impact the scores we obtained.

- The choices for PenaltyMultiplier and MergeFactor can be learned in the build phase, but given the single trial permitted in the final evaluation, we chose the conservative approach of setting their values to 1.01 and 1500 respectively, the midpoints of the [1.00,1.02] and [1000,2000] ranges we observed as acceptable values in our study. A trial run of a build script on an AWS server, covering a random sample of 1,000 of the High+Delivered training instances, selected PenaltyMultiplier = 1250 and MergeFactor = 1.016, but with only a slight score improvement over the default midpoint values.
References


*Cook, Held and Helsgaun*


Article XXII

Combining Naive Bayes Model and TSP Routing Algorithm for Predicting High Quality Route Sequences

Wenbin Zhu, Ying Fu and Luo Zhixin
Article XXII

Combining naive bayes model and TSP routing algorithm for predicting high quality route sequences

Wenbin Zhu and Ying Fu
School of Business Administration, South China University of Technology
381 Wusun St. Tianhe District, Guangzhou, GD, China

Luo Zhixin
School of Management and Engineering, Nanjing University
Hankou Road 22, Gulou District, Nanjing, Jiangsu Province, China

Abstract

In last mile delivery, experienced drivers often have tacit knowledge, such as which road is hard to navigate. Generally, such knowledge is hard to formulate explicitly and is provided as a set of historical high quality routing sequences. We try to learn the tacit knowledge and predict visiting sequences of unseen future route. Based on the observation that drivers tend to serve customers zone by zone, we decompose the task into a high-level zone sequence prediction and a low-level TSP tour construction. Assuming Markov property, we employ naive bayes model for predicting zone sequences. Historical data are employed to establish transition probability between zones and we seek to identify a zone sequence that maximizes likelihood of being observed. An iterative merging algorithm inspired by clustering algorithm is devised to train the naive bayes model. After predicting zone sequence, we compute the TSP tour for stops in each zone to form visiting sequences of stops. Compared with conventional optimization algorithms, such as heuristic TSP or VNS that attempts to minimize total travel distance, our method produces sequences closer to the actual sequence taken by the drivers. Five-fold cross validation shows the average score of our method is 0.0739. Our method also outperforms sequence to sequence neural network that attempts to directly predict the visiting sequence at stop level.

1. Introduction

Given a set of historical routing sequences with quality labeled low, medium and high by logistic operation experts, our model learns the characteristics of high-quality sequences and predicts visiting order of future unseen high-quality sequences. The performance of our model is measured by how well the predicted visiting
sequence matches the actual one by weighted editing distance (the scoring function provided by the contest organizer) between the two sequences. It is quite different from conventional routing optimization problem, where the quality of solution is measured by precisely defined objective functions such as total travel time. We believe the key to achieve a good match is to figure out the criteria used by drivers and logistic experts explicitly or implicitly to measure the quality of route sequence. Since the criteria is not explicitly described, we must learn from historical data.

Our first attempt is to utilize the successful sequence to sequence models in many NLP tasks to directly train a neural network based on historical routing sequence. This model performs poorly with an average score of 0.1124 over 2,718 historical high-quality routes, worse than 0.0864 by a heuristic TSP algorithm that merely minimize total distance. We suspect the poor performance is mainly due to the lack of data. Since historical sequence consists of 33 to 238 stops, much longer than the number of words in typical sentences, our model is dealing with prediction tasks larger in scale than typical NLP models. We suspect that more data are required to handle the lager prediction tasks. However, we only have 6,112 historical sequences, whereas typical NLP models are trained with huge corpora that consist of millions of sentences.

Data suggest that drivers tend to visit customers zone by zone, that is, customers in same zone tend to appear consecutively in route sequence. Merging consecutive stops with same zone in historical routes results in a visiting sequence of zones. The average length of zone sequence is about 22, much lesser than the average number of stops in routes, 148. Further analysis shows that zones seldom repeat in a zone sequence. Therefore, we decompose the prediction task into two sub-problems. We first predict the visiting sequence at zone level, then try to predict the visiting order of stops within a zone. For the former task we employ a naive bayes model that tries to maximize the likelihood of zone sequence based on conditional probability estimated from historical data. For the later task, we try to compute the TSP tour of stops in a zone to minimize the total travel time. It turns out that both tasks can be solved by computing a TSP tour with suitably defined distance matrix.

2. Literature Review

The traveling salesman problem (TSP) is one of the most widely studied combinatorial optimization problems, which consists of determining a set of routes for one or several salesmen who visit a given number of cities exactly once and return to the initial position [Cheikhrouhou & Khouri 2021]. Owing to its NP-hardness, TSP is regarded as one of the most challenging problems in Operational Research [Pandiri & Singh 2018]. To address the traditional TSP and its variants, three types of approaches, namely exact algorithms, heuristics and meta-heuristics, and other approaches, have been investigated in the literature.

Exact algorithms aim to achieve the optimal solution, and hence are computationally expensive and more suitable for small-sized TSPs. The related existing literature is quite limited. Bellman [1962] proposed a dynamic programming algorithm for TSP. Dell’Amico et al. [2021] presented a branch-and-bound algorithm for the flying sidekick TSP. In this algorithm, a fast lower bound, dominance rules, and three heuristic
methods were integrated. Yuan et al. [2020] proposed a branch-and-cut algorithm for the generalized TSP with time windows. To reduce the computation times, an initial upper bound was provided by a simple and fast heuristic. To deal with a new TSP with hotel selection, Barbosa & Uchoa [2020] developed a sophisticated branch-cut-and-price algorithm. Computational results showed that many medium-sized instances, having up to 75 clients and 20 hotels, could be solved to optimality.

Due to great difficulties in solving the TSP and its variants, recent research efforts have been focused on developing efficient heuristics and meta-heuristics. The heuristics are simple strategies that help produce good solutions. One of the commonly used heuristics for TSP is 2-opt. The meta-heuristics, which is capable of iteratively improving the candidate solutions, have also been introduced to find approximate solutions within reasonable computational cost. The existing meta-heuristics for TSP are classified into two categories: the single solution-based meta-heuristics (SSBM) and the population-based meta-heuristics (PBM). The SSBM maintains only one solution at each cycle and have the power to intensify the search in local regions. The commonly used SSBMs are simulated annealing (SA) Küçükoğlu et al. [2019], tabu search (TS) Lin et al. [2016], variable neighborhood search (VNS) Wang et al. [2019], and large neighborhood search (LNS) Smith & Imeson [2017], etc. Different from SSBMs, PBMs are either based on evolutionary strategies or swarm intelligence. These approaches use the properties of the population to guide the solution to the optimality, therefore are capable of performing a good exploration of search space. The most used PBMs for TSP in the literature include genetic algorithm (GA) Groba et al. [2015], particle swarm optimization (PSO) Mahi et al. [2015], and ant colony optimization (ACO) Wang & Han [2021], etc.

Different techniques, such as probability, game theory, and fuzzy logic, have also been applied for TSP. Kulkarni & Tai [2010] introduced a method using probability collectives, in which the vehicles were represented as autonomous agents and vehicles route as a strategy. Khoufi et al. [2016] proposed a multi-objective optimization problem to determine the robots tours responsible for collecting data from sensor nodes and delivering the data to the depot. A game theory approach was developed to optimize the maximum tour time, the number of robots, and balance the robots’ tours. To address a multi-objective TSP, a fuzzy logic-based approach was developed by Trigui et al. [2017]. This approach combined both metrics into a single fuzzy metric and reduced the problem to a single-objective optimization problem.

An extensive literature review indicates that most of research works neglect the travelers’ experience when solving the TSP and its variants. Since machine learning is capable of providing useful predictive strategies based on expert knowledge, a promising way is to integrate meta-heuristics with machine learning methods for problem-solving, which may provide better solutions in reasonable computation time.

Sequence-to-sequence neural networks have achieved good performance in many natural language processing tasks. Vinyals et al. [2015] extend this framework to pointer network that uses attentions to output a permutation of input sequences. It is capable to produce high quality route for small scale problems, they reported results for TSP with up to 50 stops. Bello et al. [2016] employ reinforcement learning to train the pointer network without supervised solution. Their approach can find high quality TSP tour with up to 100 stops.
stops. Deudon et al. [2018] combine reinforcement learning with a 2OPT local search to solve TSP and Kool et al. [2019] combine reinforcement learning with transformer architecture to produce TSP tour. In terms of minimizing total distance, machine learning based methods cannot compete with hand-crafted state-of-art heuristics yet, nevertheless, they have the potential to pickup domain specific knowledge from training data. The three reinforcement learning based approaches do not require known TSP tour as supervised sample, but they all need a function that can accurately evaluate the quality of a route. In our research challenge, there is no explicitly qualitative measure for the quality of route.

3. Methodology

Given a set of historical route sequences with route_score set to low, medium and high by logistic experts. Our task is to predict the visiting order of stops in unseen high quality route sequences. Since drivers tend to visit stops zone by zone, stops within same zone tend to appear consecutively in historical route sequences. We decompose the prediction task into two simpler tasks: 1. predicting the visiting order of zones; 2. computing the visiting order of stops within zones.

3.1. Predicting zone sequence

3.1.1. Naive bayes model

To predict the visiting order of zones, we first merge all stops with same zone_id in a historical route sequence to obtain a zone sequence where stops without zone_id are discarded. Since zones seldomly repeat in historical zone sequences (Table 1), we simplify the prediction task by assuming no zone will repeat. We employ a naive bayes model to predict zone sequences.

<table>
<thead>
<tr>
<th>repeated zones</th>
<th>route count</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3835</td>
<td>62.75%</td>
</tr>
<tr>
<td>1</td>
<td>988</td>
<td>16.16%</td>
</tr>
<tr>
<td>2</td>
<td>673</td>
<td>11.01%</td>
</tr>
<tr>
<td>3</td>
<td>288</td>
<td>4.71%</td>
</tr>
<tr>
<td>≥ 4</td>
<td>328</td>
<td>5.37%</td>
</tr>
</tbody>
</table>

Based on historical zone sequences, we estimate the conditional probability \( P(Z_{i+1} = b | Z_i = a) \) of visiting a next zone \( b \) given current zone \( a \). We assume Markov property holds, that is, the probability of visiting a next zone only depends on current zone but not any other zone visited before current:

\[
P(Z_k | Z_1, Z_2, \cdots, Z_{k-1}) = P(Z_k | Z_{k-1})
\]
Repeatedly apply Markov property, the probability of observing a zone sequence $Z_1, Z_2, \cdots, Z_k$ can be decomposed into

\[
P(Z_1, Z_2, \cdots, Z_k) = P(Z_k | Z_1, Z_2, \cdots, Z_{k-1}) P(Z_1, Z_2, \cdots, Z_{k-1})
\]

\[
= P(Z_k | Z_{k-1}) P(Z_1, Z_2, \cdots, Z_{k-1})
\]

\[
= \cdots
\]

\[
= \left( \prod_{i=1}^{k-1} P(Z_{i+1} | Z_i) \right) P(Z_1)
\] (2)

We would like to find a zone sequence with maximum probability. Since a sequence always starts and ends with station, $P(Z_1)$ is constant, maximizing (2) is equivalent to the following likelihood:

\[
\prod_{i=1}^{k-1} P(Z_{i+1} | Z_i)
\] (3)

Finding a zone sequence that maximizes the likelihood as defined by (3) is our naive bayes model for predicting zone sequence.

3.1.2. Estimating transition probability

We estimate the conditional probability of $P(Z_{k+1} = j | Z_k = i)$ through empirical distribution based on high quality routes. More precisely, given a training set $\mathcal{X}$ consists of some historical zone sequences (we randomly chosen 80% of the high quality route sequences to form a training set),

\[
P(Z_{k+1} = j | Z_k = i) \approx \frac{\text{# of zone sequences where zone } j \text{ immediately follows } i}{\text{# of zone sequences includes zone } i}
\] (4)

There are 17 stations in historical data, and the same zone id may appear in two stations and refer to two different zones, as data suggest they are more than one thousand kilometers apart. Therefore the transition probability as defined in equation (4) is computed station by station, for each station the transition probability between any pair of zones are computed. The matrix $P = \{p_{ij}\}$ where $p_{ij} = P(Z_{k+1} = j | Z_k = i)$ is often called transition matrix in the context of Markov chain decision.

There are 8,962 unique zone id, but only 2,718 high quality routes. Each route visits 22 zones on average. The transition matrices estimated through (4) are very sparse, merely 0.04% to 0.30% entries are nonzero. This is a sign of lack of sufficient data to estimate a reliable transition matrix. Therefore, we also tried an alternative method to estimate the transition matrix as follows.

- For each zone pair $i, j$ appear in a zone sequence, we define the rank of zone $j$ to be $r$ if $j$ is the $r$-th closest from $i$.

- We define the conditional probability $P(r | Z = i)$ to be the probability of visiting the $r$-th closest zone from $i$ immediately after zone $i$. These conditional probabilities are estimated using a training set consists of historical high quality routes.
Then the transition probability from zone $i$ to zone $j$ is defined as $P(Z_{k+1} = j|Z_k = i) = P(r|Z_k = i)$, where $j$ is $r$-th closest zone from $i$.

We call this method *estimation by rank*.

A third method to estimate the un-normalized transition probability is through distance between two zones. Let $d_{i,j}$ be the travel time from zone $i$ to $j$. We define $p_{i,j} = e^{-d_{i,j}}$, that is, we subjectively set the likelihood of visiting $j$ after $i$ to be higher if $j$ is closer to $i$. The main rational behind this rather ad-hoc method is that, we observed that most of the zones visited next are the top 3 closest zone.

### 3.1.3. Solving naive bayes model

Given transition matrix among zones, our Bayes model (3) can be solved by minimizing its negative logarithm:

$$- \log \prod_{i=1}^{k-1} P(Z_{i+1}|Z_i) = \sum - \log P(Z_{i+1}|Z_i)$$

(5)

There are two possible methods to solve this minimization problem. First, we can define a distance matrix $D = \{d_{ij}\}$ with entries representing the negative logarithm of conditional probability of visiting zone $j$ after zone $i$:

$$d_{ij} = - \log P(Z_{k+1} = j|Z_k = i)$$

(6)

A zone sequence that minimizes $\{5\}$ is a sequence that starts and ends with station and minimizes total distance, this can be solved by any standard TSP algorithm taking $D$ as distance matrix.

Our second method is inspired by an iterative clustering algorithms, by iteratively merging two shorter zone sequences into a longer zone sequence as follows:

- **step 1**, initialization: create one trivial zone segment for each zone

- **step 2**, iterative merging: pick an ordered pair of zone segments $A$ and $B$ that are most likely to be visited in order, and merge the pair into a single zone segment.
  - **step 2a**, the likelihood of visiting zone segment $B$ after zone segment $A$ is defined to be the transition probability between the last zone in segment $A$ to the first zone in segment $B$. Note that two segment $A$ and $B$ cannot be merged if $A$ starts with station and $B$ ends with station and there are other segments left. To avoid this special case we define their likelihood to be negative infinity.
  
  - **step 2b**, to merge an ordered pair of zone segment $A$ and $B$ into a single segment and the resulting segment is most likely as defined by $\{3\}$. There are two modes: first, we try to **append** $B$ after $A$. Second, we exhaustively divided both $A$ and $B$ into two sub-segments and recombine them. For example if $A$ is divided into $A1$ and $A2$, and $B$ into $B1$ and $B2$. We combine the four segments into a single segment in the order of $A1, B1, A2, B2$.

- **step 3**, repeat step 2 until there is only one zone segment left.
3.2. Computing TSP tour within zones

Let $Z_1, Z_2, \cdots, Z_k$ be a zone sequence, where $Z_1$ and $Z_k$ includes only the station. We construct a stop sequence by computing TSP tour for stops within each zone and connecting them as follows:

- $Z_1$ only consists of the station, its entry stop $e_1$ is the station.
- $s = \text{a trivial stop sequence consists of only the station}$
- For each zone $Z_i, i = 2, \cdots, k$
  - step a: we try to identify a stop $e_i$ in $Z_i$ that is closest to previous zone $Z_{i-1}$. We call $e_i$ the entry stop of $Z_i$. For a stop $s$ in zone $Z_i$ its distance to previous zone is defined as the shortest distance to any non-entry stop in previous zone.
  - step b: compute a TSP tour that starts with the entry stop $e_{i-1}$ of previous zone $Z_{i-1}$, visiting all other stops in $Z_{i-1}$ exactly once and ends at $e_i$ the entry stop of zone $Z_i$.
  - step c: append the TSP tour in step b to end of $s$. Note that the first stop in new TSP tour is the last stop in the partial route sequence $s$.

To compute a TSP tour, we first invoke the greedy heuristic implemented by the python package https://pypi.org/project/tsp-solver2/. We try to improve the TSP tour by sliding a window along the sequence as follows:

- Initialization: let $s_1, s_2, \cdots, s_n$ be the initial stop sequences and $k$ be the size of sliding window
- For $i = 1$ to $n - k$, take a sub-tour starts at the $i$-th stop in $s$ that consists of $k$ consecutive stops. Fix the start and end stop of the sub-tour and rearrange the visiting order of middle stops in the sub-tour. When the length of sub-tour is short, say $k \leq 10$ the optimal rearrangement can be computed by dynamic programming algorithm by Bellman \[1962\].

4. Results and Conclusions

We conduct computational experiments to decide the best configuration of our approach. Since there is a learning component in our approach, we follow the hold-out strategy commonly used by machine learning community to train and validate our model. We randomly sample 80% of high-quality historical route sequences to form the training set, and use the remaining 20% as validation set. The process is repeated five times. Each time we use training set to train our model and then compute two performance measurement of our model. Firstly, our model predicts the sequence for each route in validation set and the prediction is compared with actual sequence using weighted editing distance as described on the website of this research challenge. The average score over validation set measures how well our model generalizes to unseen data. Secondly, we also randomly select the same number of routes as validation set from the training set and report
the average performance of our model on the selected training routes. The second measure is an indication
whether our model have learned enough from the training set. The small weighted editing distance indicates
a good match. Perfect match will result in a score of 0 and poor match will result in a large score close to 1
or even larger than 1.

4.1. How estimating transition probability affect prediction accuracy of zone sequence

In the first experiment, we try to compare different method of estimating transition matrix as described
in section 3.1.2 and their performance on predicting zone sequences. The actual zone sequence is computed
by merging stops with same zone id in historical route sequences. Occasionally, some zone appear more
than once. In this case, we will only keep the occurrence resultant from merging the most number of stops.

We use TSP as baseline. For every zone, the centroid of stops in the zone is taken as the location of
the zone. The travel distance between two zone are approximated by the geographical distance (see https://en.wikipedia.org/wiki/Geographical_distance) between their centroids. We invoke the heuristic tsp
solver in python package tsp-solver2 to obtain a zone sequence that minimizes the total travel distances. The
result of our baseline algorithm is reported in first row in Table 2. Column “validation score” is the average
score on validation set and “in sample score” is the average score on training set. We can see both score are
rather poor for the baseline algorithm. The next two rows report the zone sequence predict by our iterative
merging method described in section 3.1.3. They differ in how transition probability between two zones are
estimated. For the row merge(rank), estimation is done by rank as described in section 3.1.2 is employed;
for the row merge(empirical), estimation is by empirical probability as described by equation (4). As we
can see, estimation by rank performs better on validation set with an average score of 0.0957, in contrast to
0.1564 by empirical.

Both merge(rank) and merge(empirical) are substantially better than our baseline TSP algorithm. There-
fore, we decided to use our iterative merging algorithm to solve our naive bayes model to predict zone
sequences.

4.2. Comparing different algorithms on zone sequence prediction

We described two algorithms for solving our naive bayes model in section 3.1.3 Regardless how we
estimate the transition probability, our iterative merging algorithm performs better than TSP algorithm, see
Table 3. For example, we transition probability is estimated empirically by equation (4), iterative merging
algorithm achieves an average score of 0.1564 on validation set (3rd row in the table), much better than 
0.2534 produced by TSP.

<table>
<thead>
<tr>
<th>estimating probability</th>
<th>algo</th>
<th>validation score</th>
<th>in sample score</th>
</tr>
</thead>
<tbody>
<tr>
<td>by_rank</td>
<td>merge</td>
<td>0.0957</td>
<td>0.0942</td>
</tr>
<tr>
<td></td>
<td>tsp</td>
<td>0.1640</td>
<td>0.1628</td>
</tr>
<tr>
<td>emperical</td>
<td>merge</td>
<td>0.1564</td>
<td>0.0163</td>
</tr>
<tr>
<td></td>
<td>solve_tsp</td>
<td>0.2534</td>
<td>0.1767</td>
</tr>
</tbody>
</table>

Therefore, we decided to use iterative merging algorithm to solve our naive bayes model for predicting the zone sequence.

4.3. Performance on predicting route sequence

Our overall method for predicting route sequence is described as follows. Firstly, we estimate the transition probability between two zones using historical data. Secondly, based on the estimated transition matrix, we employ iterative merging algorithm to predict the zone sequence and then convert zone sequence into stop sequences by computing TSP for stops in each zone as described in section 3.2. The size of sliding window \( k = 10 \) is used, so that the dynamic programming formulation of a sub-tour can be computed within 1 seconds on a moderate laptop.

We compare the effect of two methods of estimating transition probability, their results are summarized under heading “merge(rank)” and “merge(empirical)” respectively in Table 4. Training and validation is repeated five times, the average score on validation and training set for each repetition are reported. The last two rows report the average and the stand deviation across the five repetition.

<table>
<thead>
<tr>
<th>repetition</th>
<th>merge(rank) validation</th>
<th>in sample</th>
<th>merge(empirical) validation</th>
<th>in sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0959</td>
<td>0.0542</td>
<td>0.0757</td>
<td>0.0178</td>
</tr>
<tr>
<td>2</td>
<td>0.0850</td>
<td>0.0594</td>
<td>0.0729</td>
<td>0.0185</td>
</tr>
<tr>
<td>3</td>
<td>0.0843</td>
<td>0.0677</td>
<td>0.0775</td>
<td>0.0172</td>
</tr>
<tr>
<td>4</td>
<td>0.0781</td>
<td>0.0704</td>
<td>0.0714</td>
<td>0.0178</td>
</tr>
<tr>
<td>5</td>
<td>0.0772</td>
<td>0.0720</td>
<td>0.0723</td>
<td>0.0173</td>
</tr>
<tr>
<td>avg</td>
<td>0.0841</td>
<td>0.0647</td>
<td><strong>0.0739</strong></td>
<td>0.0177</td>
</tr>
<tr>
<td>std</td>
<td>0.0067</td>
<td>0.0068</td>
<td><strong>0.0023</strong></td>
<td>0.0005</td>
</tr>
</tbody>
</table>
Although merge(rank) achieve better zone level prediction accuracy than merge(empirical) as shown in Table 2, its performance on predicting stop sequences are worse with an average score of 0.0841 on validation set, in contrast to 0.0739 by merge(empirical). The performance of merge(rank) is also less stable across repetition as indicated by a larger standard deviation of 0.0067, in contrast to 0.0023 by merge(empirical).

In our submission, we estimate transition probability by empirical probability and we utilize all historical high quality routes available.

Figure 1 shows the histogram of the score of 545 route sequences in the validation set produced by our submitted model in repetition 2. More than half of scores are less than 0.063, whereas only 30 scores exceed 0.171.

4.4. Other failed attempts

We implemented a few alternative methods.

- **TSP**: we ignored the delivery time window of all packages and try to compute a TSP tour that minimizes the total travel time. The average score of this method is 0.0846 on all high quality historical routes.

- **VNS**: for each drop off stop, if any package has a delivery time window, we compute a time window for the stop. The start time for a stop is the latest start time of all packages and the end time for stop is the earliest end time of all packages. If the driver arrives at a stop before start time, it incurs a penalty proportional to the earliness; if the driver arrives after end time, it incurs a penalty proportional to the lateness. We devise a variable neighborhood search heuristic that employ well-known operators such as swap, relocate to minimize the total travel time and penalty. Our VNS is slightly better than TSP with an average score about 0.08. When we further attempt to reduce earliness and lateness, it produces route sequence with worse scores.
• ML: a pure machine learning based method. Given current stop $i$, we try to predict $f(i, j)$, the number of stops visited before we visited a stop $j$. To enrich input features, we also includes the nearest $k = 5$ unvisited stops for $j$ and $i$, respectively. That is to say, input of function $f$ will includes features of 12 stops. For each stop, we include features, such as lat, lng, package count, total service time, etc. We employ a two layer fully connected network with ReLU activation function to model $f$. After the model is trained, to predict a new route, we start with $i$ being the station, then invoke $f$ with every unvisited stop $j$ to find the stop with smallest $f(i, j)$ and use it as next stop. The average score of route sequences produced by this model is 0.1125, which is worse than TSP score.
References


Article XXIII

A Hybrid Route Planning Heuristic Based on Simulated Annealing and Iterated Local Search

Fellipe Pessanha and Pedro Vasconcellos

Source code is available on GitHub:
https://github.com/fellipessanha/LMRRC-Team-AIDA
Article XXIII

A Hybrid Route Planning Heuristic Based on Simulated Annealing and Iterated Local Search

Fellipe Pessanha  
*Institute of Mathematics and Statistics, Fluminense Federal University*  
*Rua Professor Marcos Waldemar de Freitas, Niterói, RJ, Brazil*

Pedro Vasconcellos  
*Department of Computer Science, Fluminense Federal University*  
*Av. General Milton Tavares de Souza, Niterói, RJ, Brazil*

Abstract

We present a solution for the Amazon Last Mile Routing Research Challenge focused on simplicity, according to the Less-is-More Principle: a hybrid heuristic based on the well-known metaheuristics Simulated Annealing and Iterated Local Search. The proposed heuristic tries to minimize an objective function calculated on the following features: the total distance of the route, the number of executed backtrackings, and the number of time-window violations. When considering the 6000 routes available for the challenge, the proposed approach achieved a mean score of 0.09 over 2718 selected routes for benchmarks limited by 4 hours. The mean score varied in a range of 0.01 to 0.42 with a standard deviation of 0.05.

Acknowledgements

We would like to thank our project advisors for their insight and support.

Igor Machado Coelho  
*Department of Computer Science, Fluminense Federal University, Av. General Milton Tavares de Souza, Niterói, RJ, Brazil*

Daniela Vianna  
*Independent Researcher, Bellevue, WA, USA*

Fernanda Passos  
*Department of Telecommunication Engineering, Fluminense Federal University, Rua Passo da Pátria, Niterói, RJ, Brazil*

Simone Martins  
*Department of Computer Science, Fluminense Federal University, Av. General Milton Tavares de Souza, Niterói, RJ, Brazil*
1. Introduction

The competition proposed by Amazon focused on finding routes that match the expected high quality goals, provided by a training set of around 6000 routes. Our approach focused on simplicity, according to recent advances in the metaheuristics field (Gendreau et al., 2010; Talbi, 2009) and the Less-is-More (LIM) Principle (Mladenović et al., 2016). We propose a hybrid heuristic composed of metaheuristics Simulated Annealing (Kirkpatrick et al., 1983) and Iterated Local Search (Lourenço et al., 2019) (ILS) with Variable Neighborhood Descent (Mladenović & Hansen, 1997) (VND) local search. The SA-ILS-VND quickly explores the solution space, starting from poor quality random solutions, then converging into high quality solutions according to a guiding objective function. This function is geared toward simplicity, according to LIM, and takes into consideration three significant aspects of the route: traveled time, violations of time window constraints, and backtrack violations. The violations are multiplied by a big numeric factor to decrease the advantage of these solutions compared to the traveled time.

This document has the following organization. This section introduces the problem and the general outline of the proposed technique. Section 2 describes in detail the state-of-the-art methods and tools used in this work. Section 3 digs into the methodological aspects of this work, so as implementation details, whenever is necessary. Finally, Section 4 closes the work, presenting some results and conclusions.

2. Literature Review

Recent advances in Hybrid Metaheuristics (Talbi et al., 2013) have demonstrated successful scenarios where the composition of multiple metaheuristic frameworks is able to improve upon challenging problems. State-of-the-art metaheuristics typically rely on a balance between exploration and exploitation. They do not waste time on the generation of poor-quality solutions while achieving a minimal amount of diversity so that the method is not stuck on poor-quality local optima. In this sense, Simulated Annealing (Kirkpatrick et al., 1983) has been one of the most successful metaheuristics since its inception in the ‘80s. Although quite simple, it allows quick exploration of poor-quality solutions, which is a great starting point for local-search-based metaheuristics, such as Iterated Local Search (Lourenço et al., 2019) and Variable Neighborhood Search (Mladenović & Hansen, 1997).

2.1. Simulated Annealing

The simulated annealing (SA) (Kirkpatrick et al., 1983) metaheuristic starts from an initial generated solution and consists of a procedure that iteratively generates a single neighbor \( s' \) of the current solution.
s. If \( s' \) has a better cost than \( s \), the method accepts the move and the solution neighbor becomes the new current solution. If it has a worse cost, the candidate neighbor solution can also be accepted, but in this case, with a probability that depends on the temperature parameter \( T \), which regulates the probability to accept worse cost solutions. The temperature \( T \) initially assumes a high value and is gradually decreased by a cooling ratio \( \alpha \). In the beginning, there is a greater chance to escape from local minimums. As \( T \) approaches zero, the algorithm intensifies the search, as it decreases the probability of accepting worsening movements.

In our implementation, the initial temperature parameter \( T \) was calculated using the experimental function \texttt{estimateInitialTemperature}, described in the Algorithm \[1\]. This function starts from an initial arbitrary low temperature value \( T_0 > 0 \), and multiplies it by a heating factor \( \bar{\beta} > 1.0 \) until a given ratio of accepted solutions \( \gamma \in [0, 1] \) is achieved for that temperature. This process is iterated based on maximum number of iterations on a given temperature, \( SA_{\text{max}} \) (Souza, 2011). The rationale of this process is simple: a good temperature is one, as lowest as possible, that provides enough solutions to be accepted directly, without any worsening. If temperature is too low, there is no space for diversity. If temperature is too high, too much time is lost on diversification.

The value of \( \bar{\beta} = 1.1 \) was set experimentally, as 10% rise of temperature allows a quick heating for a controlled calibration of parameter \( \gamma \). The method is not typically very sensitive to changes in \( \bar{\beta} \). The value \( \gamma = 0.7 \) represents a 30% diversity target (100% − \( \gamma \)), which was also set experimentally, and \( T_0 = 1 \) is typically a low temperature value as seen in the literature (Souza 2011). The value of \( SA_{\text{max}} = 500 \) was estimated based on the computational time used in our machines, since the process was meant to finish in few seconds maximum. If more computational time was available, this value could be certainly increased, so as the SA method could also be iterated. None of these variables were heavily tuned, although few local randomized experiments did not indicate that slight changes could improve the outcome of the method. Naturally, a solution generator \( s_{\text{gen}}() \) is needed for the process (we adopted random solutions); also some neighborhoods \( N() \) (we adopted the same as in the SA method) and a guiding function \( f() \) (also the same as in the SA method).

The Simulated Annealing method is very similar to the parameter estimation method. Parameter \( \alpha = 0.98 \) is a geometric cooling parameter typically used in literature (Souza 2011). The other parameters are the same from estimation method, including \( f(), N(), s_{\text{gen}}(), SA_{\text{max}} \), and finally, initial temperature \( T_0 \) starts from estimated temperature \( T_\Theta \).

A strong characteristic of Simulated Annealing is its ability to escape from poor quality local optima, while starting from poor quality solutions (typically random). Its weakness, however, is that it typically cannot progress much further, as long as a good quality solution is found. For this reason, it adopts a hybrid strategy by complementing the search process with some techniques specifically designed for Local Search.
**Algorithm 1** estimateInitialTemperature($f(), N(), s_{gen}(), \tilde{\beta} = 1.1, \gamma = 0.7, SA_{max} = 500, T_0 = 1$)

$$T \leftarrow T_0;$$

$$s \leftarrow s_{gen}();$$

$Continue \leftarrow true;$

while (Continue) do

$$Accepted = 0;$$

for (IterT = 1, 2, ..., $SA_{max}$) do

Generate neighbor $s' \in N(s);$

$$\Delta \leftarrow f(s') - f(s);$$

if ($\Delta < 0$) then

$$Accepted \leftarrow Accepted + 1;$$

else

Let $x \in [0, 1]$;

if ($x < e^{-\Delta/T}$) then

$$Accepted \leftarrow Accepted + 1;$$

end if

end if

end for

if (Accepted $\geq \gamma \cdot SA_{max}$) then

$$Continue \leftarrow false;$$

else

$$T \leftarrow \tilde{\beta} \cdot T;$$

end if

end while

$$T_{\Phi} \leftarrow T;$$

return $T_{\Phi};$

**Algorithm 2** BasicSimulatedAnnealing($f(), N(), s_{gen}(), \alpha = 0.98, SA_{max} = 500, T_0 = T_{\Phi}$)

$$T \leftarrow T_0;$$

$$s \leftarrow s_{gen}();$$

$s^* \leftarrow s;$

while $T > 0.000001$ do

for (IterT = 1, 2, ..., $SA_{max}$) do

Generate neighbor $s' \in N(s);$

$$\Delta \leftarrow f(s') - f(s);$$

if ($\Delta < 0$) then

$$s \leftarrow s';$$

if ($f(s) < f(s^*)$) then

$$s^* \leftarrow s;$$

end if

else

Let $x \in [0, 1]$;

if ($x < e^{-\Delta/T}$) then

$$s \leftarrow s';$$

end if

end if

end for

end while

return $s^*$;

2.2. Metaheuristics based on Local Search

Iterated local search (ILS) [Lourenço et al., 2019] is a stochastic local search method that iteratively applies local search after perturbing the current search point, leading to a randomized walk in the space of local optima. To implement this metaheuristic, first, an initial solution is generated. Then, during some iterations, a solution is perturbed to generate new starting points, and a local search is applied to the incumbent solution.

Variable Neighborhood Descent (VND) [Mladenović & Hansen, 1997] is a heuristic that initially selects a set of neighborhood structures $N_l, l = 1, ..., l_{max}$. The following sequence is applied until no improvement
is obtained. The local search uses the first neighborhood \((l = 1)\). Then, the following steps are performed, until the last neighborhood is used \((l = l_{\text{max}})\): If the best neighbor \(s'\) obtained is better than the incumbent solution \(s\), \(s\) is set to \(s'\), and the search is going to use the first neighborhood; otherwise, the next neighborhood is used.

The combination of ILS_VND is presented in the algorithm 3. This algorithm needs an initial solution generator \(s_{\text{gen}}\) and a time \(\text{Timelimt}\) stop criteria, which will be explained in the Methodology section, a neighbor structure \(\text{NeighborStructure}\), containing first improvements 2Opt and Swap Moves, a perturbation parameter \(P_0\) describing the initial number of times the local search will be applied, and a perturbation limit \(X\), the number of times a neighbor search must run without improvement before we increase the value of \(P\).

### Algorithm 3 ILS_VND\((f(), \text{NS}_\text{List}(), s_{\text{gen}}(), P_{\text{max}}, \text{NoImproveIterMax})\)

\[
\begin{align*}
s &\leftarrow s_{\text{gen}}(); \\
s^* &\leftarrow \text{VND}(s, f, \text{NS}_\text{List}); \\
P &\leftarrow 1; \text{Iter} \leftarrow 0; \\
\text{while } (P < P_{\text{max}}) \text{ do} & \\
& \quad \text{if } (\text{Iter} = \text{NoImproveIterMax}) \text{ then} \\
& \quad \quad P \leftarrow P + 1; \text{Iter} \leftarrow 0; \\
& \quad \text{end if} \\
& \quad s \leftarrow s^*; \\
& \quad \text{for } (\text{Perturb} = 1 \text{ to } P + 2) \text{ do} \\
& \quad \quad N() \leftarrow \text{randomly chosen from } \text{NS}_\text{List}; \\
& \quad \quad \text{Generate neighbor } s' \in N(s); \\
& \quad \quad s \leftarrow s'; \\
& \quad \text{end for} \\
& \quad s' \leftarrow \text{VND}(s, f, \text{NS}_\text{List}); \\
& \quad \text{if } f(s') < f(s^*) \text{ then} \\
& \quad \quad s^* \leftarrow s'; \\
& \quad \quad P \leftarrow 1; \text{Iter} \leftarrow 0; \\
& \quad \text{end if} \\
& \text{end while} \\
& \text{return } s^*;
\end{align*}
\]

### Algorithm 4 VND\((s, f(), \text{NS}_\text{List}())\)

\[
\begin{align*}
k &\leftarrow 0; \\
s^* &\leftarrow s; \\
\text{while } (k < 2) \text{ do} & \\
& \quad s' \leftarrow \text{FirstImprovement}(s^*, \text{NS}_\text{List}(k)); \\
& \quad \text{if } (f(s') < f(s)) \text{ then} \\
& \quad \quad k \leftarrow 0; \\
& \quad \quad s^* \leftarrow s'; \\
& \quad \text{else} \\
& \quad \quad k \leftarrow k + 1; \\
& \quad \text{end if} \\
& \text{end while} \\
& \text{return } s^*;
\end{align*}
\]

2.3. Computational Aspects

Dealing with large JSON files in memory-constrained scenarios is also a challenging problem in the literature. From a simplicity perspective, the currently most recommended C++ library for JSON is

\[Pessanha and Vasconcellos\] XXIII.5
nlohmann::json[^6] rather than other more complex (but more efficient) ones, such as RapidJSON[^3]. We discovered that both of these libraries suffer from excessive memory consumption, so we decided to code our own open-source JSON library, specialized for big data scenarios, named Vast.JSON[^4].

Finally, from an implementation perspective, we considered the C++ optimization framework OptFrame in C++[^5](with bindings to other popular languages such as Python[^6]). OptFrame has been developed since 2008, being successful in many state-of-the-art optimization problems, with single or multiple objectives (Coelho et al., 2010, 2011, 2016b). The framework contains classic metaheuristics and it recently introduced the concept of a functional-style library, named OptFrame Functional Core (Coelho et al., 2020).

3. Methodology

3.1. Problem Representation

The problem consists of three files that were made available for the contest: new_package_data.json, new_route_data.json, and new_travel_times.json. Each file contains several problem instances, each one called a RouteID. The Route Data describes the positions of the stops and their respective zones. The Package Data indicates time window constraints for each package, destined to some stop. The Travel Times contains the time spent by the vehicle to move from one stop to another.

3.2. Solution Representation

The solution is represented as a permutation of stops, starting at the station. We used a `std::vector<int>` in C++, where the number \( j \) at position \( i \) indicates that stop \( j \) is in the \( i \)-th position of the route (appearing after stop \( i - 1 \)). Naturally, for \( i = 0 \), \( j \) denotes the station for the given RouteID.

3.3. Neighborhood Structures

We consider two different neighborhood structures, or moves, to transform solutions during a search: Swap and 2-Opt. The Swap performs a swap of two random stops in the route. The 2-Opt reverses a random segment of the route. For more information, see Coelho et al. (2016a).

3.4. Objective Function

We define a non-negative objective function \( F_h \), given in seconds with minimization direction, as follows:

\[
F_h = 1 \times F_{Travel} + 10 \times F_{TimeWindowEarly} + 86400 \times F_{TimeWindowLate} + 86400 \times F_{Backtracks}
\]

[^6]: https://github.com/nlohmann/json
[^3]: https://rapidjson.org
[^4]: https://github.com/igormcoelho/vastjson
[^5]: https://github.com/optframe/optframe
[^6]: https://github.com/optframe/pyoptframe
The constant 86400 is a strong penalization calculated as the number of seconds in a regular day $24 \times 3600 = 86400$, effectively punishing every violation with an extra day of travel. The package deliveries that arrived too early were considerably less penalized with a constant of 10, so the algorithm would prioritize minimizing the other parameters.

Each part of the objective function is described below:

- $F_{Travel}$: number of seconds spent in the route to perform all deliveries
- $F_{TimeWindowEarly}$: number of seconds in early delivery of packages (sum for all packages)
- $F_{TimeWindowLate}$: number of seconds in late delivery of packages (sum for all packages)
- $F_{Backtracks}$: for every revisited zone (backtrack) one second is accumulated

We also denote $F_h$ as the SA+ILS Objective Function.

### 3.5. Normalized Objective Function

We define a normalized $[0,1]$ version of objective function $F_h$, called $F_s$:

$$F_s = \begin{cases} 
1.0, & \text{if } F_h \geq 10,000,000 \\
\frac{F_h}{10,000,000}, & \text{otherwise}.
\end{cases} \quad (2)$$

The constant 10,000,000 defines the precision of $F_s$ as one tenth a microseconds scale of $F_h$. On practice, non-penalized routes (no late deliveries or backtracks) assume the simplified form:

$$F_s = 10^{-7} F_h \quad (3)$$

We also denote $F_s$ as the SA+ILS Score.

### 3.6. Amazon Score

We consider a function $F_A$ called Amazon Score, that gives real scoring of a solution (only used in experimentation phase, not production).

### 4. Results and Conclusions

We perform some experiments to determine the performance of proposed SA+ILS. From 6000 routes in model build input, we select the 2718 high routes to test our solution. The total time considered for the experiment was 4 hours in a 4-core AMD Ryzen 5 3400 GHz processor and 16 GB of principal memory.

#### 4.1. Model Build

No execution is performed during Model Build phase.
4.2. Model Apply

The SA+ILS metaheuristic is executed during the available time.

The Algorithm 5 describes the main proposal of our SA+ILS solution. We use $\rho$ to represent the given JSON data, and $\text{LoadData}()$ to represent the function we used to read the input data.

Algorithm 5 Proposed Solver

Start $G_{\text{TIME}}$;

TARGET\_TIME = $4 \times 3600 - 60$; \>

four hours minus one minute

$\text{AllRouteData} \leftarrow \text{LoadData}(\rho)$;

$\text{PROBLEM\_TOTAL} \leftarrow \#\rho$;

$\text{WORKING\_PROBLEMS} \leftarrow 0$;

PartialOutsuts[\text{NTHREADS}];

for $\text{thread\_id} \in 0..\text{NTHREADS} - 1$ do

$\text{PartialOutsuts[\text{thread\_id}] \leftarrow \emptyset}$;

end for

for parallel $r \in \text{AllRouteData}$ do

$\text{WORKING\_PROBLEMS} \leftarrow \text{WORKING\_PROBLEMS} + 1$;

Start $t_{\text{local}}$;

$\text{RouteData} \leftarrow r$;

$max\_time \leftarrow \min(\frac{\text{TARGET\_TIME} - G_{\text{TIME}}}{\text{PROBLEM\_TOTAL} - \text{WORKING\_PROBLEMS}}, \text{TARGET\_TIME} - G_{\text{TIME}})$;

$T_{\Phi} \leftarrow \text{estimateInitialTemperature}(f, N, s_{\text{gen}}, 1.1, 0.7, 500, 1)$;

$\text{BestRouteSA} \leftarrow \text{Run}(max\_time, \text{BasicSimulatedAnnealing}(f, N, s_{\text{gen}}, 0.98, 500, T_{\Phi}))$;

$max\_time2 \leftarrow max\_time - t_{\text{local}} - LIMIT$;

$\text{BestRouteILS} \leftarrow \text{Run}(max\_time2, \text{ILS\_VND}(f, N, \text{BestRouteSA}, 0, 10))$;

PartialOutsuts[\text{thread\_id}] \leftarrow \text{PartialOutsuts[\text{thread\_id}] u BestRouteILS}$;

end for

file(“proposed\_sequences.json”) $\leftarrow \bigcup_{out \in \text{PartialOutsuts}}$;

4.2.1. General Parameters

During local experiments, we had the following constraints: $\text{TARGET\_TIME} = 2 \times 60 = 120$, $\text{NTHREADS} = 4$, $\text{PROBLEM\_TOTAL} = 2718$.

For each $\text{thread\_id} \in [0, 1, ..., \text{NTHREADS} - 1]$, number of already solved problems $\text{WORKING\_PROBLEMS}$, and global timer $G_{\text{TIME}}$, we calculate $\text{max\_time} = (\text{TARGET\_TIME} - G_{\text{NOW}})/(\text{PROBLEM\_TOTAL} - \text{WORKING\_PROBLEMS})$. Finally, we ensure that: $\text{max\_time} = \min(\text{max\_time}, \text{TARGET\_TIME} - G_{\text{NOW}})$. $G_{\text{NOW}}$ is the current time. Each thread has a local timer $t_{\text{local}}$ to calculate: $\text{max\_time2} = \text{max\_time} - t_{\text{local}} - LIMIT$ (with $LIMIT = 3$).

For the instance parallelization, we use the well known library OpenMP. We pre-determine a dynamic
schedule for the threads with chunks of 10 in order to decrease the makespan between threads and OpenMP scheduler overhead (by setting the chunk).

The Simulated Annealing executes for $max_{time}$, while Iterated Local Search consumes $max_{time2}$.

4.3. Visualizing the progress of score during the search

We experimented with showing how the solutions generated by $SA+ILS$ are scored, thus comparing the proposed $F_s$ with the real score $F_A$. Due to the non-deterministic nature of $SA+ILS$ and $F_s$, many peaks and oscillations within the interval $[0,1]$ are expected, although we expect to have some correlation between both score metrics (excluding peaks and outliers).

Figure 1d illustrates the progress of $SA+ILS$ over RouteID_00143bdd-0a6b-49ecbb3536593d303e77, considering metrics Amazon Score $F_A$ (as solid gray) and $SA+ILS$ Score $F_s$ (as dashed black), considering 4101 iterations of $SA+ILS$.

From Figure 1d, we notice that $SA+ILS$ is able to minimize $F_s$ quickly in the first iterations, reducing the score from 1.0 to 0.4, when due to diminishing returns, it struggles to reduce it further. It is worth mentioning that, while executing SA, we calculate the real Amazon Score only every 100 iterations, due to the high computational costs of calculating it causing SA iterations to get much slower. In this case, the $SA+ILS$ method did not effectively reach the ILS within the timelimit (due to practical experimentation constraints), as it is already stagnated in the given test instance and quite far from SA freezing/stopping temperature.

We decided to run this visualization experiment on a strict 5 minutes window — the rough equivalent of a 10 seconds production iteration — as a means to help understanding the progress made by the heuristics with a limited access to resources. In a 120 seconds production settings, the $SA+ILS$ method is able to achieve Amazon Score $F_A$ of 0.03. In this best case scenario, $F_s = 0.2$ (against $F_A = 0.03$), demonstrating that correlation with Amazon Score is far from perfect.

4.4. Source code analysis

The final code is incredibly small, with only 600 lines of C/C++ for the declaration of problem-specific OptFrame components and the processing of problem data, plus 200 C/C++ lines for the instantiation of the solver and management of computing time/multi-thread requirements.

4.5. Results and Conclusions on Preliminary Tests

We have considered 2718 routes with High rating, and the $SA+ILS$ approach scored 0.09, indicating that approach is promising, although still far from target score 0.00. Figure 1b shows the graphic plot of all tested instances. More precisely, the mean score was 0.09212, the maximum value was 0.42375, the minimum value was 0.01077, and the standard deviation was 0.04636. Figure 1c represents the histogram of how the scores were distributed, with peak on near-optimal routes with score slightly worse than 0.09.
The results on Figure 1a suggest the worst performing routes were the ones with a smaller number of stops and shortest total time elapsed, which also have a larger mean number of packages per stop. The worst results in the routes with a large mean packages per stop may be explained by tighter constraints on packages due to time windows.

For the future, we believe that the introduction of advanced data structures could strongly reduce computational impact of a high number of packages, improving the results. The method also can benefit from further calibration of parameters, so as introducing novel neighborhood structures from TSP literature.
(a) Boxplot showing the range of scores against total delivery time, route size (number of stops), and mean number of packages per stop. Smaller routes with a large number of packages per stop achieved the worst performance.

(b) Scores for all 2718 routes.

(c) Histogram of route scores.

(d) Comparison of the Normalized Objective Function with the Amazon score during the execution of the heuristic.
References


Article XXIV

A Frequency-Based Routing and Scheduling Engine

António Ramos and Manuel Lopes

Source code is available on GitHub:
https://github.com/agrisep/MEGI.git
A frequency based routing and scheduling engine model

António Ramos, Manuel Lopes
School of Engineering, Polytechnic of Porto
R. Dr. António Bernardino de Almeida 431, Porto, Portugal

Abstract
In this work we present an engine for the generation of zone_ids sequences as the core element of our approach. The engine is driven by information gathered in the model build phase of the Challenge. Following the generation of the sequence of zone_ids the proposed sequence for each route is obtained applying a nearest neighbor heuristic. The approach was tested against 2718 routes provided for training that have a high route score. The submission score obtained for the set of instances was 0.04213.

1. Introduction
The approach to address the proposed problem followed can be divided into two phases. The first phase analyses the training instances and sets the foundations needed to address the problem. The main tasks involved in this phase are:

• create a data structure that integrates the data from all the input files. Due to the records related to each route is segregated in multiple files, the first step is to aggregate the data into a single data structure that contains all relevant data.

• compute a set of metrics for the route sequence solutions. Since the available data is limited to a selection of fields, it is expected that relationships exist, or should arise between the fields and within each field.

• create test instances for the model apply phase from the data provided. The test instances were built using the data provided and filtering the routes with "high" route_score.

• implement an algorithm for the TSP with the objective of minimizing the total travel time and respecting the delivery windows and service time. The purpose of the algorithm is to serve as a benchmark for the proposed approach solutions.
• visualize the routes data on a map.

At the end of this exploratory phase we gain an understanding of the fields in the data provided and we acquired a deeper knowledge of the problem.

The second phase builds on the outcomes of the first phase. From the map data visualization analysis it was identified the zone_id field as the pivot element in the route sequencing determination. Our approach is based on the assumptions that resulted from the analysis undertaken:

• the zone_id field is a postcode system with different levels of geographical aggregation,
• the coding system guides the sequencing of the routes,
• there is no universal pattern to determine the sequencing of the routes based on the zone_id field,
• the time windows constraints disrupt the zone_id driven sequencing of the routes.

One of the key aspects that conditioned our approach was the available processing time. It was clear from the beginning that the time available in the model apply phase for each instance would be very short.

2. Literature Review

City logistics involves the logistics and transportation activities in urban areas (Dolati Neghabadi et al., 2019). The last-mile delivery refers to the final stages in logistics networks and includes all those logistics activities related to the distribution of shipments to private customers in urban areas (Boysen et al., 2020).

Estimates indicate that in 2050 70% of the world’s population is living in major urban areas (megacities) (Bretzke, 2013), which, associated with the strong growth of e-commerce (Keenan, 2021), translates into a significant increase in the volume of orders to be delivered to the places of choice of customers. This is a trend that presents important challenges in several areas: Sustainability, Costs, Time pressure, and aging workforce (Boysen et al., 2020).

The urban congestion and its negative impacts on the environment, health, and safety, caused by the increasing level of parcel deliveries, will drive major changes in governmental legislation, which will induce carriers to adopt sustainable and environment-friendly operations (Hu et al., 2019). Traffic jams are an important cost driver of home parcel delivery (Boysen et al., 2020; Janjevic & Winkenbach, 2020) as well as first-time delivery failures (ranging from 12% to 60%) (Song et al., 2009). The customer expectations of free or low delivery prices and the promises of fast delivery services, next or same day delivery, will further increase the pressure to reduce service time and costs in a sector that already suffers from marginal profits (1-2%) (Bates et al., 2018; Yaman et al., 2012). The low payments environment that characterizes the last mile delivery sector and the observed aging of the workforce in developing economies (Otto et al., 2017), demands alternative and novel parcel delivery concepts based on automation (Boysen et al., 2020).

The setup and operation of last mile delivery systems involve the following strategic, tactical, and operational decision problems (Boysen et al., 2020):
• Design of the infrastructure,
• staffing and fleet sizing, and
• routing and scheduling.

The setup of the facility infrastructure is a very challenging problem and is related to the location and capacity planning of storage facilities to tackle the last mile parcel delivery. Stricter consolidation and coordination of individual orders into a single fleet and facility infrastructure lead to efficiency improvements in urban freight distribution (Winkenbach et al., 2016). Territory design for last mile delivery should balance the daily workload among regions with stochastic customer demands (Haugland et al., 2007; Lei et al., 2012). Most service providers opt to partition the geographic area of a distribution center in fixed smaller regions, which are serviced by a daily route, tuned to the set of customer orders to be delivered in each particular day. The routing planning process is either manual, based on specific knowledge of the driver, or supported on computer systems (Boysen et al., 2020).

Staffing and fleet sizing deals with the workforce planning of delivery persons and fleet sizing of delivery vans. Demand peaks, balance varying experience levels, absenteeism and other short-term labor-related decision problems make the assignment of drivers to regions and their daily tours one of the most important decision problems and challenging fields for research (Boysen et al., 2020; Hiermann et al., 2016; Pelletier et al., 2016).

The routing problem to be solved in the last mile context is the asymmetric Traveling Salesman Problem (aTSP). The asymmetric distances are due typically to the habitual one-way urban streets, and the no-left-turn policies used by carriers to reduce the risk of accidents and the travel times and emissions caused by the time waiting to cross the traffic flow (Fernandes et al., 2017). However, the aTSP does not address some relevant aspects of the daily delivery of parcels in urban areas (Boysen et al., 2020):

• Time windows, e.g., specific time intervals agreed with customers, commercial customers with limited opening hours, or restricted access to pedestrian streets. Exact and heuristic recent contributions for the aTSP with time windows can be found, for instance, in Baldacci et al. (2012); Da Silva & Urrutia (2010).

• Time-dependent travel times, e.g., typical morning/evening pendular movement of the traffic flow of urban areas. Some recent contributions are provided, for instance, in Cordeau et al. (2014); Arigliano et al. (2019).

• Time-dependent service times: the time needed to deliver the parcel to its final destination after parking the van. These times can vary according to, for example, additional waiting time during peak hours in commercial customers, or extra time for finding an alternative delivery option for private customers not at home. For instance, a recent contribution to the aTSP with time-dependent service times is Tas et al. (2016).
• Electric vehicles: because of the limited ranges the need for recharging have to be considered into routing problems when an electrified van fleet is used. A recent survey on this topic is provided by Erdelić & Carić (2019).

• Pedestrian subtours: delivery person walk distance from the parking space to the customer’s address. According to the empirical study of Allen et al. (2018) an average delivery van remains stationary for more than 60% of daily tour times and delivery persons walk up to 12 km on foot. A recent work on TSP with time windows and pedestrian subtours is Nguyễn et al. (2019).

Though the high volume of scientific literature published on routing optimization, specific driver knowledge about important variables of last mile parcel delivery, such as parking spaces, time dependent congestion, and customer preferences, is not appropriately modeled by commercial routing and scheduling software. The lack of solution frameworks that integrate all the above requirements is considered to be a main difficulty for the application of optimization-based routing (Allen et al., 2018; Nguyễn et al., 2019). The practice shows that the region-specific routing decisions of most last-mile service providers are still made by drivers (Snoeck et al., 2020). This problem is referred in Snoeck et al. (2020) as the route deviation problem.

The difference between the driver’s preferred route and the planned route causes many inefficiencies such as an out of order access to items due to a truck load sequence based on the planned route or the lack of the overall robustness of the optimization-based routing systems (Snoeck et al., 2020).

Today’s availability of large real-time data sets is an open window to the use of Machine Learning algorithms which can identify the driver’s routing patterns without a priori knowledge of the underlying system, and, therefore, contribute to close the gap between route planning and execution (Snoeck et al., 2020).

3. Methodology

The assumption that resulted from the map data visualization that the zone_id field is a postal code that is at the core of the sequencing of the route led to a preliminary examination of the zone_id field. Figure 1 illustrates the visualization of a route. It can be observed that each stop is connected to other stops with identical zone_id with the exception of two stops for entering and exiting the zone_id that connects with zone_ids with a very similar value.
We segmented the zone_id field by using the non-alphanumeric characters as delimiters and the type of alphanumeric character. That resulted in four new fields that we designated as Z1, Z2, Z3 and Z4. These new fields are referred hereafter as Z-segments and are illustrated in Figure 2.

To determine the hierarchy of geographical aggregation of the Z-segments we made a frequency analysis to the change of each of the Z-segments value between two consecutive stops. We used all the provided routes for the model build phase. The results are presented in Figure 3. As can be observed the higher level of aggregation is Z1, followed by Z2, Z4 and Z3.
A frequency analysis of the number of Z-segments that change between two consecutive stops was also performed (Figure 4). As can be observed, there is no change in any of the Z-segments between two consecutive stops in approximate 87% of the times, and when a change occurs, it is primarily in a single Z-segment. Together they account for 98.5% of the values. The results support our assumption that the zone_id field is a pivot field in the route sequencing.
In Figure 5 we observe for a section of the route illustrated in Figure 1 how the Z-segments change each time there is a change of the Z-segments between stops.

<table>
<thead>
<tr>
<th>Z1</th>
<th>Z2</th>
<th>Z3</th>
<th>Z4</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>7</td>
<td>2</td>
<td>E</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>1</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>1</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>2</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>3</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>3</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>2</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>1</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>1</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>2</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>3</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>3</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>2</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>1</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>1</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>2</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>3</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>3</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>2</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>1</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>1</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>2</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>3</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>2</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>1</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>2</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>3</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>3</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>3</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>1</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>1</td>
<td>▼</td>
</tr>
<tr>
<td>G</td>
<td>5</td>
<td>2</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>5</td>
<td>3</td>
<td>▲</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>3</td>
<td>▲</td>
</tr>
</tbody>
</table>

Figure 5: Z-segments changes for a segment of route RouteID:024a44ee-5a90-48d3-8f80-d8e7b124971e

Based on these results, we developed a zone_id sequencing engine, that is, a mechanism that given a zone_id generates a zone_id sequence. The engine is illustrated in Figure 6. The Z-segment with the highest level of aggregation Z1 uses a circular motion pattern while the remaining Z-segments follow a pendulum motion pattern, i.e, for a given list of z-segments, the circular motion pattern iterates always in a single direction (either forward or backwards) and when the end (or front) of the list is reached returns to the front (or end) of the list, while in a pendulum motion when the limit of the z-segment list is reached it reverses the iteration direction.
The overall idea of the engine is the following, the zone_id sequence is generated by moving one unit of a Z-segment at a time. Each time a Z-segment performs a full displacement in a given direction, the next upper-level Z-segment moves one unit in a given direction. It then returns to the lower level Z-segment and, by moving a unit at a time, performs a full displacement in the opposite direction.

Our approach is designed around the zone_id sequencing engine. The model apply framework is represented in Figure 7.
The zone_id engine requires as inputs the Z-segments hierarchy, the direction of displacement for each of the Z-segments and the zone_id’s of the route. The Z-segments hierarchy and the direction of displacement for the Z-segments for a given station are obtained during the model build phase.

The direction of displacement for the Z-segments is obtained by an analysis of frequency done for each of the stations of the direction of displacement of the sequence of the Z-segments. An overall Z-segments frequency analysis is also performed to be used if stations in the model apply phase are not present in the model build phase.

The zone_id sequence engine algorithm is illustrated in Algorithm 1.

The output of the Zone ID sequencing procedure is a sequence of zone_id’s. Within each zone_id the stops are sequenced using the Nearest (in time) Neighbor heuristic. The first stop of the sequence is determined by the stop in the first zone_id with the shortest time to the station. The first stop in each zone_id is the one with the shortest time to the last stop of the previous zone_id.

Following the stop’s sequencing, a set of measures are calculated to evaluate the time windows constraints.

In case some time window constraints are violated, the time windows heuristic is applied. This heuristic was not implemented and tested due to time constraints. However, since it was part of our strategy, it is
briefly described. The overall idea was to first try to fix the time window violation within the zone_id by swapping the order of stops, and if not possible swap zone_ids so to minimize the number of Z-segments changes while fixing the time windows violations.

The output of the Stops Sequencing procedure is the proposed route sequence.

4. Results and Conclusions

The approach was evaluated against a set of instances build by selecting from the supplied data the routes with ”high” route score. The number of instances used was 2718. The submission score obtained was 0.04213. This result is in line with the best results obtained in the preliminary submission of May 15 and considerably better than the results obtained with our benchmark TSP algorithm of 0.14.

Figure 8 presents the frequency of the scores obtained for the routes in the instance.

![Figure 8: Route score frequency](image)

We were unable to analyze these results during the project development. Future work is to implement and validate the time windows heuristic and improve the learning process for the elements required to guide the zone_id engine.

*Ramos and Lopes*
Appendix A. Algorithms pseudocode

The values for $Z_3 CP$ and $Z_4 CP$ are based on external configuration (based on what was found in the frequency analysis).

**Algorithm 1** Zone Sequencing based on Frequency Analysis

```plaintext
1: procedure Zone ID Hierarchical Generation($Z_1, Z_2, Z_3, ..., Z_n$)
2:   for Each $Z_1$ in CodesTree sorted('asc') do ▷ Lock $Z_1$ sequence as ascending
3:     for Each $Z_2$ in CodesTree sorted('asc') do ▷ For each $Z_2$ pair find $Z_3$ connecting codes
4:         Define $Z_{2p0}$ as current $Z_2$, and $Z_{2p1}$ as next $Z_2$
5:         Find and define $\min Z_{3p0}$ as minimum code $Z_3$ in $Z_{2p0}$ ▷ Find minimum $Z_3$ pair inside $Z_2$
6:         Find and define $\min Z_{3p1}$ as minimum code $Z_3$ in $Z_{2p1}$
7:         Find and define $\max Z_{3p0}$ as maximum code $Z_3$ in $Z_{2p0}$ ▷ Find maximum $Z_3$ pair inside $Z_2$
8:         Find and define $\max Z_{3p1}$ as maximum code $Z_3$ in $Z_{2p1}$
9:         Define $Z_{2p0}$ sublist and $Z_{2p1}$ sublist as ascending ▷ Default sequence of $Z_2$ pair
10:        if $\min Z_{1p0}$ is equal to $\min Z_{1p1}$ then ▷ If minimum $Z_3$ inside $Z_2$ pair is identical
11:            Define $Z_{2p0}$ sublist as descending ▷ $Z_{2p0}$ sublist ($Z_3$, etc) is descending
12:            Define $Z_{2p1}$ sublist as ascending ▷ $Z_{2p1}$ sublist ($Z_3$, etc) is ascending
13:        end if
14:        if $\max Z_{1p0}$ is equal to $\max Z_{1p1}$ then ▷ If maximum $Z_3$ inside $Z_2$ pair is identical
15:            Define $Z_{2p0}$ sublist as ascending ▷ $Z_{2p0}$ sublist ($Z_3$, etc) is ascending
16:            Define $Z_{2p1}$ sublist as descending ▷ $Z_{2p1}$ sublist ($Z_3$, etc) is descending
17:        end if
18:   end for
19:   for Each $Z_2$ in CodesTree sorted('asc') do
20:     for Each $Z_3$ in CodesTree sorted(REVERSE = $Z_3 CP$) do ▷ Sorting by external $Z_3 CP$
21:        Define $Z_3$ sublist as ascending if even, descending if odd
22:     end for
23:   end for
24:   for Each $Z_3$ in CodesTree sorted(REVERSE = $Z_2$ sublist ) do ▷ Sorting by external $Z_4 CP$
25:     for Each $Z_4$ in CodesTree sorted(REVERSE = $Z_4 CP$) do ▷ Sorting by external $Z_4 CP$
26:        Append to $Zone ID_{Seq}$ as $Z_1 - Z_2 Z_4 Z_3$
27:     end for
28:   end for
29: end for
30: Execute $Z_1$ Sorting Algorithm
31: end procedure
```
Algorithm 2 $Z_1$ Sorting based on Frequency Analysis

1: procedure $Z_1$ Sorting Algorithm($Z_1$ sorting)  

2: for Each $Z_1$ in CodesTree sorted('asc') do  

3: 

4: for Each $Z_s$ in $ZoneID_{Seq}$ do  

5: 

6: end for  

7: Define reverseSeq as False  

8: if $Z_1$ in $Sorting_{Z1}$ then  

9: Define $l_z_1$ as $Sorting_{Z1}.iniLesser$  

10: Define $g_z_1$ as $Sorting_{Z1}.iniGreater$  

11: if $g_z_1 > l_z_1$ then  

12: if $ZoneID_{Seq}.Z_2$ in position($first_{Z1}idx$) > $ZoneID_{Seq}.Z_2$ in position($last_{Z1}idx$) then  

13: Define reverseSeq as False  

14: else  

15: Define reverseSeq as True  

16: end if  

17: else  

18: if $ZoneID_{Seq}.Z_2$ in position($first_{Z1}idx$) > $ZoneID_{Seq}.Z_2$ in position($last_{Z1}idx$) then  

19: Define reverseSeq as True  

20: else  

21: Define reverseSeq as False  

22: end if  

23: end if  

24: end if  

25: for Each $ZoneID_{Seq}$ sorted(REVERSE = reverseSeq) do  

26: Append to CompleteZoneIDSeq as $Z_1 − Z_2.Z_4Z_3$  

27: end for  

28: end for  

29: Execute Node $ZoneID$ Correction Algorithm  

30: end procedure

ExternalCodeFormat is an external configurations (based on what was found in the frequency analysis, and defines the code structure such as − between the first and second sub-code and . between the second and the third sub-code).
Algorithm 3 ZoneID Correction

1: procedure Node ZoneID Correction Algorithm(Node ZoneID Correction)
2:  for Each $R_1$ in Routes do
3:      for Each Node$_1$ in $R_1$ do
4:         if CodeFormat(ZoneID) is not equal to ExternalCodeFormat then
5:             Find Closest Node$_C$ to Node$_1$ in $R_1$
6:             Replace Node$_1$.ZoneID with Node$_C$.ZoneID
7:      end if
8:  end for
9: end for
10: end procedure

Appendix B. Route Metrics

Metrics computed for each node in the route sequence, and between every pair of consecutive nodes.

1. travel time to node, from the previous node
2. arrival time
3. waiting time
4. time from arrival to time-window opening
5. total node service time for packages with time-windows
6. total node service time for packages without time-windows
7. time from departure to time-window closing
8. departure time,
9. delay time (time between time-window close and departure)

The accumulated counterparts are also computed (by adding for each field the data since the beginning of the route up to the current node). The last node contains the total accumulated data for each field.

The computed accumulated fields are the following:

1. accumulated travel time to node
2. accumulated waiting time
3. accumulated time from arrival to time-window opening
4. accumulated service time for packages with time-windows
5. accumulated service time for packages without time-windows
6. accumulated time from departure to time-window closing
7. accumulated delay time (time between time-window close and departure)

Other compiled data consists of the frequency of the occurrence of certain events, for all routes, such as:
1. the number of times a connection is made for each pair of zone IDs (and the departure / arrival timestamps to check for patterns related to traffic, etc)

2. the minimum, maximum and average travel time for each pair of zone IDs that have been connected

3. the number of times a connection is made leaving a station and arriving the first zone ID

4. the number of times a connection is made leaving the last zone ID and arriving to a station

5. the number of times a connection is made leaving a station and arriving the first node

6. the number of times a connection is made leaving the last node and arriving to a station

7. the association between the zone code and node coordinates

8. the association between time-windows and node (common overlap)

Other data that is also verified is the codification of the route ID, zone ID and coordinates, searching for repeating patterns along the route sequence, for every node:

1. the decomposition of route ID code into sub-codes that may imply a pattern related to the sequence of nodes/zones

2. the decomposition of zone ID code into sub-codes that may imply a pattern related to the sequence of nodes/zones

3. the decomposition of node coordinates into sub-groups that may imply a pattern related to the sequence of nodes/zones

where the frequency of change during the route is observed, and if the code is increasing/decreasing, and the frequency of the every sequential increase/decrease. This allows to detect if there is a base encoding structure, such as an alphabet used to represent a sequence, like the structure used to specify hexadecimal base notation (0 to 9 followed by a to f) using 16 characters. The frequency of change allows to determine the significant sub-codes, and the sequential changes determine the sequence inside each sub-code.

These statistics are compiled into two files, StationLinksTree.json and ModelConfigStats.json, that are used in the next phase.

The second phase computes the sequence of a route considering the calibrated parameters and patterns observed during the first phase. If no relevant patterns have been observed, then the route sequence is generated using a custom made variable neighborhood search (VNS) algorithm that considers the time-window, service time, and travel time constraints.

If any pattern is identified in the preferential connections between zone IDs or nodes, then the initial route sequence is generated considering that data. There is an extra step that tries to adjust the sequence if the time-windows are not compatible.

If any pattern is identified in the sub-codes of the sequence of zone IDs, then the initial route sequence is generated considering that data.
References


2021 Last-Mile Routing Research Challenge
Technical Proceedings

Article XXV

Learning to Solve the Vehicle Routing Problem

Xinyang Yuan and Ji Gao

Source code is available on GitHub:
https://github.com/xinyangyuan/routing
Article XXV

Learning to Solve the Vehicle Routing Problem

Xinyang Yuan
Department of Computing, Imperial College London
Exhibition Rd, South Kensington, London, UK

Ji Gao
Mathematics and Applications, Institut Polytechnique de Paris
Route de Saclay, Palaiseau, France

Abstract

The vehicle routing problem (VRP) is a classical combinatorial optimization problem. The problem is of critical industrial importance as it determines logistic efficiency; however, it is NP-difficult to find the optimal solution to the problem. This paper introduces a learning-based system that solves VRP as a 2D graph, in which a convolutional neural network, named RouteNet, is trained to make accurate predictions of stop-wise transition probability for stop pairs, which effectively transform the input 2D Euclidean graph into a meta transition graph that conveys information such as time of the day, or region in the city. The network adapts to route with different lengths and make predictions in a non-autoregressive manner. Route generation can be done with greedy or beam search. Finally, we find that the network output meta graph can be used as a standalone problem graph, so the network can be pipelined with any operation research solvers to form the final solution system. An average test score of 0.08 was achieved for the internal testing data.

1. Introduction

The VRP problem is traditionally solved using mixed-integer linear or nonlinear programming (MILP or MINLP), in which the problem graph is specified with an edge adjacency matrix along with constraints such as time window, vehicle capacity, and pick-up and delivery nodes.

The solution model then iteratively finds the optimum solution, which the individual searches can perform randomly or using problem-specific heuristics against the fitness metrics. The optimum route output by the model is the one that minimizes the total vehicle travel distance.

However, the real-world VRP problem involves more information that is not captured by the Euclidean adjacency matrix and constraints. Hidden information, e.g., traffic conditions at different times of the day,
the effects of weather or holidays, directly change the optimal solution. This hidden information is well known by the driver, but it is challenging to incorporate it into the optimization routine of traditional solvers.

Consequently, learning-based models become good candidates for real-world VRP. In our design specification, the learned model should capture both hidden route-specific features and VRP constraints (including time-window).

To formulate the VRP into a supervised learning problem, we want to maintain the graph inductive bias, so the proposed RouteNet model uses CNN architecture and the model input has the form of a 2D graph (or image). The model only uses operations that maintain input graph spatial dimension (height and width), so the model automatically scales to routes with variable number of stops and can be batch-trained through padding and masking. Lastly, model output is a 2D matrix that can be interpreted as a meta graph (or heatmap) that predicts transition probability or edge probability among stop-pairs.

The output meta graph capture full information of the VRP. If the RouteNet is used in an end-to-end manner, valid travel sequences can be directly obtained through greedy or beam search decoding of model output. However, the real benefit of maintaining graph representation is to use the meta graph as a new VRP problem, in which its Euclidean adjacency matrix already embeds hidden feature information.

The new VRP problem can then be solved with traditional solvers without any additional constraints. This full pipeline system allows the smooth incorporation of hidden information into the solution routine, and easy augmentation to existing VRP solvers.

The RouteNet makes predictions non-autoregressively, so it is highly parallelizable and scales linearly to problem size. Also, the new meta graph VRP is relaxed without any constraints, so traditional solvers can obtain the optimum solution in a shorter time window.

2. Literature Review

The vehicle routing problem (VRP), first formulated as a truck dispatching problem by Dantzig & Ramser (1959), is considered as a generalization of the classic traveling salesman problem (TSP). Both of them are well-studied combinatorial optimization problems, which families of algorithms addressing them have been extensively studied and developed.

Finding the optimal solution to VRP is NP-difficult, so the exact algorithmic solvers all have restrictions on problem dimensions. For real-world applications, approximate solvers are often used; and near all state-of-the-art solvers use handcrafted heuristics to guide solution search. Solvers that implement LinKernighan (Lin & Kernighan 1973) or newer Lin-Kernighan-Helsgaun (Helsgaun 2000) heuristics are shown to solve the symmetric TSP with hundreds of stops to optimality. Thus, algorithm performance heavily relies on the fitness of heuristics to the problem, and often we cannot use the same optimized heuristics for a different set of problems even if they have only differed slightly. Some works attempt to generalize these heuristics to larger problem space through hyper-or-meta heuristics (Burke et al. 2013).
Earlier neural models for combinatorial optimization problems do not involve learning from samples. Two prominent early attempts are (1) Hopfield networks that formulate the optimization objective into the networks’ energy function \cite{Hopfield1985}; and (2) deformable template models such as elastic nets \cite{Durbin1987}.

With the rise of deep learning, neural models that learn through error gradient back-propagation have become the standard. Motivated by the advances in sequence to sequence neural translation task, Vinyals et al. \cite{Vinyals2017} proposed Pointer Network (PtrNet) to solve combinatorial problem. The PtrNet embed route information through the RNN encoder, and uses a modified attention mechanism by Bahdanau et al. \cite{Bahdanau2016} to output permutation assignment in the RNN decoder. The model is trained fully in a supervised setting, in which the model learns in an autoregressive manner with teacher forcing. The PtrNet directly outputs stop-pair transition probabilities in linear time, and the final solution can be obtained using greedy or beam search. Bello et al. \cite{Bello2017} and Kool et al. \cite{Kool2019} builds upon the PtrNet, which the model is trained in reinforcement setting using Actor-Critic and REINFORCE.

The sequence-to-sequence formulation lends the task of learning the nodes Euclidean graph fully to the network. On the other hand, Dai et al. \cite{Dai2018} and Joshi et al. \cite{Joshi2019} better preserve the problem combinatorial nature by formulating the problem as graph using graph neural network. Then, the model can be trained in either supervised or reinforcement setting.

We design our model, RouteNet, using convolutional neural network topology, which is highly parallelizable and preserves the VRP graph nature. The VRP problem graph is encoded into a 2D image, in which the image channels contain inter-stop travel time matrix, and stop-specific features. The RouteNet has a backbone of Router blocks. Each Router block consists of a self-attention module \cite{Vaswani2017, Zhang2019, Huang2020} and a convolution module, along with residual connections. RouteNet directly output all stop-pair transition probabilities (i.e. a 2D matrix) in a single step. The model can be batch trained with supervised labels, and it is sample efficient because it does not need many routes to generalize the inter-stop relationship. Similarly, route solution can be obtained using greedy or beam search.

Lastly, the output stop transition or edge probability matrix can be interpreted as a standalone VRP graph, that is obtained by transforming the original one. The new meta graph captures full information of the VRP, including both hidden feature (e.g. traffic conditions at different times of the day) information and constraints (e.g. delivery time-window). Thus, we use Google’s vehicle routing problem solver with guided local search metaheuristic to obtain the final solution using new meta graph \cite{Perron2019}.

3. Methodology

3.1. Data processing

All route, stop, and package specific information are used for training. The input features are preprocessed into three different input tensors: input\_0d, input\_1d, and input\_2d. Input\_0d are route-level features
that do not vary with the number of stops, including departure date, time, whether it is weekend or holiday, capacity and route score. **Input**. 

1. **Input.** 

1d are features that scale linearly with the number of stops, including stop-specific features (e.g. latitude, longitude, zone) and stop’s package features (e.g. number of delivery package, total package volume). **Input.** 

2d are stop-pair features that scale quadratically with the number of stops, and they include time-window, latitude difference, longitude difference, and Euclidean distance matrices. The final inputs to the RouteNet contain two tensors: (1) **input tensor** that formed by concatenating **input.** and broadcasted **input.** through channel axis, and (2) the **input.** tensor.

To perform batch training, the batches are bucketed to routes with similar length and shorter **input.** is zero-padded. Random permutation of stops in the **input.** is also carried out, which serves as data augmentation that allows the model to learn the permutation invariance, thus generalize better.

3.2. RouteNet

RouteNet is the core component of the whole system, and its overall structure is illustrated by figure 1. The input fusion block merges **input.** and **input.** tensors, and scales up the channel dimension to be the same as Router embedding dimension. The merge of **input.** into the **input.** is done in three steps: (1) the **input.** is fed into fully connected layers that have dimension of \(N \times C\), which \(N\) is a hyperparameter and \(C\) is the number of channels of **input.**; then (2) \(N\) kernels are created by truncating the fully connected layer output; finally, (3) convolve these kernels on the **input.** tensor. If \(N\) is less than the Router embedding dimension, the remaining channels are obtained through normal conv2d layer.

![Figure 1: Diagram of the overall RouteNet modules.](image-url)

A series of Routers form the backbone of the RouteNet, and its internal layers used in single Router block is illustrated by figure 2. The block uses pre-norm strategy proposed by [Nguyen & Salazar, 2019] and uses group normalization [Wu & He, 2018]. The multi-head criss-cross self attention layer is built upon [Huang et al., 2020], which each edge \(x_{ij}\) is only allowed to attend on its neighbouring edges, i.e., \(x_{il} \cup x_{mj}\) where \(l, m \in \{1, 2, ..., num\_of\_stops\}\).

The convolutional layers then perform feature abstraction and representation learning, and bypass residual connections inside the Router block enable gradients to pass back through the network undiminished, permitting the training of deep networks.

---

1 **Input** can be thought as a 2D image/graph with many channels.
3.3. Loss function

Given the known ground-truth VRP tour, we convert the tour into pairwise transition labels. For each route, there are $L$ number of transition predictions ($L$ is the number of stops in the route, for the final stop of the sequence it should predict the station as next stop), and the multi-class cross-entropy loss is used for training. To minimize the highly unbalance towards the negative classes, i.e., there is only a single succeeding stop (positive class) v.s. the number of route stops minus one negative classes, we add label smoothing on top of the cross-entropy loss.

3.4. Solution generation

If the RouteNet is used in end-to-end fashion, final route solution can be obtained using greedy or beam search of the model output.

To boost solution performance, we use RouteNet output as a new meta graph (distance matrix) for the VRP problem. The new meta graph captures full information of the VRP, including both hidden feature (e.g. traffic conditions at different times of the day) information and constraints (e.g. delivery time-window). Hence, the original time-window constrained VRP is relaxed to a VRP problem without time window constraints using the new meta graph. We then use Google’s vehicle routing problem solver with guided local search metaheuristic to obtain the final solution using new meta graph \[\text{Perron} & \text{Furnon}].

4. Test Results

Limited to the training time, the model is trained for all provided data for 1 epoch with batch size 2. During the tuning process for model training, fast convergence to plateau can be seen. The convergence speed is almost independent of data size. This unusual behavior may due to the mismatch of large model size and small number of epochs of training. If condition allows, more time and computation resources should
be used to train the model for more epochs for further analysis. The number of epochs for further testing should be in the range of 10 to 50, or more. In practice, sometimes significant increase in performance may be observed if high number of epochs is used for training. The bright side of this observation is that only a small amount of time and data are needed to obtain reasonable performance for this relatively large model.

A portion of the high quality routes (718 routes) randomly chosen from the data set are used for testing purposes. Distribution of the route score over the test data is shown in Figure 3 with mean score of 0.086 and median score of 0.074. The model output is scaled with the row-wise mean of the original distant matrix before feeding to the Google-Ortools VRP solver. The scaling process is closely related to the final score obtained. Final score can be improved for 20 percent subject to different scaling strategies. However, we could not identify a universal strategy to tune the scaling factors so far.

Figure 3: Distribution of route scores over test data

Figure 4 shows the comparison of route distribution of routes with top/bottom 30 percent of scores. The orange/blue plots represent the 30 percent of testing routes with highest/lowest scores. Their frequency is plotted against different features.

The routes are clustered based on their geographical location into 5 clusters (or in reality 5 states in the US). The first plot shows that the model made more predictions with high scores in cluster 3, 4 and 5, due to the lack of training data in those regions. Similarly, some qualitative observations can be deduced, that it is easier for the model to make good prediction when:

- there are more training data in the corresponding cluster;

- the departure time of the day is between 6 to 8 am;

- the delivery is made during weekend;

- the operating capacity is less than 65 percent of the total capacity;

- the total number of packages for the route is below 220;
- less delivery time window constraints are encountered.

Figure 4: Comparison of top/bottom 30 percent test data over different features

5. Conclusions

The RouteNet captures the information that are hard to formulate by the traditional VRP model, and relaxes time window constraints. It also allows parallelization and scales linearly to problem size. This work provides a way of integrating hidden and history information into the meta graph as an input to the downstream optimization process.
Listing 1: Pseudo code of input fusion in a PyTorch-like style.

```python
class InputFusion(nn.Module):
    def __init__(self, in_dim, in_dim_0, router_embbed_dim, aux_embbed_dim):
        self.input = nn.Conv2d(in_dim, router_embbed_dim - aux_embbed_dim, kernel_size=1)
        self.input_0 = nn.Sequential(
            nn.Linear(in_dim_0, in_dim*aux_embbed_dim, bias=False),
            nn.SiLU(),
            nn.LayerNorm(in_dim*aux_embbed_dim),
            nn.Linear(in_dim*aux_embbed_dim, in_dim*aux_embbed_dim)
        )

    def forward(self, x, x_0):
        batch_m, C, height, width = x.size()  # height==width==n
        kernels = self.input_0(x_0)
        kernels = kernels.reshape(-1, self.channel_in, 1, 1)
        # (1) 2d-feature maps from input directly
        out = self.input(x)  # out_C = router_embbed_dim - aux_embbed_dim
        # (2) 2d-feature maps from aux-input using adaptive kernels (out_C = aux_embbed_dim)
        x = x.reshape(1, -1, height, width)
        out_0 = F.conv2d(x, kernels, groups=batch_m)
        out_0 = out_0.reshape(batch_m, self.channel_out_0, height, width)
        return torch.cat((out, out_0), 1)
```

Yuan and Gao
References


Perron, L., & Furnon, V. (). Or-tools. URL: https://developers.google.com/optimization/.


2021 Last-Mile Routing Research Challenge
Technical Proceedings

Article XXVI

Hybrid Adaptive Large Neighborhood Search for the Traveling Salesman Problem with Time Windows and Adjusted Costs

Stefan Voigt

Source code is available on GitHub:
https://github.com/HybridSteVo/HALNSamazon
Hybrid Adaptive Large Neighborhood Search for the Traveling Salesman Problem with Time Windows and Adjusted Costs

Stefan Voigt

Department of Supply Chain Management & Operations, Catholic University of Eichstätt-Ingolstadt
Auf der Schanz 49, 85049 Ingolstadt, Germany

Abstract

This article models the underlying problem of the Amazon/MIT last mile routing research challenge as Traveling Salesman Problem with Time Windows (TSPTW). It uses an efficient heuristic based on the recently proposed Hybrid Adaptive Large Neighborhood Search (HALNS) to solve the TSPTW. Furthermore, the approach leverages information on high-quality routes to adjust the traveling costs and therefore guide the HALNS to mimic the behavior of drivers. Thus, the overall approach follows a classical Operations Research paradigm without using sophisticated Machine Learning (ML) techniques. Nevertheless, the HALNS generates a population of individuals, which may be used as additional features in an ML approach. We achieve an average score of 0.0678 on test data consisting of all 2718 high-quality routes.

1. Introduction

The Amazon/MIT last mile routing research challenge (Amazon Challenge\(^1\)) covers a routing problem, where routes with few hundreds of customers have to be constructed route by route, i.e., independently. Some customers have time windows during which the delivery must be completed. Additionally, servicing a customer takes time. The problem is similar to a Traveling Salesman Problem with Time Windows (TSPTW). The Traveling Salesman Problem (TSP) is a routing problem where one salesman (delivery vehicle) is required to visit each location (customer) once, starting at and returning to the same location (depot). However, the goal of the Amazon Challenge differs from the goal of the classical TSP, as not only the traveling costs are to be minimized, but the route should be similar to a high-quality route realized by an actual driver. The driver may visit customers in a different sequence as in the route suggested, for a couple of reasons, like avoiding traffic jams, lack of parking spaces, more convenient roads, avoiding left-turns, and

\(^1\)Please refer to [https://github.com/MIT-CAVE/rc-cli](https://github.com/MIT-CAVE/rc-cli) for details on the scoring process and to access the data.
so on. The data provided by the Amazon Challenge can be leveraged to identify and include such strategies into constructing solutions. Considering the size of the problem with a few hundred customers and the structure of the problem, both Machine Learning (ML) and Operations Research (OR) approaches seem to be well-suited to the problem. This article however, follows a classical OR process, consisting of the phases problem definition, formulation of a mathematical model, development of a solution procedure, testing and refinement (see e.g., [Hillier & Lieberman (1995)]). It is the author’s belief that the performance of any ML approach used in the field of optimization must be compared to a state-of-the-art OR approach. The lack of an OR based benchmark is a shortcoming often found in ML literature applied to classical optimization problems, like the TSP (see Section 2).

The problem is modeled as TSPTW and solved with a metaheuristic solution approach based on the recently proposed Hybrid Adaptive Large Neighborhood Search (HALNS) by [Voigt et al. (2021)]. The HALNS generates a population of solutions with an efficient ALNS, which are then improved by a crossover phase. Before the HALNS is applied, the cost matrix (costs for traveling from node to node) is adjusted to capture some of the behavior of drivers. Additionally, we tested different strategies, e.g., violating time windows, and tried to predict which cost adjustment/strategy should be applied depending on instance features. Unfortunately, the prediction accuracy was not good enough. Henceforth, we simply use the cost adjustment/strategy with the lowest average score.

This article makes several contributions: (1) The approach suggested can serve as a baseline for comparing ML approaches. (2) Information gathered within the population may be included as additional features in ML approaches. (3) An ML model (with sufficient accuracy) may leverage the proposed strategies.

The remainder of this paper is structured as follows: Section 2 shows related literature. Section 3 models the problem as TSPTW, introduces the cost adjustments, and presents the HALNS. Section 4 experiments with different adjustments of the cost matrix and strategies drivers may follow. Finally, Section 5 concludes the work and indicates avenues for further research.

2. Literature Review

The TSP is one of the most studied \( \mathcal{NP} \)-hard optimization problems in the field of OR with many scientific papers and books published (e.g., [Applegate et al. (2007); Cook (2012)]). There exist many approaches to solve the TSP heuristically. A very well-performing heuristic is the implementation of the Lin-Kernighan heuristic by [Helsgaun (2000)] (LKH\( ^2 \)). Approaches for solving large-scale instances with millions of cities include POPMUSIC [Taillard & Helsgaun (2019)]. The method of choice for exactly solving TSPs is the Concorde solver, which is able to solve instances with thousands of cities [Applegate et al. (2006)]. Despite the \( \mathcal{NP} \)-hardness, both heuristic and exact approaches can solve instances with few hundreds of customers

\( ^2 \)http://akira.ruc.dk/~keld/research/LKH-3/

\( ^3 \)https://www.math.uwaterloo.ca/tsp/concorde.html
within reasonable runtime, as demanded in the *Amazon Challenge*. This review neglects the time window component since it can efficiently be integrated into heuristic approaches [Vidal et al. 2015].

Mainly in the last decade, researchers developed ML approaches for graph problems and use the TSP as testing ground for their models. ML approaches either construct the solution directly or use ML together with classical optimization techniques.

**ML combined with optimization techniques.** [Arnold et al. 2021] use ML within a metaheuristic solution approach to identify high-order moves and improve the performance of state-of-the-art heuristics for the Vehicle Routing Problem. For a general overview of ML at the service of metaheuristics please refer to [Karimi-Mamaghan et al. 2021]. Instead of integrating ML within the metaheuristic, [Hutter et al. 2014] use ML to predict the performance of algorithms depending on instance characteristics. This approach is similar to our idea of solving the TSPTW with different cost adjustments/strategies and trying to predict which strategy should be applied for solving the instance. Closely related is ML-based hyperparameter tuning, where an ML model tunes the parameters of heuristics depending on instance characteristics (e.g., *irace package* by [López-Ibáñez et al. 2016]).

**ML models constructing solutions directly.** There exists a variety of ML models that try to solve the TSP directly. [Bello et al. 2017] use neural networks with reinforcement learning and negative tour lengths as reward signal. They solve instances with up to 100 cities and use LKH, Google OR-Tools and Concorde as benchmark. They achieve near-optimal results but are outperformed by LKH, which finds optimal results in similar runtime. [Dai et al. 2018] propose a combination of reinforcement learning and deep graph embedding. The authors train their model with instances ranging from 50-100 cities and apply the trained model on instances with up to 1,200 cities achieving a rather high optimality gap. The approach is compared to simple heuristics. However, a benchmark against state-of-the-art heuristics is missing. [Kool et al. 2019] present a model with attention layers, solving instances with up to 100 cities and outperform simple heuristics. [Dwivedi et al. 2020] present a framework for benchmarking graph neural nets, showing that Gated-Graph Convolutional Networks achieve promising results for solving the TSP.

To summarize, models that directly construct solutions fall short compared to the performance reached by state-of-the-art heuristics/solvers and work only for instances with few hundreds of customers. Furthermore, most approaches generalize poorly on larger instances [Joshi et al. 2020] and have issues with additional constraints, e.g., time windows. It remains to be seen if these issues can be solved in the future - maybe the *Amazon Challenge* stimulates research in this direction. Right now, classical optimization techniques together with ML seem to be more suitable than ML models that directly construct solutions.

### 3. Methodology

Section 3.1 models the problem as TSPTW. Section 3.2 details strategies and cost adjustments. Section 3.3 presents the HALNS to solve the TSPTW with then adjusted costs. Lastly, Section 3.4 introduces the
data-driven approach for leveraging the given data.

### 3.1. Model for the Traveling Salesman Problem with Time Windows

The TSPTW can be defined on a directed graph $G(N, A)$ with node set $N$, consisting of the depot 0 and customers $j \in C$, and arc set $A$. The arc set is defined as $A = \{(i, j) : i \neq j, i, j \in N\}$. $c_{ij}$ denotes the associated transportation costs. The vehicle starts and ends at the depot. Customers must be visited within their time window defined by lower limit $e_j$ and upper limit $l_j$. The duration of a tour is determined by the traveling times denoted by $t_{ij}$, by the service time $S_j$, and eventually waiting times between customers. The duration may not exceed the delivery period’s length $D$. Table 1 summarizes the notation.

<table>
<thead>
<tr>
<th>Table 1 Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sets</strong></td>
</tr>
<tr>
<td>$C$ Set of customers, $C = {1, \ldots,</td>
</tr>
<tr>
<td>$N$ Set of nodes, $N = {0} \cup C = {0, \ldots,</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td>$c_{ij}$ Traveling cost from $i$ to $j$, $i, j \in N$</td>
</tr>
<tr>
<td>$t_{ij}$ Traveling time from $i$ to $j$, $i, j \in N$</td>
</tr>
<tr>
<td>$S_j$ Service duration at customer $j$, $j \in C$</td>
</tr>
<tr>
<td>$e_j$ Earliest start of service for customer $j$, $j \in C$</td>
</tr>
<tr>
<td>$l_j$ Latest start of service for customer $j$, $j \in C$</td>
</tr>
<tr>
<td>$D$ Length of delivery period</td>
</tr>
<tr>
<td><strong>Decision Variables</strong></td>
</tr>
<tr>
<td>$x_{ij}$ Binary variable indicating whether arc $(i, j)$ is used, $(i, j) \in A$</td>
</tr>
<tr>
<td>$s_i$ Start time of service at node $i$, $i \in N$</td>
</tr>
</tbody>
</table>

**Model TSPTW**

Minimize $C_{TSPTW} = \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}$  

s.t.

\[ \sum_{j \in N} x_{ij} = 1 \quad \forall i \in C \]  

\[ \sum_{j \in N} x_{ij} = \sum_{j \in N} x_{ji} \quad \forall i \in N \]  

\[ s_j + S_j + t_{j0} \leq D \quad \forall j \in C \]  

\[ s_j - s_i \geq (t_{ij} + S_i)x_{ij} - D(1 - x_{ij}) \quad \forall i, j \in C, i \neq j \]  

\[ e_j \leq s_j \leq l_j - S_j \quad \forall j \in C \]  

\[ x_{ij} \in \{0, 1\} \quad \forall i, j \in N \]  

\[ s_i \in \mathbb{R}_0^+ \quad \forall i \in N \]

The objective function (1) minimizes traveling costs. Constraints (2) ensure that each customer is vi-
ited exactly once. Constraints \([3]\) conserve flow. Constraints \([4]\) ensure that the vehicle returns in time. Constraints \([5]\) guarantee that the service starts after traveling time plus service time starting from its predecessor. Constraints \([6]\) ensure that service starts and ends within the time window. Please note, that this restriction is more restrictive than usual assumed, as the service must generally only start before the upper limit of the TW. Constraints \([7]\) and \([8]\) define the domains of the variables.

3.2. Strategies and adjustments of cost matrix

Before applying the HALNS for solving the TSPTW, the given traveling cost matrix is adjusted to mimic the behavior of the driver. This subsection details the strategies and cost adjustments. Please note, that traveling costs are originally equivalent to traveling times, i.e., \(c_{ij} = t_{ij} \quad \forall i, j \in N\). However, after adjusting the cost matrix, traveling costs will in general differ from traveling times.

Triangle inequality. This adjustment simply checks if the triangle inequality holds for all arcs. The triangle inequality holds, if it is impossible to achieve lower costs by traveling via another node, \(k\), instead of traveling directly from node \(i\) to \(j\). The traveling costs of arcs are adjusted until the triangle inequality holds for all arcs. Costs are adjusted by finding the shortest paths for all nodes as follows.

\[
c_{ij} = \begin{cases} 
  c_{ij}, & \text{if } c_{ij} \leq c_{ik} + c_{kj} \\
  c_{ik} + c_{kj}, & \text{else}
\end{cases}
\]

Unreasonable expensive arcs. This adjustment checks if the costs given for an arc are unreasonably high compared to the great-circle distance (gcd) of node \(i\) and node \(j\) with corresponding coordinates \(coord_i\) and \(coord_j\). Costs are adjusted as follows. The factor 1.3 is chosen after preliminary testing.

\[
c_{ij} = \begin{cases} 
  \min(c_{ij}, c_{ji}), & \text{if } c_{ij} > 1.3 \cdot \gcd(coord_i, coord_j) \\
  c_{ij}, & \text{else}
\end{cases}
\]

The reasoning behind this adjustment is that drivers may use the arc if the arc has a small gcd and the difference between the way from \(i\) to \(j\) and back (\(j\) to \(i\)) are different, as this may indicate an error in the data. The minimum of \(c_{ij}\) and \(c_{ji}\) can be interpreted as the best guess for correcting the data.

Depot arcs. The depot is usually located outside the city and therefore connected by a highway with the delivery area. Drivers may accept higher traveling costs (longer traveling time) on highways, if this means, that they can in turn reduce the distance traveled within the delivery area - assuming that the highway is more convenient to drive. Costs are adjusted as follows. The factor 0.1 is chosen after preliminary testing.

\[
c_{0j} = 0.1 \cdot c_{0j} \quad \forall j \in N
\]

\[
c_{j0} = 0.1 \cdot c_{j0} \quad \forall j \in N
\]

Time windows. The last strategy disregards all time windows, resulting in solving a classical TSP. Drivers may disregard time windows of some or of all customers, because they are for example short on time.
3.3. Hybrid Adaptive Large Neighborhood Search for solving TSPTW

This subsection briefly introduces the Hybrid Adaptive Large Neighborhood Search (HALNS) which is used to solve the TSPTW. The metaheuristic is derived from the recently proposed HALNS for the Vehicle Routing Problem with Availability Profiles. For a detailed explanation, please refer to Voigt et al. (2021). Algorithm 1 describes the HALNS. The HALNS generates an initial population, $P$ of $n_P$ solutions by iteratively applying an efficient ALNS (lines 1-3). Afterward, the HALNS executes crossovers within that population (line 6-9). During crossover, two individuals - one solution from the population, $s$ and the global best solution, $\hat{s}$ - are combined by using the ALNS, instead of using an explicit crossover operator. The global best solution guides the removal procedure. Customers are removed if their successors differ in both solutions, i.e., if they are placed differently in both solutions. After crossover, the procedure selects surviving individuals (line 10) according to objective value and diversity. The procedure ends after $gen_{\text{max}}$ generations.

### Algorithm 1: Hybrid Adaptive Large Neighborhood Search

```plaintext
1 while $|P| < n_P$ do // Initial Population
2     $s \leftarrow \text{ALNS}()$
3     $P \leftarrow P \cup \{s\}$
4 while $\text{gens} < gen_{\text{max}}$ do // Generations
5     $\hat{s} \leftarrow \text{DetermineBestSolution}(P)$
6     while $i < n_P$ do // Crossover and Education Phase
7         $s \leftarrow P[i]$
8         $s \leftarrow \text{ALNS}(s, \hat{s})$
9         $P \leftarrow P \cup \{s\}$
10    $P \leftarrow \text{DiversityManagement}(P)$ // Select survivors and manage diversity
```

### Algorithm 2: Adaptive Large Neighborhood Search used within HALNS

**Input:** Starting solution $s$, global best solution $\hat{s}$  
**Output:** best solution $s^*$

```plaintext
1 $s^* \leftarrow s$
2 while Iterations without improvement $< it_{\text{stop}}$ do
3     ChooseOperators() // Determine Removal Candidates
4     $C_R^d \leftarrow \text{getRemovalCandidates}(s, \hat{s})$
5     $(s_{\text{new}}, C_R) \leftarrow \text{Remove}(s, C_R^d)$ // Remove Customers
6     $C_R \leftarrow \text{sort}(C_R)$ // Determine Insertion Order
7     $s_{\text{new}} \leftarrow \text{Insert}(s_{\text{new}}, C_R)$ // Insert Customers
8     if $f(s_{\text{new}}) < f(s^*)$ then
9         if $f(s^*) < f(\hat{s})$ then
10            $\hat{s} \leftarrow s^*$
11        else if $\text{accept}(f(s_{\text{new}}), f(s^*), \tau)$ then
12            $s \leftarrow s_{\text{new}}$ // Simulated Annealing
13    $\tau \leftarrow \tau \cdot \alpha$
14    UpdateParameters()
```

Algorithm 2 describes the ALNS algorithm. The ALNS was proposed by Ropke & Pisinger (2006) and has been applied to a variety of routing problems (Pisinger & Ropke, 2007; Hemmelmayr et al., 2012; Voigt & Kuhn, 2021). The algorithm begins with setting the local best solution $s^*$ to the starting solution, $s$ (line 1). The ALNS uses the global best solution $\hat{s}$, as soon as it exists, i.e., in all but the first generation. After initialization, the ALNS generates a new solution $s_{\text{new}}$ by removing and inserting customers with
randomly chosen removal operators and a deterministic insertion operator, as there is only one insertion operator (line 3). The probability of choosing removal operators depends on their performance during previous iterations. By comparing the current solution $s$ and the global best solution $\hat{s}$, a set of customers which are candidates for removal, $C_R^C$ is generated (line 4). The removal operator chosen then removes some or all customers from the solution which are included in set $C_R^C$ (line 5). The maximum number of customers to be removed in one iteration, $q_{\text{binom}}$, is sampled from a binomial distribution with sample size $|C|$ and probability $p_{\text{binom}}$. The number of customers that are actually removed by one of the below removal operators is expressed by $q^* = \min\{q_{\text{binom}}, |C_R^C|\}$. The ALNS uses four removal operators: Random Removal, Historic Cost Removal, Worst Cost Removal, and Shaw Removal (for details, see Voigt et al. (2021)). The removed customers are added to set $C_R$. After sorting $C_R$ according to data collected during the search (line 9), the customers are inserted using the Best Insertion Operator (line 7). This operator simply finds the position, where the customer can be inserted with the lowest total cost while respecting time windows. The ALNS uses simulated annealing to escape local optima (lines 8-12). Deteriorating solutions are accepted with a probability depending on the difference in costs of the candidate solution $f(s^{\text{new}})$, the cost of the best solution obtained during the run of the ALNS $f(s^*)$ and the current temperature $\tau$. The initial temperature is determined instance-specific with $\tau = -\frac{\Delta E}{\ln(\chi_0)}$ using the formula from Johnson et al. (1989). At the end of every iteration the temperature $\tau$ is multiplied with the cool rate $\alpha$ (line 13) and the parameters are updated (line 14).

The determination of removal candidates, adaptivity of parameters, and diversity management are equivalent to the HALNS originally proposed (Voigt et al. 2021).

3.4. Data-driven approach

This subsection presents the planned and then realized approach to leverage the data. During training, the planned approach (top of Figure 1a) solves instances using different strategies with the HALNS. The solutions are evaluated, resulting in instance-specific scores of strategies. An ML model is then trained to predict which strategy should be applied depending on instance features to achieve the lowest score. Unfortunately, the prediction accuracy of several tested ML models was not good enough during application (bottom of Figure 1a) to justify the additional complexity. Therefore, the final approach simply uses the cost adjustment/strategy with the lowest average score (Best Strategy) identified during experiments in Section 4 independent of instance features (see Figure 1b).

4. Experiments and Results

All experiments were conducted in parallel on an AMD Ryzen 7 2700X CPU with eight cores and 32 GB of RAM. Data and instances are prepared in Python, the HALNS is coded in C++. The test data set contains all 2718 high-quality routes given in the Amazon Challenge. The delivery period’s length $D$ is set to 9.5 hours for all experiments.
In addition to the strategies mentioned in Section 3.2, a combination of strategies and the influence of the number of generations were tested. All but the last two strategies use ten HALNS generations, $gen_{\text{max}} = 10$.

- **TSPTW**: Solve TSPTW without adjustments.
- **TSP**: Solve TSP, i.e., disregard time windows.
- **TSPTW-tri**: Solve TSPTW with triangle inequality adjustment.
- **TSPTW-gcd**: Solve TSPTW with adjustment of unreasonable expensive arcs.
- **TSPTW-depot**: Solve TSPTW with adjustment of depot traveling costs.
- **TSPTW-tri-gcd-depot**: Combination of cost adjustments.
- **TSPTW-tri-gcd-depot-1**: Combination of cost adjustments, $gen_{\text{max}} = 1$.
- **TSPTW-tri-gcd-depot-50**: Combination of cost adjustments, $gen_{\text{max}} = 50$.

Table 2 shows the average score and standard deviation for the different strategies. Interestingly, the scores of **TSPTW** and **TSP** are similar. This may indicate, that time windows are often not restrictive. All other cost adjustments reduce the average score compared to strategy **TSPTW**. The combination of strategies further reduces the score. Rows **TSPTW-tri-gcd-depot-1** and **TSPTW-tri-gcd-depot-50** show, that the score is only slightly affected by the number of generations, indicating that already good solutions are found in just one generation. The second-last row shows the theoretical performance of combining all strategies, assuming a perfect prediction model. Unfortunately, the ML model constructed was not able to predict the strategies with sufficient accuracy. Henceforth, the final approach uses simply the best strategy, i.e., **TSPTW-tri-gcd-depot-12**. As mentioned before, increasing the number of generations does not necessarily improve the score. Therefore, the number of generations is only slightly increased to $gen_{\text{max}} = 12$, also to remain well within the time limit of 240 minutes.

Figure 2 shows the distribution of scores for the chosen strategy, **TSPTW-tri-gcd-depot-12** for all 2718 high-quality routes tested. The majority of routes has a score ranging from 0.0 to 0.25. Only few routes have scores above 0.30. For these high-scoring routes, drivers may have followed other strategies.
Table 2: Results

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Average Score</th>
<th>Standard Deviation</th>
<th>Total Runtime [min]</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSPTW</td>
<td>0.0754</td>
<td>0.0512</td>
<td>104</td>
</tr>
<tr>
<td>TSP</td>
<td>0.0758</td>
<td>0.0512</td>
<td>103</td>
</tr>
<tr>
<td>TSPTW-tri</td>
<td>0.0748</td>
<td>0.0508</td>
<td>104</td>
</tr>
<tr>
<td>TSPTW-gcd</td>
<td>0.0721</td>
<td>0.0499</td>
<td>117</td>
</tr>
<tr>
<td>TSPTW-depot</td>
<td>0.0708</td>
<td>0.0522</td>
<td>103</td>
</tr>
<tr>
<td>TSPTW-tri-gcd-depot</td>
<td>0.0683</td>
<td>0.0513</td>
<td>120</td>
</tr>
<tr>
<td>TSPTW-tri-gcd-depot-1</td>
<td>0.0698</td>
<td>0.0517</td>
<td>38</td>
</tr>
<tr>
<td>TSPTW-tri-gcd-depot-50</td>
<td>0.0679</td>
<td>0.0510</td>
<td>461</td>
</tr>
<tr>
<td>Result assuming perfect prediction</td>
<td>0.0467</td>
<td>0.040</td>
<td>NA</td>
</tr>
<tr>
<td>Instead, chosen strategy: TSPTW-tri-gcd-depot-12</td>
<td>0.0678</td>
<td>0.0520</td>
<td>138</td>
</tr>
</tbody>
</table>

![Histogram of scores for chosen strategy](image)

Figure 2: Histogram of scores for chosen strategy TSPTW-tri-gcd-depot-12

5. Conclusion

Summary. This paper models the Amazon Challenge as Traveling Salesman Problem with Time Windows and uses the Hybrid Adaptive Large Neighborhood Search as solution approach. The cost matrix is adjusted to mimic the behavior of drivers. The results show, that there exist arcs with an unreasonably high difference in the way to and from a customer/depot, resulting in the tendency of drivers to still use these arcs even if the costs are supposedly high. Additionally, the costs resulting from traveling from the depot to the delivery area (and back) are less important, meaning that drivers tend to accept a long distance on their way to the delivery area (and back) if this results in short distances within the delivery area. This can be explained by the availability and convenience of highways to get from the depot to the delivery area (and back). Furthermore, the combination of different strategies results in a good score. A model that predicts which strategy should be used depending on the instance with sufficient accuracy has yet to be developed.

Avenues for further research. There exist three avenues for further research, building on this work:

Voigt
• **OR**: Interview drivers to get insights into strategies they may follow, e.g., avoiding left-turns, respecting breaks and customer-availability. Construct optimization models that include such strategies.

• **ML combined with optimization techniques**: Implement a model with higher prediction accuracy for predicting which strategy should be used depending on instance characteristics. Additionally, ML-based insertion operators could be integrated into the HALNS.

• **ML models directly constructing solutions**: Predict the sequence directly by using Gated-Graph Convolutional Neural Networks, complemented with features from the HALNS population.
References


Article XXVII

A Two-Stage Approach to Routing with Driver Preferences via Heatmaps

Connor Lawless, Sotiris Ntanavaras and Anders Wikum

Source code is available on GitHub:
https://github.com/conlaw/driver_preferences_heatmap
A Two-Stage Approach to Routing with Driver Preferences via Heatmaps

Connor Lawless, Sotiris Ntanavaras, Anders Wikum

Department of Operations Research and Information Engineering, Cornell
136 Hoy Rd, Ithaca, NY 14850

Abstract

In this report we outline a hierarchical approach to constructing vehicle routes based on historical driver preferences. We decompose the routing problem into a coarse zone-level planning problem, and then a more granular stop-level planning problem within each zone. To construct the zone-level plan we model the transition between zones as a Markov chain, where the transition probabilities between zones, denoted a heatmap, are computed based on historical routes taken by drivers. The objective of the zone level plan is to maximize the probability of a given route over the constructed Markov chain, which we solve by using an open source travelling salesman problem solver where the the logarithm of the transition probabilities are the edge costs. To generate a stop-level plan within each zone we construct an augmented $s-t$ Path-TSP instance that sequentially orders all the stops in a zone and the first stop in the next zone. On a validation set of over 300 high scoring routes, our method is able to achieve an average score of 0.036 - achieving a score under 0.01 nearly 50% of the time. Many of the remaining low scoring routes stem from incorrect prediction of the first zone in a given sequence or from ignoring backtracking for repeated delivery.

Acknowledgements

We would like to thank our project advisors for their insight and guidance in creating this work.


1. Introduction

Traditional last mile vehicle routing algorithms aim to create a tour of delivery stops in a given route that minimize both distance travelled and the number of missed time windows for packages. However in practice many drivers, each with their own tacit knowledge of their delivery routes and areas, divert from seemingly
optimal routes for a litany of reasons that are difficult to codify into an optimization model directly. Further complicating the problem is the need for algorithms that can scale to the volume of routes a large company like Amazon faces on a daily basis. Motivated by insights gleaned from the Amazon-MIT last mile routing challenge dataset, in this report we introduce a scalable approach to capture these driver preferences and create routes that closely follow real routes taken by drivers.

Our approach is derived from two key observations in the Amazon-MIT dataset. The first is that while driver preferences have a large impact on the broad structure of a route, namely the ordering of zones, the order of stops that a driver visits within a small geographical planning zone is very similar to the optimal \( s-t \) Path TSP tour, where \( s \) and \( t \) are the stops that a driver chooses to enter and exit the zone respectively. This suggests that given the zone ordering that a driver will visit, one can generate a route that is very close to the route that the driver will take by solving a \( s-t \) Path Travelling Salesman Problem (TSP) problem in each of the zones, after selecting the appropriate \( s \) and \( t \). For instance \( s \) and \( t \) can be selected to be the stops in the current zone that are closest to the previous and next zone in the zone ordering of the driver.

We create such a zone ordering based on the historical data provided, using a Markov model for transition between zones. The second key observation, albeit counter-intuitive, is that drivers rarely change the order in which they visit zones, even if it would mean meeting time windows that would otherwise be violated. This suggested that doing local adjustments to the order that the zones are visited might not yield better results.

These key observations motivate a two-stage approach for generating routes, where in the first stage, we generate the route at a zone level, taking into account the historical data given to us, thus trying to replicate the zone sequences present most often in the data, and in the second stage we solve a \( s-t \) Path-TSP problem in each of the zones to find the order in which the stops are visited in the zone. Focusing on solving the Path-TSP problem in each zone rather than on the set of all stops, makes the problem tractable such that most modern day solvers return an optimal solution almost instantaneously.

2. Literature Review

The Traveling Salesman Problem is a fundamental combinatorial optimization problem, and there have been countless instances of using TSP tours to plan vehicle routes, to ensure that every stop is visited once while minimizing an objective function \([1]\). Using historical data to predict vehicle routing, and specifically using Markov models for the predictions has also been the focus of several recent publications. Krumm \([2]\) proposed an algorithm for turn prediction using a Markov model, where the probability of the driver making a turn is computed based on the sequence of edges that were just traversed. Most similar to our work, Canoy et al. \([3]\) develop a Markov chain based model, that incorporates subjective driver preferences while also trying to minimize the standard distance based objective function that is used in vehicle routing problems. They model the stop sequences that appear in the driver routes as a Markov chain, and compute the corresponding probabilities from the historical data. Using those probabilities, they compute the most likely order that a
driver will visit the stops in a given route. One shortcoming of their approach is they assume individual stops are visited in multiple historical routes, however in this challenge stops are essentially unique. To circumvent this shortcoming, our approach constructs the Markov chain on a broader set of geographic zones that are visited multiple times throughout the historical routes. We then include an additional stage where we order stops within each geographic zone.

Our work also connects to the recent line of work that leverages machine learning to learn heuristics for routing problems (see [4] for an overview of machine learning applied to combinatorial optimization). While many researchers have designed end-to-end algorithms that attempt to learn how to map from route-level details to an optimal tours directly [5] [6] [7] [8], our work most closely relates to methods that use machine learning alongside traditional combinatorial optimization tools. Similar to our approach, Joshi et al. use a graph convolutional neural network to predict the probability an edge is included in an optimal TSP tour [9]. Kool et al. build upon this work and integrate edge prediction directly into a dynamic programming algorithm [10]. Our approach differs in that we aim to predict not the optimal tour but rather tours taken by real drivers. Our use of broader geographic structure (i.e. zones) as opposed to individual stops also lets us directly compute historical heatmaps, eschewing computationally intensive deep learning methods that would require thousands if not millions of sample routes to learn an effective policy.

3. Methodology

The key observation behind our approach is that within a given zone most historical routes closely resemble the optimal Path-TSP route between the starting and ending stops in the zone. To leverage this observation, we decomposed the route planning problem into a zone-level planning phase where we order the zones, and then do a stop-level planning phase where we order the stops within each zone. The advantages of such an approach are two-fold. Firstly, decomposing the problem into a set of smaller routing problems (i.e. one zone level problem, and a sequence of small stop-level problems) leads to a more computationally tractable framework that can be solved by standard open source routing solvers such as Google OR tools. Secondly, learning driver preferences over a coarser set of geographic locations (i.e. zones vs individual stops) is a simpler learning problem that can gain insights off of a small number of historical routes, as opposed to the large datasets required for sophisticated approaches such as deep learning. The underlying assumption in our approach is that zones are only visited once. Such an assumption is reasonable in the absence of time windows as the primary goal would be to limit backtracking between zones. However in the presence of conflicting time windows within a given zone an optimal route may revisit zones to accommodate both time windows. In practice we observed that conflicting time windows within a zone occurred rarely, and any local search procedures to minimize time windows resulted in lower empirical performance. Figure 1 shows a graphical representation of our approach, subdivided into the model build phase and the model apply phase. Sections 3.1 and 3.2 explain both stages of the route planning process in greater depth.
3.1. Zone Level Planning via Heat maps

One of the challenges of learning driver preferences from historical route data is that the specific stops are essentially unique, limiting the ability to generalize historical stop-level sequences to new routes. Geographic zones in the routing data provide a broader structure to the stop data that are repeated frequently across routes. For the first level of our approach, we aim to leverage historical zone to zone sequences from previous routes to learn driver preferences in route planning.

Following the approach of [3], we model the problem as a Markov chain over the zones where the arc probabilities correspond to the likelihood a driver takes a specific zone sequence. We construct these arc probabilities by looking at historical zone to zone sequences taken by drivers. Let $f_{i,j}$ be the number of times a driver travelled between zone $i$ and zone $j$ in the training data. We define the transition probability between zones $i$ and $j$, $p_{i,j}$ as follows:

$$p_{i,j} = \frac{f_{i,j} + \lambda}{\sum_{j'} f_{i,j'} + \lambda}$$

where $\lambda$ is a laplacian smoothing term, which we set to 0.01 in our experiments. Note that additional weighting can be applied to transits from routes of different qualities (i.e. placing higher weight on high scoring routes), however empirically we found that equal weighting to all historical routes was the most effective. We denote the graph constructed from these historical visitation sequences a heat map. We construct a separate heat map for each station included in the training set of routes.

Our goal is to maximize the total probability of a given route $R$ (i.e. the product of arc probabilities along the chosen route) out of the set of all possible routes $\mathcal{R}$, which is equivalent to maximizing the sum of the log probabilities of transition probabilities.
\[
\max_{R \in \mathbb{R}} \prod_{(i,j) \in R} p_{i,j} = \max_{R \in \mathbb{R}} \sum_{(i,j) \in R} \log(p_{i,j})
\]

We solve the zone-level planning problem by using the log transition probabilities as the edge costs in Google’s OR Tools TSP solver.

**Heat Map Normalization:** While the heat maps are generated for all zones seen for a given station, a given route will likely only visit a small subset of the total zones. To account for this when creating the zone-level route, we normalize the heat map for the subset of zones included in the given route. Consider a given heat map \( M \), where the columns and rows are permuted such that all the rows and columns corresponding to zones for a given route are in the top left of the matrix.

\[
M = \begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\]

A represents all transitions between zones in a given route, B represents transitions between zones in a route and additional *extra zones* (i.e. potential zones for the given depot, not included in the route), C represents transitions from the extra zones to zones in the given route, and D represents transitions between extra zones.

To compute a heat map just for the subset of relevant zones, the goal is to capture any multi-step transitions that occur between relevant zones via zones not included in the set of stops (i.e paths that start and end in relevant zones but visit additional extra zones). The probability of a path starting in a relevant zone, transiting to \( k \) extra zones, and then returning to a relevant zone can be represented in matrix form as \( BD^k C \). Let \( H \) be the normalized heat map for a given set of stops. If we sum over all possible routes involving relevant zones that transit to extra zones, we can represent \( H \) mathematically as:

\[
H = A + BC + BDC + BD^2C + \ldots = A + B\left(\sum_{i=0}^{\infty} D^i\right)C = A(I - D)^{-1}C
\]

where the last step follows from the use of the Neumann expansion. For each new set of routes, we compute the normalized heatmap using the above closed-form equation and proceed as discussed previously in constructing a zone-level plan.

**Missing Data:** Unfortunately, there is no guarantee that every stop identified in a given route has an associated zone. For such stops, we merge them into the closest zone. An alternative would be to create a new zone for each unidentified stop, however we found that such an approach both led to higher computation time, as it increased the number of zones to consider in the zone level planning, and led to worse performance empirically. In addition to stops with no zones, there may be new zones encountered in a given route that was not seen in the historical data, and thus would not be included in the generated heat map. To account for these new zones, we first generate a zone-level plan for the zones included in the heat map and then insert the new zone between zones where it causes the smallest increase in distance travelled. If more than
50 percent of the zones have never been seen before, we forgo using the heat map and simply construct a zone level plan by using the zone to zone distances as edge costs.

3.2. Stop Level Planning via Path TSP

After generating an ordering of the zones in a given route, the final step is to order the stops within each zone. A baseline strategy is to fix the starting and end stops of a given zone by selecting the stops that are closest to the preceding and following zones in the route respectively. With the start and end stops fixed, the remaining stops can be ordered by simply running a Path TSP between the start and end stops using an open source TSP solver such as OR Tools. However, in practice we found that for many zones such a strategy leads to selecting the same start and end stop or requires inefficient backtracking. Instead, we solve a Path TSP problem that learns the route between the first stop in each zone, and the first stop of the next zone. To do so, for each zone we create an augmented graph consisting of all the stops in the zone and one node representing the next zone. The arcs from each stop in the current zone to the new node is the minimum distance from that stop to any stop in the next zone. We then run a Path TSP on the augmented graph with the given starting stop and the new node as the end stop. To determine the starting stop of the next zone, we simply look at the final arc in the optimal path and select the stop that attains the minimum distance between the last stop in the current zone and the next zone.

4. Results and Conclusions

To evaluate the performance of our model, we combined both the given training and evaluation datasets provided and randomly selected 10 percent of the given high scoring routes to act as a larger evaluation set. The remainder of the routes, regardless of their route score, were included for generating the heat maps. Figure 2 (Left) shows the distribution of scores on our validation set. Our two-stage algorithm achieves an average score of 0.036 on our validation set. While the algorithm is able to closely match the true high scoring route on a large fraction of routes, there are some notable outliers that score as high as 0.2.

Figure 3 shows an example of a low scoring route (0.18) which exemplifies the types of routes where the algorithm performs poorly. In this case there are two major discrepancies between the proposed route and the high scoring route taken by the driver. The first is that heatmap incorrectly estimates the first zone to visit from the depot to be in North Laguna Heights (bottom section of map) as opposed to Los Olivos (top section) where the true route starts. The actual driver also returns to Los Olivos at the end of route to re-attempt a delivery that was missed earlier in the route. However, given that a delivery failure cannot be known a priori, it is reasonable to expect the algorithm to miss such re-attempts. Beyond the choice of initial zone and backtracking, the routes take a nearly identical sequence of stops. This holds true across the other small number of low scoring routes - suggesting a more accurate prediction model for initial zones visited could remedy a number of these outliers.
4.1. Ablation Study

To evaluate the impact of our various algorithm components, we ran ablation tests that sequentially added components of the algorithm to determine the relative impact of each aspect. Figure 2 (right) shows the average score on the validation set for models incorporating subsets of the features in the final algorithm. Stop-TSP refers to a baseline algorithm that simply computes the shortest path between all the given stops using the Lin-Kernigham Heuristic [11]. Two-Stage refers to using our hierarchical approach that first generates a zone-level ordering, then a stop-level ordering. In this approach the zone-level ordering is computed using distances between zones, defined as the minimum distance between a stop in one zone to a stop in the other. To compute the stop-level ordering, this approach simply fixes the start and end

![Figure 2: (Left) Distribution of scores on validation set for final algorithm. (Right) Ablation study of algorithm components on model score.](image)

![Figure 3: (Left) Actual high scoring route taken by a driver. (Right) Proposed route from two-stage algorithm. Arrows indicate direction of travel, and orange outlines indicate zone boundaries.](image)
node to be the closest stop to the preceding and following node respectively. Heatmap also uses the two-stage approach but generates the zone ordering using heatmaps. Similarly, Normalized Heatmap normalizes the route-specific heatmap. Finally, contracted node TSP uses the path TSP described in section 3.2 to determine the stop level ordering.

The largest jump in performance comes from switching to a two-stage approach that first does zone-level planning. However, using heatmaps based on historical driver preferences and normalizing them on a route level also led to an almost 50% performance over the baseline two-stage method. Finally using a contracted node TSP as opposed to naively setting the start and end node within a zone to be the closest stop to the preceding and following zone also led to a modest bump in performance.

4.2. Conclusion

In this report we outlined a two-stage approach for vehicle routing that leverages historical driver preferences to create zone-level route planes before using a finer-grained path TSP to do stop-level planning within routes. The zone-level planning is driven by a heatmap, constructed from previous routes, that predicts the likelihood a driver travels between zones. These heatmaps are also normalized to account for the zones present in a new route. Our approach is able to achieve an average score of 0.036 on a validation set of previous high scoring routes, closely matching nearly half the routes. Further development into more sophisticated machine learning techniques to generate the heatmaps could further improve the algorithm, mitigating the small fraction of the time we incorrectly order the first few zones in a route.
References


Article XXVIII

Imitating a Variant of the Traveling Salesman Problem

Venktesh Pandey, Tarun Rambha and Neha Singh

Source code is available on GitHub:
https://github.com/tarunrambha/amazonRoutingChallenge.git
article xxviii

imitating a variant of the traveling salesman problem

venktesh pandey
assistant professor
north carolina agricultural and technical state university, greensboro, nc, usa

tarun rambha
assistant professor
indian institute of science, bangalore, ka, india

neha singh
phd candidate
carnegie mellon university, pittsburgh, pa, usa

abstract

variants of the traveling salesman problem are widely used in logistics and may involve multi-criteria objectives and other practical constraints. using historical data on optimal and sub-optimal routes from a problem with an unknown model specification, we try to infer the features that can help classify route sequences into ordinal categories based on their solution quality using supervised learning techniques. this classifier is then used to pick high-quality route sequences on new data from a pool of solutions generated using neighborhood search. results on the amazon challenge data indicate that the scores between the actual and the proposed sequences fall in the range of 0.058-0.062 when tested on a hold-out sample from the training data.

1. introduction

the routing challenge required constructing route sequences for new stops given historical data on the quality of older sequences and their stop and package information. the given problem can be viewed as a version of the traveling salesman problem with time windows (tsptw). a logistics firm, however, may not only care about the travel time or cost of a tour but also factor in other aspects such as fuel consumption, ease of navigation, tightness of delivery times, etc. [holland et al., 2017]. these can appear as constraints or as parameters in the objective of the tsptw problem. however, solving these integer programming formulations is challenging since finding a feasible solution to the tsptw itself is known to be an np-hard problem [savelsbergh, 1985]. from a practical standpoint, firms solve the same model repeatedly across
days but with new data sets (Holland et al., 2017). This begs the following question: Is it possible to learn complex mappings from stop/package data to ‘optimal’ tours by using past solutions and inferring the planner’s objectives?

Our approach to this problem is summarized in Figure 1 and involved the following steps:

1. Extract features that may be relevant for a logistics firm from data on actual sequences.
2. Train a classifier to predict the probability with which the sequence quality is high, med, or low.
3. Use the new route stops to create a pool of candidate tours.
4. Each time a new candidate tour is created, extract features similar to those used in Step 1 and predict the quality of the tour.
5. Select the tour from the pool of solutions that has the maximum probability of being rated high.

2. Literature Review

Our methodology uses techniques from both operations research and supervised learning. The summarized approach in Figure 1 gives an idea of the problems (TSPTW + feature selection + train classifier) that are tackled separately, and the solutions are combined to get the final result. The ML component of the problem follows standard steps from the literature (Bishop, 2007), where the difficulty lies in finding the best features followed by choice of the classifier. The OR component of the problem relies on zone-based TSP solution methods, where TSP tours for stops within the zones are found, and obtained routes are then stitched together to get the final route with time window feasibility, similar to partitioning algorithms in (Karp, 1977; Khan et al., 2012). The candidate solutions for the zone-based TSP tours used in this work have the same flavor as neighborhood search algorithms that have been successful in solving TSPs (Croes, XXVIII.2).
While a few Reinforcement Learning and Graph-ML based methods have been recently proposed that learn from the graph structure, (Dai et al., 2017; Kool et al., 2018; Joshi et al., 2019), they have not been shown to work well on problems that may involve multiple objectives and constraints. Also, the training time and hardware requirements of these methods are more demanding than what was available for the competition.

Recent papers by Arnold & Sørensen (2019) and Lucas et al. (2019) have the closest resemblance to this challenge. The authors solve a capacitated vehicle routing problem for benchmark datasets to different levels of optimality and extract features to distinguish between good and poor solutions. However, a key difference between their framework and the current problem is that the goals of the decision maker are unknown in our case.

3. Methodology

As described in Figure 1, our approach can be broken down into three steps: (a) selecting features from the provided data on routes, stops, and packages, (b) training a classifier that, given the features, predicts the route label (low, medium, or high), and (c) designing routes for a given set of new stops that predict a higher probability of being labeled high. We describe each step in the following subsections.

3.1. Feature selection

It can be noticed from the data that high route sequences did not necessarily have the shortest TSP tour times and did not always deliver all the packages. This motivated us to search for features that were significantly different across the three route categories (see Figure 3 for example).

To calculate the TSPTW solutions and a few stop-level feasibility features, we had to make assumptions on the time windows at each stop. Since time windows were specified at a package level, we could assume a copy of the stop for each package, but such a transformation would approximately double the problem size and may encourage revisits to a stop. Instead, we assign to each stop the tightest time window within which all packages can be delivered (see Figure 2 where the brackets for different colors indicate the start and end times for different packages). This assumption did not result in instances where the end time is earlier than the start time at a stop.

The features that were extracted are listed below, along with a brief description in cases where the name is not self-explanatory. Some of these metrics were not used in the final code and have been marked with an asterisk (*).

- **Journey-based metrics**
  - Travel time, total Haversine distance, and average speed
Volume seconds traveled: The fuel consumption of a vehicle is usually proportional to its mass. Thus a route sequence can lead to better fuel economy if the heavier packages are delivered in the initial part of the trip. However, visiting such stops first may lead to longer tour times. To balance these trade-offs, we calculated the product of the volume of packages on the vehicle and the travel time between each pair of stops. This metric was then summed across all the edges in the tour as a proxy for the mass of the vehicle, which influences the fuel consumption. The volume was used in these calculations since the weight information was not available.

TSP time: Since different routes have a different number of stops to visit, we used the TSP time without time windows as a reference or a lower bound for the total travel time. This was calculated using the Google OR tools library.

Time window-based metrics

- Number of stops and packages with TW violations
- Number and volume of packages not delivered
- Total wait time: We assume that a vehicle can arrive at the stop before the start of the service time, in which case the driver will have to wait. Waiting affects driver productivity, and hence we expect routes with higher total wait times (at all stops with time windows) to be less likely to be rated high.
- Total slack time: We expect that stops with time windows should be visited well in advance of their end service times. For example, if the end service time of a package is, say 10:00 AM, it is risky to plan to reach the stop at 9:55 AM since there could be delays due to traffic and other events. Hence, we record the sum total of the time differences between the end service time and the departure time of the truck at stops with time windows.
Package-level wait and slack times weighted by volume: The above two variables measure visits to a stop that are too early or too close to the deadline. However, the time windows at a stop-level are calculated using some assumptions described in Figure 2 and hence these metrics are approximations. To calculate the actual wait and slacks, we also looked at package-level data for those packages with time windows. Similar metrics were weighted by package volumes before calculating the sum since packages that are not delivered successfully will have to be fit in another truck and be delivered at a later date/time. We occasionally used normalized variants of some of these metrics since each route had different packages, stops, and zones. For example, we used ratio of the total slack time to maximum available slack which was computed as the sum the differences in end times and the (start times + service times) across all stops that had time windows.

Is sequence feasible: Since the stops have time windows, we check if the actual route sequence is feasible if each of the stops with time windows is serviced within the start and end time limits.

**Navigation-related metrics**

- Number of small angles: Delivery routes may often require trucks to make sharp turns. This may require trucks to perform multi-point turns since they have a larger turn radius. Such maneuvers are time-consuming. We counted the number of small angles using only the stop coordinates and the straight line segments of the actual route sequences.

- Number of left turns: Making left turns at signalized and unsignalized intersections can also lead to additional delays and increased fuel consumption. Since trucks have to find gaps in oncoming traffic, it can potentially lead to accidents. We computed this metric using the shortest distance paths between stops on an Open Street Maps (OSM) network, and the classification accuracy improved by about two percentage points. However, the computational costs of calculating these metrics for every candidate route in the apply phase prevented us from using this in the final submission.

**Segment- and zone-level metrics**

- Maximum segment time/distance: From the actual sequences, we noticed that in several cases, trucks make a long trip from the depot to the first stop. However, it is less common to have a long trip in the subsequent part of the journey for the routes rated high than the ones rated med. This may be unavoidable in some situations depending on the nature of time windows, but if prevented, avoiding long trip segments can reduce fatigue due to driving. Hence, we choose to measure the maximum of the travel time and Haversine distances between stop segments.

- Variance of segment time/distance: We also included the variance of the segment-level travel times and distances to encourage the driving sections to be uniform.
– Number of zone switches: It was also observed that stops belonging to a zone were delivered around the same time before proceeding to other zones. Hence, we counted the number of times a truck switches zones as it delivers packages along its route. The lowest possible value for this feature is equal to the number of zones.

• **Others static features** A few other static features which do not change with changes to the route sequence were also estimated and used in training. These variables were included to capture interaction effects. That is, they could jointly influence the route classification when paired with other features.
  
  – Percentage of capacity filled
  
  – Boolean variable representing a Weekend/Weekday

3.2. Classifier training

Given the feature vector $x_r \in X$ of a route $r$ provided in the model_build phase, the classification problem trains a “non-linear” function $f : X \rightarrow Y$ that maps the feature vector to an associated label (that is, $Y = \{\text{high}, \text{med}, \text{low}\}$) such that the classification accuracy is improved. The classification process follows the standard steps from the machine learning literature [Bishop 2007], where we split the given data into testing and training sets, train the classifier on the training data, and evaluate the label prediction accuracy on the testing data. The testing data comprised of 15% of **High** routes from the build phase and were randomly chosen. In our experiments, we considered the following classifiers and implemented them using Python's **scikit-learn** library.

  • Ordinal linear regression (Logit/Probit)
  
  • Random-forest classifier
  
  • Ensemble learning classifier—XGBoost and AdaBoost
  
  • Deep-neural network

Almost all models resulted in a prediction accuracy in the range of 64–68%. XGBoost offered a marginally better percentage and was used in the final submission. The training times for the above methods were also comparable. The hyperparameters were tuned using random grid search methods.

3.3. Route design

For the model apply phase, we first generate candidate routes that may potentially be rated high. Our candidate routes are developed using a partitioning method in which we first compute the centroid of each zone (see square markers in Figure 4), and zone-to-zone travel times are derived from averaging the travel times between every pair of stops from one zone to the other. To find the optimal TSP tour, most algorithms use neighborhood local search methods in which new solutions are generated by perturbing the order in which
vertices are visited. We used the GUIDED LOCAL SEARCH metaheuristic from the Google OR tools library and saved a pool of routes that were discovered before finding the optimal solution.

For every TSP tour between zone centroids say $\text{Depot} - z_1 - z_2 - \ldots - z_n - \text{Depot}$, we construct a full tour in the following way:

1. Find the stop in zone $z_1$ that is closest to the depot. Let us denote it as $s_d^{z_1}$.
2. With $s_d^{z_1}$ as the depot solve another TSP with only the stops contained in $z_1$. Refer to the graphic in the second column of the top row in Figure 4.
3. Instead of going back to $s_d^{z_1}$, connect the last node of this sub-tour to the nearest stop in the next zone of the original tour $s_d^{z_2}$ and repeat Step 2 until a sub-tour is constructed for the last zone $z_n$.
4. Connect the last stop visited in this sub-tour to the original depot to get a full TSP tour.

The time windows are ignored in this process since it is difficult to generate feasible TSPTW tours and because we expect that the classifier will help discard routes that violate the time window constraints. For each of the zone-based TSP tour and sub-tours within each zone, we also reverse the route order to get more candidate solutions. We limit the number of such solutions and also have a computational timeout to allow processing all the routes in the model apply phase within the time budget. The travel times of the zone-based solutions are not far from that of the global TSP, and it also ensures that stops within each
zone are served together, a feature that we noticed was common to many of the high route sequences in the actual data.

Finally, for each of these full TSP tours, we extract the features used during the build phase and predict the probability of being rated high using the XGBoost method. The one with the highest estimated probability is saved as the proposed route.

4. Results and Conclusions

To test the performance of the proposed method, we randomly removed 15% of the high route sequences from the original data set of 6,112 routes and evaluated the scores between the actual and proposed routes. Our scores varied in the range of 0.58–0.62 depending on the seed used to select the testing sample.

A plot of the estimated probability of high and the actual score for a subset of solutions in the candidate pool is shown in Figure 5.

The labels PST indicate reversed TSP tours. Ideally, the plot should have been linear in which routes with low scores should have a high probability of being rated high. However, we noticed that the relationship between these two variables could be noisy, and the probability of being rated high is not always a proxy for the score. A possible reason could be that we may not have captured a few key features important to the planner.

The distribution of scores for a test sample of 385 routes is shown in Figure 6. The average score across all routes for this instance was found to be 0.0582 with a majority of routes with scores in the bin 0.043–0.052. We did not find any noticeable trend in the features of the routes which had low and high scores except for the average distance and the TSP optimality gap. As can be seen from the second and third panels of the
The trend of count of score, average of total distance and average of TSP optimality gap for score (bin).

**Figure 6:** Histogram of scores for the proposed routes and its relationship with average values of actual sequence features.

A reason for this trend could be that the original route sequences in these cases were longer than the ones discovered by zone-based TSP methods. The average number of stops for which time windows were violated were more or less similar across these routes except for the ones on the far right, and hence the longer detours can be assumed to meet other objectives or constraints that were not captured in our current set of features.

We however found noticeable differences in the distribution of scores when they are grouped by the cities to which the routes belong (see Figure 7). On an average routes in Chicago had relatively higher scores. The variance for routes in Austin is high because of fewer data points. There were also differences in the average scores for routes that were operated on weekdays (0.060) vs. weekends (0.052). These findings probably indicates that there could be city-specific factors such as congestion and parking which plays a role in the quality of routes. Engineering features that capture the motives of a planner is a challenging problem and warrants further exploration.
References


Article XXIX

Transformer Model with Graph Convolution Embeddings for Route Planning

Rashmi Kethireddy

Source code is available on GitHub:
https://github.com/r39ashmi/LastMileRoutingResearchChallenge.git
Transformer model with graph convolution embeddings for route planning

Rashmi Kethireddy

Speech Processing Laboratory
International Institute of Information Technology-Hyderabad (IIIT-H)
500032, India.

Abstract

Our approach focuses on the application of heuristic methods that are close to human behaviour. Humans selection of stop depends on previous stops visited and the stops to be visited in future. We propose to you an attention-based neural network model that considers the context for selecting a stop. Attention-based model is trained to predict the route sequence and the loss function is computed as cross-entropy between the actual and predicted routes. Adjacency matrix of travelling time is represented as graph embeddings using graph convolution layer.

1. Introduction

The routing algorithm requires longer temporal context that is the decision of current stop selection depends not only on the current stop features but also previous and future decisions. Deep neural network with transformer model that provides wider temporal context if investigated in text and speech domain [Vaswani et al. 2017] where longer temporal context is required for decision at each node. The network architecture of our model is inherited from the transformer model (that provide longer context to) in [Kool et al. 2019; Vaswani et al. 2017]. This study investigated transformer model in the Vehicle Routing Problem (VRP), the Orienteering Problem (OP) and (a stochastic variant of) the Prize Collecting TSP (PCTSP) and compared to state-of-the-art pointer network. It is observed that the attention model performed better in all the problems. The attention architecture from [Kool et al. 2019] has only latitude and longitude information for each stop. To adapt our current problem to this network, we handled each input differently, the embeddings of each input are extracted independently and provided as input to the network. Stops information is handled differently (Refer Section 3 for more details).
2. Literature Review

First application of neural network for combinatorial optimization problems was in Hopfield & Tank (1985). Pointer network (Vinyals et al., 2015) is the first DNN with attention is introduced for combinatorial optimization problems. Transformer based neural network is applied to Vehicle Routing Problem (VRP), the Orienteering Problem (OP) and (a stochastic variant of) the Prize Collecting TSP (PCTSP) in Kool et al. (2019).

Last mile routing challenge have different input data representations and constrained to follow the optimal route than feasible route. To handle the travel times in neural network, temporal dependencies are computed with graph convolution layer Kipf & Welling (2016) to extract embeddings for each stop.

3. Methodology

This approach has three main modules, extraction of embeddings from input features, network architecture, and cost function for the network to learn to route.

3.1. Embedding layers:

- **ZONE**: Zone of the station is always labelled as 0 and zones of stops are randomly labelled with integer numbers. Missing zones are filled with the closest stop’s zone. These label encoded zones are given as input to zone embedding layer. The size of embeddings extracted from zone is 8.

- **LOCATION**: Sum of volumes of packages for each stop are computed from length, height, and width of the package. This is included along with latitude, longitude of each stop. Location embedding layer is used to encode them to network.(120)

- **DEPOT**: Embeddings for depot node are separately extracted. 128

- **TIME CONSTRAINTS**: Time constraints of each stop includes start time, end time (that considers overlapping time of all the packages), and total time required for delivering all the packages. Latent representation of these features are extracted using time constraint embedding layer. Embeddings of dimension 8 are extracted from time constraints.

- **TRAVEL TIME**: Travel time between stops are given in an cost matrix. Embeddings for each stop are extracted using graph convolution layer Kipf & Welling (2016).

All the embeddings are concatenated and processed with the encoder.
3.2. Model Architecture:

Deep neural network (DNN) has encoder and decoder framework. Encoder converts variable length stop information to fixed length latent representations. These latent representations as input, decoder predicts the order of stop sequence. DNN architecture is inherited from Kool et al. (2019).

Figure 1 shows the encoder architecture. \( x_n \) in the figure denotes the input features from each node. \( h_n \) represents concatenated embeddings from each variant as defined above. For details about other layers please refer Kool et al. (2019).

![Figure 1: Block schematic showing encoder architecture](Kool et al. 2019)

Figure 2 shows the decoder architecture. Stop is selected at each time in the sequence with greedy criteria. For more details about layers please refer Kool et al. (2019).

![Figure 2: Block schematic showing decoder architecture](Kool et al. 2019)

3.3. Loss Function:

Cross entropy between target and prediction at each instant is considered as loss. The weighted loss is back-propagated for each route sequence. Selection of weight is based on the order of their score.

4. Results and Conclusions

Minimum score obtained across the epochs is 0.364. Routes with route ids RouteID.5486294a − 503f − 4346 − b8a9 − 862e988cbe7c, RouteID.e6687a05 − 2453 − 4edc − b86c − 7558ab683f6, RouteID.f3261fad −
5f97–44f6–ae7f–ef169f5d6452, and RouteID_ffffff257c–3041–4736–be7a–5efa8af1173 are consistently easy in every epoch while RouteID_9475872b–287f–4e2c–8e29–887766a4e090, RouteID_a8f0009d–e50a–49c9–84d3–f9885ad14a54, RouteID_d1a8c3dd–fa67–455c–a68d–af2fd6aa5d91, and RouteID_2b8df66df–fc4d–438e–931c–3b8b3a5c6b are consistently hard to predict the sequence. Further applying shuffling of input sequence for every iteration might improve the training space of routing sequences.
References


Article XXX

TSP with Learned Zone Preferences for Last-Mile Vehicle Dispatching

Rocsildes Canoy, Jayanta Mandi and Victor Bucarey

Source code is available on GitHub:
https://github.com/JayMan91/vub-routingchallenge
Abstract

In the context of the Amazon routing problem, we propose a two-stage approach to generate routes which capture the drivers preferences. During the initial stage, we learn the latent preferences of the route planners from actual routes in the historical data in order to determine the preferred zone sequence. Learning is done by adapting a preference learning technique introduced in the literature, where we construct a zone transition probability matrix for a given delivery station then use the resulting matrix to solve for the zone order. In the second stage, we determine the final route by solving a traveling salesman problem (TSP) while respecting the zone order obtained from the previous stage. Numerical results show that our proposed approach, with an added modification of combining zone transition probabilities and distance information during the initial stage, performs significantly better than the standard TSP solution method.

1. Introduction

The demand for delivery services has substantially increased, especially during the COVID-19 pandemic, particularly in urban areas. Moreover, this trend is expected to continue. Last mile logistics refers to the last stretch of the supply chain from the last distribution center to the end users. Last mile deliveries account for 53% of shipment costs (Gutierrez-Franco et al., 2021), and Gevaers et al. (2014) describe it as the most expensive and polluting segment of the supply chain. This makes the case for efficient management of last mile deliveries.

Last mile delivery is unique in its characteristics. The set of customers is bound to vary from one day to another as oftentimes, the majority of the users do not place an order daily. Also, on a particular day, the set of users sprawls in all directions throughout the city. Consequently, drivers may not be familiar with
the exact addresses and the routes to reach their destinations. Moreover, the customers may have specified the service time and in order to respect that, it is important to have an accurate estimate of the travel time between the addresses. That is why organizing efficient delivery routes for a large number of fragmented customers in last mile delivery is a challenging problem.

In this manuscript, we focus on the challenges faced by Amazon in delivering products to the customers. We have access to historical route data from multiple delivery stations in North America. Each day, each station receives the delivery locations and solves a TSP for a single vehicle. Some of the locations have specific service time, whereas most do not indicate any preferred time. Each station’s service area is further divided into multiple zones and except for a few, most of the locations are assigned to zones. A quality score for each of the historical routes is also provided.

The goal of this research is to develop methodologies to generate good quality route sequences using the historical data. As is the usual case in machine learning problems, we have two phases. The first phase involves building the model using the historical data. The second phase entails deploying the built model on the new instances. We place emphasis on low computational time during the deployment phase, such that our algorithms can be applied in real-life large scale instances.

The contribution of this work is the development of a two-stage approach to improve the single vehicle routing assignment for the last mile delivery using historical data. In the first stage, we determine the order by which the different zones are to be visited. In the second stage, we identify the final sequence of stops while respecting the predetermined zone order.

2. Literature Review

The last mile delivery [Zeng et al., 2019] problem has been well studied in the literature. Because of its extensive use in e-commerce, it has received increasing attention in recent years. Last mile delivery can be considered as a variant of the travelling salesman problem (TSP) [Miller et al., 1960] or the vehicle routing problem (VRP) [Dantzig & Ramser, 1959]. As it involves multiple stakeholders, each having a different objective function, different routings are possible depending on which has been considered as the objective. For instance, a common approach is to minimize the time to finish serving all customers, i.e., the makespan of all the routes [Yu et al., 2017]. On the other hand, Das et al. [2018] approach the problem with the objective of minimizing latency or the waiting time of the customers.

However, Li & Phillips [2018] state that in most cases, the last mile delivery does not go according to the planned schedule. In the context of routing between point to point, Ceikute & Jensen [2013] highlight that drivers tend to follow frequently used routes, which are not necessarily shorter in terms of distance or travel time. This is primarily because, in deciding their routes, drivers consider factors which are not in the objective function, e.g., traffic congestion and availability of parking and fuel stations. Drivers have tacit knowledge of these information. Toledo et al. [2013] also point out that drivers tend to rely on tacit knowledge to plan their routes. Explicit formalization of their knowledge poses an insurmountable task.
Several works have studied the problem of learning the routing preferences of the drivers from past routing trajectories. Delling et al. (2015) present a customizable route planning framework that infers the preferences of the drivers from GPS traces by learning individualized cost functions. Letchner et al. (2006) compute the ratios of the individual drivers’ travel time and the theoretical optimal travel time to learn the implicit preferences and biases. Both of the above-mentioned studies focus on point to point commute. In our work, we do not explicitly learn the parameters of the cost function nor do we estimate preferences from the travel times.

Canoy & Guns (2019) approach the problem from a different perspective—they introduce a weighted Markov model to learn the preferences from historical paths. This technique avoids the need to explicitly specify the preference constraints and implicit sub-objectives. Their Markovian model computes the transition probabilities between the stops. The model is then used to solve a maximum likelihood routing problem to come up with a routing scheme. We extend their framework to develop a two-stage TSP, where in the first stage we adapt the technique to compute the transition probabilities between zones to find the zone order.

3. Methodology

We exploit the historical data during the training phase. Each of the routes in the data starts from the station and returns to the station after visiting all the stops. As each route uses one vehicle, we have a traveling salesman problem (TSP). We propose a two-stage approach in solving the TSP. The main idea is to determine the preferred zone order during the initial stage by learning from the zone sequences in the historical data and the route quality labels attached to the routes. We also use the technique introduced in Canoy & Guns (2019) of combining transition probabilities and approximate distances, adapted here to the context of zones, to obtain an optimal zone order. With this technique, we get a desirable balance by maximizing the likelihood of the resulting zone order, while at the same time minimizing the travel distance with respect to the zones. In the second stage, the route (i.e., the stop order) is optimized by minimizing the total travel time while respecting the zone order obtained during the initial stage.

We formally introduce our two-stage approach by presenting the following algorithms. Algorithm 1 outlines the process of constructing the zone transition probability matrix for a single station from historical data. All route instances from the historical data are grouped by station code and Algorithm 1 is applied to each group to obtain a probability matrix for each of the stations.

Let $R_S = \{r_1, \ldots, r_p\}$ denote the set of historical routes originating from station $S$. The first step in Algorithm 1 is to create $\Sigma$, the set of all the zones visited in $R_S$. The cardinality ($t$) of $\Sigma$ will determine the dimension ($t+1$) of the probability matrix. The $(i,j)$-th element of this matrix denotes the transition probability from zone $i$ to zone $j$. It has $t+1$ rows and columns because we consider the station as a separate zone. To form the probability matrix, in line 4 of Algorithm 1, we first construct an adjacency matrix of size $(t+1 \times t+1)$ for each route in $R_S$, from the sequence of zones traversed in the route. The
zone transition frequency matrix is built by taking the sum after each adjacency matrix is multiplied by a numerical weight. This weight is dependent on the quality score label of the route from which the adjacency matrix is constructed. Finally, the desired probability matrix is obtained after normalization.

We manipulate how much the zone sequence contained in a route influences the resulting zone transition probability matrix by considering the route quality score associated to each route. Our goal is to put more importance and hence, allocate a larger weight to high quality routes (and a smaller weight to low quality routes). We therefore assign numerical weights \( \{w_{\text{high}}, w_{\text{medium}}, w_{\text{low}}\} \) corresponding to each of the score labels \( \{\text{high}, \text{medium}, \text{low}\} \), where \( w_{\text{high}} \geq w_{\text{medium}} \geq w_{\text{low}} \). Starting with default weights \( \{3, 2, 1\} \) for \( \{w_{\text{high}}, w_{\text{medium}}, w_{\text{low}}\} \), the parameters can be further fine-tuned during training.

Algorithm 1: Building the zone transition probability matrix for a single station

**Input**: Set \( R_S = \{r_1, \ldots, r_p\} \) of actual traversed routes from historical data, all originating from the same station \( S \); each route \( r_k \) labeled with route quality score \( q_k \in \{\text{high}, \text{medium}, \text{low}\} \)

Weights \( w_k \), which are numerical equivalents of the labels \( q_k \)

**Output**: Zone transition probability matrix \( P_S \) for routes originating from station \( S \)

1. Extract and gather all the zones visited in \( R_S \) into a set \( \Sigma = \{z_0, z_1, \ldots, z_t\} \), where \( z_0 \) denotes the zone of station \( S \).
2. for each \( r_k, k = 1, \ldots, p \) do
   3. Determine the sequence of zones \( Z_{r_k} \) traversed in \( r_k \).
   4. Construct an adjacency matrix \( A_{t+1} \times t+1 = [a_{ij}] \), where \( a_{ij} = 1 \) if the transition \((z_i \rightarrow z_j)\) is in \( Z_{r_k} \), and 0 otherwise.
5. end
6. Build the zone transition frequency matrix \( F_{t+1} \times t+1 = [f_{ij}] \) with weights \( w_k \) and the adjacency matrices constructed in the previous step
   \[
   F = \sum_{k=1}^{p} w_k A^k. \tag{1}
   \]
7. Normalize rows of \( F \) by dividing each element by the row sum to obtain the zone transition probability matrix.
8. return \( P_S = [p_{ij}] \), where
   \[
   p_{ij} = \text{probability of } (z_i \rightarrow z_j) = \frac{f_{ij}}{\sum_{j=1}^{t+1} f_{ij}}.
   \]

The Traveling Salesman Problem (TSP). Here, we provide a formalization of the TSP, which we will need in our proposed algorithms. Given a set of \( n \) vertices, with distance or cost \( c_{ij} \) associated to each vertex.
(i, j)-pair of vertices, the TSP consists of determining a minimum cost circuit passing through each vertex once and only once (Laporte 1992):

\[ \begin{aligned} \text{min} & \sum_{i \neq j} c_{ij} x_{ij} \\
\text{subject to} & \sum_{j=1}^{n} x_{ij} = 1 & i \in \{1, \ldots, n\} \\
& \sum_{i=1}^{n} x_{ij} = 1 & j \in \{1, \ldots, n\} \\
& \sum_{i,j \in S} x_{ij} \leq |S| - 1 & S \subset \{1, \ldots, n\}, \ 2 \leq |S| \leq n - 2 \\
\end{aligned} \]

where \( x_{ij} = \begin{cases} 1 & \text{if salesman travels from city } i \text{ to city } j \\ 0 & \text{otherwise.} \end{cases} \)

**Two-stage TSP algorithm.** Now, we are ready to introduce the main two-stage TSP algorithm (Algorithm 2). The goal here is to create a high quality route given the information on the instance—location and zone_id of each stop, station code \( S \), the travel time matrix provided—and the zone transition probability matrix \( P_S \) from Algorithm 1. The first step is to determine the set of all zones in the instance. This will determine the dimension of the zone cost matrix \( ZC \) in step 3. Next is to compute the approximate location of each zone. This will allow us to mix probability information from \( P_S \) with zone distances when building \( ZC \) in step 3. Stage 1 ends when we get the desired zone order after optimizing with \( ZC \), using the TSP formulation above. In the second stage, we build the stops cost matrix \( SC \) using travel time information between the stops and some zone_penalty, whose values are dependent on the zone order obtained during the initial stage. Finally, we again solve the TSP, this time using \( SC \) to get our desired sequence of stops.

**Zone location approximation.** Each stop in the historical data is geolocated along with the zone in which it belongs to. In order to approximate the location of each zone, we define the centroid of each zone \( z_i \). As the coordinates of the stops within a zone are close to each other, we treat the earth as being locally flat and determine the centroid by taking the average of the latitudes and the longitudes of all the stops.

**Zone penalty.** During the cost matrix construction in the second stage of Algorithm 2, we propose to add a zone_penalty, to the travel time \( tt_{ij} \) of each (i, j)-pair of stops. The goal here is to guide the optimization process towards producing a route which respects the zone order \( Z = \langle z_0, z_1, z_2, \ldots, z_0 \rangle \) obtained during the initial stage. Hence, zone_penalty, is set to 0 when stops \( i \) and \( j \) belong to the same zone. If the stops belong to succeeding zones in \( Z \), that is, if stop \( i \) belongs to \( z_k \) and stop \( j \) belongs to \( z_{k+1} \), with \( z_k, z_{k+1} \in Z \), then zone_penalty, is set to some parameter \( \mu \). Otherwise, zone_penalty, = \( \nu \), with \( \nu \gg \mu \).

In our actual implementation, we made some adjustments which resulted to further improvements in the results. The zone order condition was slightly relaxed by adding additional parameters—zone_penalty,
\( \delta < \mu, \) when the zone of stop \( j \) is closest to the zone of stop \( i \) regardless of the zone order; \( \text{zone\_penalty}_{ij} = \lambda, \) with \( \mu < \lambda \ll \nu, \) when stop \( i \) belongs to \( z_k \) and stop \( j \) belongs to \( z_{k+2}, \) etc.

### Algorithm 2: Two-Stage TSP

**Input**: Current instance with

- Stops information (location, \text{zone\_id})
- Station code, \( S \)
- Travel time matrix, \( TT = [tt_{ij}] \)

Zone transition probability matrix for station \( S, P_S = [p_{ij}], \) from Algorithm 1

**Output**: Route starting from and ending at station \( S, \) visiting each stop exactly once

1. **Stage 1.** From stops information, determine set of all zones \( Z \) that need to be visited.
2. Compute approximate location of each zone in \( Z \).
3. Build zone cost matrix \( ZC = [zc_{ij}]: \)
   \[
   zc_{ij} = d_{ij} - \alpha(p_{ij}), \quad (2)
   \]
   where \( d_{ij} \) denotes the distance between zone \( i \) and zone \( j. \)
4. Solve zone TSP by minimization using \( ZC \) as cost matrix to determine zone order.
5. **Stage 2.** Build stops cost matrix \( SC = [sc_{ij}]: \)
   \[
   sc_{ij} = tt_{ij} + \text{zone\_penalty}_{ij}, \quad (3)
   \]
6. Solve route TSP using \( SC \) as cost matrix to obtain final route.

### 4. Results and Conclusions

In order to evaluate the performances of our models, we implement Algorithms 1 and 2 in Python 3.8. To solve the zone (line 4) and route (line 6) TSPs of Algorithm 2 we use OR-Tools 9.0 with the default routing search parameters.

The construction of the zone transition probability matrix for each of the 17 unique stations is done during the \text{model\_build} phase. It is also during the \text{model\_build} phase that we approximate the location of each of the zones.

With the data generated from \text{model\_build}, we solve for the optimal route for each of the instances provided in the model evaluation setup during the \text{model\_apply} phase. Here, we evaluate three different models. The results of the evaluation are shown in Figure 1.

The first StandardTSP model denotes the standard approach, where we use OR-Tools to solve the TSP by minimizing the total travel time. We implement our proposed two stage approach initially as TwoStage1, where in step 3 of Algorithm 2 only zone transition probabilities are used to construct the zone cost matrix.
i.e., we have $z_{cij} = -(p_{ij})$ instead of eq. (2). Our approach as described exactly in Algorithm 2, where zone transition probabilities and distances between zones are used in constructing the zone matrix, is denoted by TwoStage2.

![Model Score Comparison](image)

<table>
<thead>
<tr>
<th>Model Score</th>
<th>Standard TSP</th>
<th>TwoStage1</th>
<th>TwoStage2</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>0.0979</td>
<td>0.0639</td>
<td>0.0508</td>
</tr>
<tr>
<td>min</td>
<td>0.0357</td>
<td>0.0096</td>
<td>0.0117</td>
</tr>
<tr>
<td>max</td>
<td>0.1634</td>
<td>0.1476</td>
<td>0.0910</td>
</tr>
<tr>
<td>model apply time (s)</td>
<td>149</td>
<td>132</td>
<td>136</td>
</tr>
</tbody>
</table>

Figure 1: Comparison of model scores and total computation times

From the evaluation (Figure 1), we learn that even the standard TSP is able to produce satisfactory results. A close inspection of the resulting routes, however, reveals that the routes can be improved further by making changes on the order by which the zones are visited. Indeed, applying TwoStage1, which learns the preferred zone order from historical data, on the test instances results to an improved model score. Finally, our proposed TwoStage2 approach, which uses not only the zone transition probabilities but also distance information in deciding on the zone order, is able to produce results that are generally better than the previous solution methods. We add a remark that we also experimented with only pure zone transition probabilities instead of zone penalty in eq. (3) of Algorithm 2. However, we observed superior performance when using zone penalty. We believe that this is because, with the zone penalty formulation, we put stronger restrictions on deviations from the zone order. Contrarily, the zone transition probabilities approach penalizes less severely when the zone order is not followed.

4.1. Discussion

Our approach is relatively simple and does not use other stop information than the zone, distances between zones, and travel times. The learning is hence fast and has a sparsification effect on the final cost matrix, allowing us to use a (potentially expensive) optimal TSP solver while still obtaining results relatively quickly. There is opportunity to improve both the learning of the zone ordering with more complex
probabilistic models (including neural networks), as well as what the zone penalty of each stop-pair is in this framework.
References


Article XXXI

Combined Exact and Heuristics Based Approach to Hamiltonian Path Problem Optimization for Route Planning

Fernando Hernandez, Rafael Sotelo and Marcelo Forets

Source code is available on GitHub:
https://github.com/AlphaCentauri21/RouteSequencing.jl
Abstract
Routing optimization problems have been present worldwide for years, even before the recent practical implementation of modern Internet which made a certain proliferation of the subject. Particularly, the cargo transport problem presents a grounded and realistic perspective on this matter, displayed as a complex, multi-stepped and layered logistics problem, which embodies fundamental challenges in vehicular routing, applicable for all terrestrial, aerial and maritime contexts. This paper describes an approach to the Amazon Last Mile Routing Research Challenge supported by the MIT Center for Transportation and Logistics. The proposed models aim to analyze this mathematical optimization problem by examining each custom layer and step of the routing process, detailing and justifying the entirety of its relevant combinations and testing ordering. Different solution approaches are compared and inspected within each other, reaching a level of mean prediction accuracy score of 0.112 when tested locally on the validation set.

1. Introduction

Package routing optimization consists of algorithms applied to routes with the objective of finding the sequence of stops for a minimal cost-effective performance of the network [Luo et al. 2020]. This does not necessarily entail selecting the shortest distance or fastest time route, but choosing from a compound of meaningful attributes. The different attributes that should be considered include a detailed understanding of the network characteristics, where a parameter hierarchy must be determined following an order cause-effect procedure by analyzing its possible logical combinations. The conceived model demands a process with heuristics in conjunction with optimization algorithms, producing an approximate interpretation to the routing problem in a reasonable time frame [Lin & Kernighan 1973].
In the next sections, a discussion over different solution strategies is performed both qualitatively and quantitatively, with the intention to analyze the resulting sequence to the optimal given. This technical paper begins by presenting a brief literature review on the routing problem, with a focus on formulating an initial abstraction to the problem. The Methodology section explains the layered, step-based approach followed in this article. The Analysis section specifies and intents to justify the main effects detected of the selected attributes on the model. Finally, the Results section concludes with the evaluation of the methodology on the challenge routes, and mentions some perspectives on future work.

2. Literature Review

Upon further examination, the problem studied is closely related to the Vehicle Routing Problem (Toth & Vigo, 2014), where the main objective is to find the minimum-length and minimum-time routes for a fleet of vehicles visiting a set of given locations. Another similar problem is the Quadratic Assignment Problem (Cela, 1998), fundamental in combinatorial optimization, where given the flow costs between facilities and distance weights between locations, the objective is to assign all facilities to different locations with the goal of minimizing the sum of the distances multiplied by the corresponding flows (e.g. in a custom delivery system). Both of these problems are considered NP-complete or NP-hard (Mehlhorn, 2012), meaning the problem is not solvable in polynomial time, explaining the reason why this is still an active research subject. In fact, possibilities are being analyzed for alternate studies such as in Quantum Computing (another recent proliferating area) that intends to propose a solution to this problem through Quantum Annealers (Borowski et al., 2020).

Moreover, it is possible to connect the proposed problem to the well-known Traveler Salesman Problem (TSP) (Kara & Derya, 2015), instance of the Vehicle Routing Problem which has their same objective of finding the shortest yet most effective route of a network but uses a single transport instead of several routes. This NP-complete problem was first formulated in 1930 and has been very intensively studied and worked upon, due to its application not only in routing logistics but also in other areas like microchip manufacturing electronics and DNA sequencing. Several generalized approaches have been developed for this problem, including standard logic algorithms such as greedy (like nearest neighbour) and/or with dynamic programming, and constructive heuristics based approximations like multi-fragment (Ahiod et al., 2017) and different variable-opt techniques (Punnen et al, 2003). However, none of these possibilities reaches a global solution in polynomial time, and are deemed not applicable to the current problem, given the extended amount of stops in each route and their interrelationships.

An additional consideration to better approach the problem at hand is to consider it as a Hamiltonian Path Problem (Libura, 1991), which is a sub-problem of the TSP. Hamiltonian problems have proven to be NP-complete, hence it is computationally intractable to find a global optimal sequence with the added restriction of only stopping once per stop and not returning after the last stop. Taking the perspective of graphs, both directed or undirected, where the stops are represented by its nodes, the idea is to find the
shortest traceable path with no cycles which makes the graph Hamiltonian-connected, i.e. for every pair of vertices there is a unique path between them. The mentioned cycles exist when the endpoints of a path are adjacent, although any Hamiltonian cycle can be converted to a Hamiltonian path by removing one of its edges. Solving this problem by exhaustive search is not feasible, as the number of different Hamiltonian cycles in a complete undirected graph on $n$ vertices (number of stops) is $\frac{(n-1)!}{2}$ and in a complete directed graph on $n$ vertices is $(n-1)!$. This enumeration assumes that cycles that are the same apart from their starting point are not counted separately [Balakrishnan & Ranganathan, 2012]. Figure 1 shows a scheme comparing the Vehicle Routing and Hamiltonian path problems, using stop names for a certain route and which belong to a given zone.

In sum, the proposed objective consists of designing an applicable heuristic-guided approach for the Hamiltonian Path Problem following similar philosophy as previously mentioned formulations, where the approximate solution should be achieved through a non-exponential order algorithm.

3. Methodology

The selected procedure involves a layered approximation to the logistic problem guided by an incremental and iterative process as illustrated in Figure 2. Given a subset of relevant features, the process starts by aligning the attributes variable possibilities, then designing an algorithm with the chosen parameters combination and finally validating with the proposed routes as well as integrating and comparing results between different routes.

Due to the fact that the work is also code-based, an initial step of programming an interface to work with the problem is required. The choice of language was Julia given its extensive package ecosystem and familiar syntax for applications in Optimization [Bezanson et al, 2017]. In particular, the JuMP library for mathematical optimization is used [Dunning et al, 2017].

The initial step of the implemented methodology relies on formally understanding the problem and its
objective, while searching the literature for similar issues and possible standard guidelines to follow and/or use. This implies knowing the given attributes and its meaning and relationships, as well as schematizing the descriptive information on maps and charts in order to grip a proper clarity on how and why the filtered attributes are selected. The Amazon Last Mile Routing Challenge supported by the MIT Center for Transportation and Logistics provided experienced drivers data of thousands of real routes made by Amazon delivery trucks in different cities of USA. Based on this data, the challenge consists in building a model to predict the order in which real drivers would go through certain given stops. The objective is thus not to discover the shortest path or that which takes less time, but to build a system that would decide the route and the order of the stops like an experienced driver.

Then, the process splits on simultaneous tasks of data segmentation and attribute selection, where the former delineates the characteristics to use and the latter chooses and classifies from an already filtered list of parameters. Both tasks demand a structured methodology, guided by a layered approach based on routes, which are defined by sequences of zones and then stops, as shown in Figure 3.

Continuing the process, the conceptual relevance of the different model parameters is defined for the segmented and selected parameters. A hierarchy is elaborated to order and mirror the arrangement of decisions taken by the actual sequences. Once the theoretical considerations are stipulated, it follows the practical testing and formulation of necessary algorithms to accommodate for the possible combinations analyzed, with the possibility to recur to attributes modifications and go back in the flow process if needed.

Figure 2: Step-based methodology designed for the heuristic approach solution to the logistics problem.
A common practice when setting these algorithms is to perform an iterative approximation process which begins with an initial estimate with the selected actual objective properties, in order to have them as guide for the best possible score in the step. Finally, the obtained results are compared and analyzed, completing the process iteration and returning to a previous step if the final score does not satisfy the tolerance provided to the best score defined in the previous step.

4. Analysis

In order to model and define proper heuristics when testing all training and validation routes, it is relevant to consider the data segmentation of routes, zones, stops and packages. This selection of the model parameters is based on the visualization of a reduced set of routes. Its acceptance is conditioned to the performance of the generality of the routes with the provided score metric. An initial observation reveals that the real sample sequences adhere to this premise: all the stops in a given zone are visited before going to another zone. This can be seen in Figure 4 where a sample sequence is displayed. Each point is a stop, and is coloured according to the zone it belongs. Yellow lines show the sequence of the route and the number labels are the order of visit of the stop. The approach is based on this hierarchy and estimates the zone order and then the stops order inside each zone (hereinafter generically called *cluster*). 

The presented model, schematized in Figure 5, encapsulates given data segments properties into decision parameters or logical points where to select one of possible approaches to the routing problem. The hierarchy of these parameters is defined mainly empirically, while theoretically trying to diminish the available use cases. Regarding the parameters definition, as illustrated in Figure 6, first it is decided the mechanism to
differentiate or cluster the stops in groups (i.e. stops clustering), based on the fact that zones and stops can have same identifier but different geographic locations. Afterwards, a protocol for sequencing inside these clusters is determined, for then to define how the clusters themselves are ordered based on distance measurements between each group representative points. Continuing, the next decision is regarding which selected criteria to use in order to connect the clusters (i.e. next stop), and finally a contemplation of which algorithm step is needed for including time restrictions and capacity limitations.

Regarding the stops clustering method (SC), the decision remains on whether to group stops by already defined zones codes (SC1) or using alternative methods like density-based (Gonzalez et al., 2007) (i.e. judging on the distance metric and criterion for a minimum number of data points) or fuzzy clustering (Ewbank et al., 2016) (i.e. where one data point can belong to one or more clusters), named SC2. The option to avoid clustering (SC3) is also an alternative, which demand an analysis of the stops all together and certainly more processing time.

With respect to the travel times sequencing method (TTS), one possible approach consists on deciding locally (TTS1, i.e. on the spot) the nearest stop of the sequence based on the given travel times or cost matrix per route, demanding less computational resources but lacking on route layout generality. Counterpointing this approach is the possibility to perform global searches, both exhaustive (TTS2) or restricted (TTS3), calculating the minimum travel times of the entire cluster. The restricted version of finding the maximum possible flow on a cluster follows the Ford-Fulkerson algorithm (L.R. Ford & Fulkerson, 1956) with particular restrictions to determine the minimum path for a fixed initial and final stop, iterating the latter and keeping the option with the global lesser cost.

The task TTS is thus solved by exact methods for global approaches (TTS2, TTS3), and by heuristics
Figure 5: Considered assignments and approaches specification indicating possible perspectives per decision turning point.

<table>
<thead>
<tr>
<th>TASK</th>
<th>POSSIBLE METHODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stops clustering (SC)</td>
<td>By zones (1)</td>
</tr>
<tr>
<td></td>
<td>Density-based (2)</td>
</tr>
<tr>
<td></td>
<td>No clustering (3)</td>
</tr>
<tr>
<td>Travel times sequencing (TTS)</td>
<td>Local approach (1)</td>
</tr>
<tr>
<td></td>
<td>Global restricted approach (2)</td>
</tr>
<tr>
<td></td>
<td>Global exhaustive approach (3)</td>
</tr>
<tr>
<td>Cluster sequencing (CS)</td>
<td>Using Euclidean distance (1)</td>
</tr>
<tr>
<td></td>
<td>Using Hausdorff distance (2)</td>
</tr>
<tr>
<td>Next stop definition (NS)</td>
<td>Last inner stop (1)</td>
</tr>
<tr>
<td></td>
<td>First outer stop (2)</td>
</tr>
<tr>
<td>Time window conditionality (TWC)</td>
<td>Defined at clustered stops (1)</td>
</tr>
<tr>
<td></td>
<td>Defined globally (2)</td>
</tr>
<tr>
<td>Capacity limitations (CL)</td>
<td>Per clustering division (1)</td>
</tr>
<tr>
<td></td>
<td>Upon max-saturation (2)</td>
</tr>
</tbody>
</table>

Figure 6: Potential hierarchical tree schema considered for routing optimization, following logical cases. Numbers and names are those defined in Figure 5.
methods for local approach (TTS1). This combined perspective is deemed necessary in order to fulfill most relevant approaches given determined tasks. To decide which to finally use, the score of each method for validation routes is evaluated with the metric provided. Regarding the exact method formulation, consider a travel times square matrix $T$ per cluster with $N$ elements and a binary decision square $N \times N$ matrix $X$ with components $X_{s,t}$ with a value of 1 if stop $s$ links to stop $t$ and 0 otherwise. The restrictions to minimize the cost function $\sum_{s=1}^{N} \sum_{t=1}^{N} X_{s,t} \ T_{s,t}$ are presented next.

- $X_{1,N} = 0$, given the unavailability to transport from the first to last stop in the cluster.
- $X_{s,1} = 0 \ \forall s \in [1,N]$, as no intermediate node should reach the initial stop.
- $X_{s,s} = 0 \ \forall s \in [1,N]$, as no self-loops are allowed.
- $X_{N,t} = 0 \ \forall t \in [1,N]$, as no intermediate node is allowed after the last stop.
- $\sum_{t=2}^{N} X_{s,t} = 1 \ \forall s \in [1,N-1]$, as each starting stop links to only one termination stop (except the last one).
- $\sum_{s=1}^{N} X_{s,t} = 1 \ \forall t \in [2,N]$, as each termination stop links to only one starting stop (except the initial one).
- $X_{s,t} + X_{t,s} <= 1 \ \forall s, t \in [1,N]$, as no repetitions are needed.

If the optimal solution does not contain a unique connected path, additional constraints are added to forbid such solutions, and the optimization is solved again until the maximum length path is found.

The cluster sequencing method (CS) is only considered if clustering is utilized. A method needs to be defined so to arrange spatially the stops group order in which the cargo needs to be transported. The criterion is to order these clusters by minimum distance, using either the group geometrical baricenter considering the geographical position of its components (CS1), or the Hausdorff distance metric (Zhu et al., 2009) while prompting to find the greatest of all the lengths from a point in one set to the closest point in another (CS2). Due to the relative short distances managed, these approaches do not consider added parameters consequence of the differences on earth curvature, which clearly should be the case for long maritime or aerial transports.

It is also necessary to determine a method to transition between clusters (NS). One possible method considers the last stop of the current cluster (NS1, i.e. last inner stop) as the first stop of the next cluster sequence. Other approach is defining a local filtered approach to decide the immediate next stop after the last inner stop and performing the cluster sequence arrangement beginning on the first stop of the next group (NS2, i.e. first outer stop). Both options, albeit similar, are found to have a considerable impact on the result.

The time window conditionals (TWC) mandates the definition of sequence boosts and/or delays for those stops with a casement defined. In consequence, the current time is necessary to be maintained at each stop,
calculated by the route departure time and the sum of the traversed travel times between stops and planned service time per node. This normal flow modifications are possible to be determined at each clustered stop while keeping the groups order unaltered (TWC1, i.e. locally) or alternatively stating conditionals on each node, cluster and parameter decision (TWC2, i.e. globally). Similarly, the decision to resupply at station due to transport capacity exceeded (CL) can be concluded by predicting the packages dimensions needed per cluster and judging before accessing the next cluster (CL1), as well as determining accumulated package volume and comparing it with maximum transport capacity on a per stop basis (CL2, i.e. until max-saturation). Each of these perspectives, albeit apparently subtle, provoke considerable differences on final sequence and therefore need to be taken into account for optimization.

Figure 7 shows sample algorithm flows using the aforementioned parameters and approaches combinations, with a purposely considerable variation between each other. As can be appreciated, algorithms lengths are combination dependable. In order to define computationally efficient procedures, iterations are minimized and decisions are performed sequentially avoiding repetitions and double checks.

5. Results and Conclusions

The algorithms develop logically different effects due to the distinct perspectives on which the routing optimization problem is examined. These approaches are analyzed both through mean as well as route-specific score results and graph layout, as shown on Figure 8. Thus, given the considered use cases quantities, several general decisive conclusions can be ruled for the utilized base routes.

Globally, the methods decisions on more overarching tasks (e.g. SC and TTS) cause a higher score differentiation than specific ones (e.g. TWC and CL). In particular, with regards to the stops clustering determination, considering no clustering SC3 or seeking guidance through already defined zones SC1, not necessarily mapped to any neighbour or county, resulted on the better approaches. Specially, the former give the best results yet tested on the validation dataset, with a mean score top of 0.112 while the latter remain with a mean score bottom of 0.828. Then, the travel times sequencing derive on better outcomes mainly on local approach TTS1 but also using global brute-force per clusters TTS3 in those with less than ten stops, with a mean score difference of 0.542 between each method. With respect to cluster sequencing, Hausdorff distance incorporation CS2 performed better than Euclidean distance CS1 by a mean score difference of 0.373, possibly by closely mirroring intuitive ordering. For the next-stop definition, a considerable better mean score gap of 0.128 result while considering last inner stop NS1 option, using the last stop of previous cluster to analyze the next cluster stop sequence. Finally, both time window and capacity limitations grant better results when examined locally (TWC1 and CL2, i.e. defined at cluster stops and upon max-saturation respectively) by close variation. Apparently, as observed on the validation dataset, routes that unfold along a predominant direction result in better scores than other equally expanded geographically, favored by a more clearly procedural solution on the sequence definition.

Regarding computation time, all defined approaches converge in seconds for given validation routes. As
Figure 7:  (a) Usage of no stops clustering SC3 and local travel times sequencing TTS1, both options counteracting the possible over-increment on needed memory and processing time. It also utilizes a global approach to time windows TWC2 and a max-saturation per stop perspective to capacity limitations CL2. (b) Utilization of local time sequencing TTS1 and first outer next-stop definition NS2, as well as per stops time window considerations TWC1 and also max-saturation per stop for capacity limitations CL2. (c) Application of global time sequencing TTS2 and last inner next-stop definition NS1 in addition to cluster-level time window TWC2 and capacity limitations decisions CL1.
en exception, the running time of methods involving TTS3 are larger than the previous methods, and are not applicable for routes with more than 10 stops. Thus, route characteristics such as its clusters quantity and the amount of stops per cluster are relevant factors which impact sequence performance.

The results are then varied and not uniformly distributed on all considered routes, with clear satisfactory accuracy for routes with disjoint and distanced clusters and with neither time window indication nor applicable capacity limitations. The better scores are not dependent either on geographical considerations such as where the route is located or how densely built or populated the considered clusters are.

Considering the mean predicted score on given routes to evaluate the algorithms with increased processing speed as having no training step, the better results are obtained when using a no stops clustering approach and local travel time sequencing, using consequently a global perspective on time windows and capacity restrictions. Curiously, this mean score of approximately 0.112 is result of one of the simplest approaches combinations detected according to the previously stated paradigm, in spite of unnecessary turnings and double crossings detected once analyzed graphically.

Regarding future research on this combined approach with respect to this specific problem, continuing the delimited investigation cases, may consist on meticulously classifying the routes layout on which some models worked better with a given approach, and propose a mixture of current algorithms determined by these particular layout characteristics.

Remarkably, when using the proposed approaches with given actual cluster sequences of validation routes, the mean score is as low as 0.011. This proves that this perspective is particularly feasible for local decisions inside a zone. Future research can then also be focused on involving a learning algorithm to predict cluster sequences by zones based on historical data.
References


MIT (2021). Amazon last-mile routing research challenge | supported by the MIT center for transportation logistics. URL: https://routingchallenge.mit.edu/

