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:

   (a) Nearest Neighbor
   (b) Farthest Insertion
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 backtrackings, 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).

$$tw_i = [max(a_{p_{i,0}}, \ldots, a_{p_{i,n}}), min(b_{p_{i,0}}, \ldots, b_{p_{i,n}})] \tag{1}$$

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 *time window* to model tw_p. Accordingly, *start time utc* and *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 *start time utc* and *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).

$$s_i = \sum_{p \in P_i} s_p \tag{2}$$

Finally, we impute potential *NaN*-values in the *zone id* data field of the stops. The imputation logic depends on the *type*-attribute of the stop with *zone id* = *NaN*:

- If *type* = *Station*, replace *NaN* by an artificial 'Start' zone ID
- If *type* = *Dropoff*, replace *NaN* by the zone ID of the closest stop with *zone id* $\neq$ *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

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 a 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.

$$f = (1 - w) \cdot dist + w \cdot v \tag{3}$$

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.

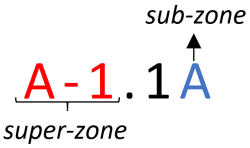

Figure 1: Super-zone, sub-zone definition

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.

$$\text{H-10.2H} \rightarrow \text{H-10.3H} \rightarrow \text{H-10.3J} \rightarrow \text{H-10.2J} \rightarrow \text{H-10.1J} \rightarrow \text{H-9.1J} \rightarrow \text{H-9.2J} \rightarrow \text{H-9.3J} \rightarrow ...$$

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 $change_{S,S'}$ and calculated as:

$$change_{S,S'} = \begin{cases} (1 - w_1) \cdot (dist_{S'} - dist_S) + w_1 \cdot m \cdot (v_{S'}^{sp} - v_S^{sp}) & if v_{S'}^{sp} - v_S^{sp} \neq 0 \\ (1 - w_2) \cdot (dist_{S'} - dist_S) + w_2 \cdot m \cdot (v_{S'}^{sb} - v_S^{sb}) & otherwise \end{cases} \tag{4}$$

where:

- $dist_{S'}$ and $dist_S$ 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 → **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-10.3**H** → H-10.3**J** 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 $change_{S,S'}$ 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 *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

procedure ADJUST($tour$)
  for $i = 1$ to $len(tour) - 1$ **do** ▷ not iterate over the *Start* zone
  ▷ (i) Case of violation to **sub-zone rule**
    if $tour[i]$ and $tour[i+1]$ have **same super-zone codes then**
      if $tour[i]$ and $tour[i+1]$ have **different sub-zone codes then**
        $tour \leftarrow$ 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 $tour[i]$
  ▷ (ii) Case of violation to **super-zone rule**
    else
      $tour \leftarrow$ insert all zones, which are from the index $i+2$ and have the same super-zone codes as those of $tour[i]$, after $tour[i]$
  return $tour$

Algorithm 2 Iterative 2-Opt

procedure ITERATIVE2OPT($starting\_tour, w_1, w_2, max\_allowed$) ▷ $starting\_tour$ obtained from NN
  $starting\_tour \leftarrow$ 2opt($starting\_tour, w_1, w_2$) ▷ local search: 2-Opt using (4)
  while ¬ termination criteria **do**
    $new\_tour \leftarrow$ adjust($starting\_tour$); ▷ improve spatial structure
    if distance(new_tour) - distance($starting\_tour$) $\leq max\_allowed$ **then** ▷ new solution accepted
      $starting\_tour \leftarrow new\_tour$; $changed \leftarrow$ FALSE; ▷ get the new tour and exit
    else
      $new\_tour \leftarrow$ 2opt(new_tour, w_1, w_2) ▷ local search with new starting solution
  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:

$$f(t_{final}, d_{total}) = w_3 \cdot t_{final} + (1 - w_3) \cdot d_{total} \tag{5}$$

with $final$ 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:

$$t_i = max(a_i, t_j + c_{j,i} + s_i) \tag{6}$$

with j being the preceding customer in the delivery tour.

The metric d_{total} to measure delayed delivery, in turn, is given by (7):

$$d_{total} = \sum_{i \in I} max(a_i - t_i, 0), \tag{7}$$

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 *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 ≥ 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 ≥ 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 ($w_2 = 0, w_3 = 0, max\_allowed = 600$). After tuning, we obtain the best value of w_1, denoted as w_1^*.

(2) w_2: the search range is based on the best value of w_1 and defined as $[0,\ 0.5 \cdot w_1^*,\ w_1^*]$. Other parameters are fixed ($w_1 = w_1^*, w_3 = 0, max\_allowed = 600$). The best value of w_2 after tuning is w_2^*.

(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 $w_1, w_2, max\_allowed$ are fixed at $w_1^*, w_2^*, 600$ respectively. The best value of w_3 after tuning is w_3^*.

(4) *max_allowed*: three values 150, 300, 600 are selected for tuning. Parameters w_1, w_2, w_3 are fixed at w_1^*, w_2^*, w_3^*, 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

Subset	No. of routes	w_1	w_2	w_3	$max\_allowed$	**Avg. score**
Subset 1 (1 super-zone)	497	0.9	0.9	0	300	**0.049446**
Subset 2 (2 super-zones)	1928	0.8	0.8	0	150	**0.049655**
Subset 3 (at least 3 super-zones)	293	0.8	0.4	0	300	**0.062585**
All high-score routes	2718	-	-	-	-	**0.051010**

Table 1: Average score and best parameter set on 2718 high-score routes

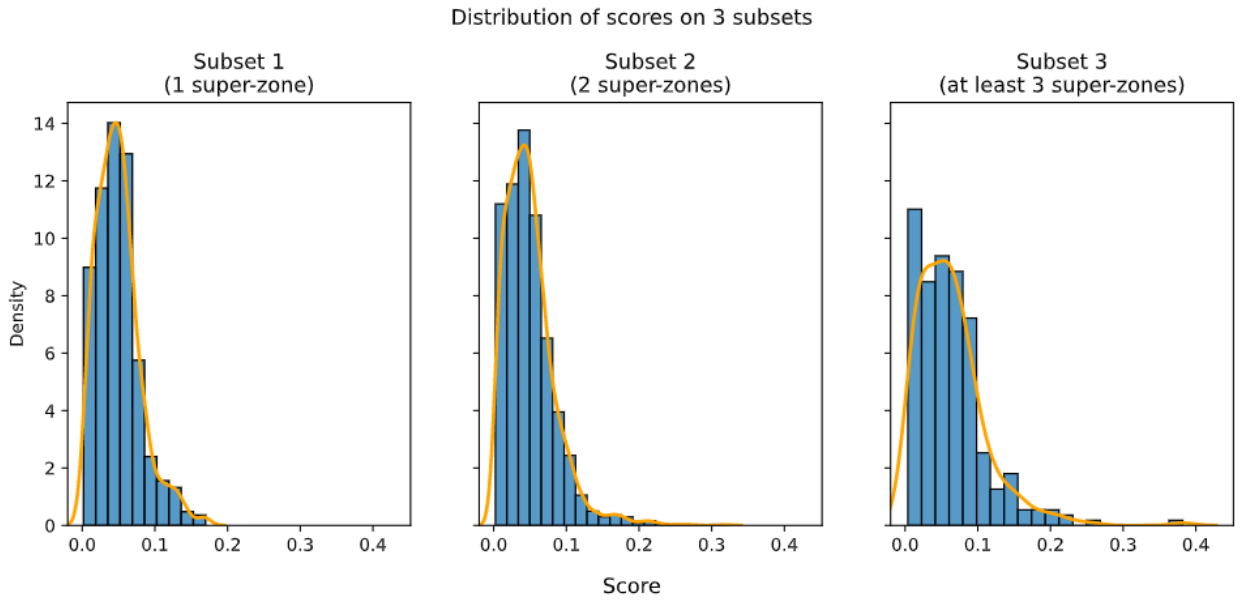

Figure 2: Distribution of scores on each subset of high-score routes

least 3 super-zones, has the highest average score, which is much higher than that of the other two subsets. 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. Consequentially, 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

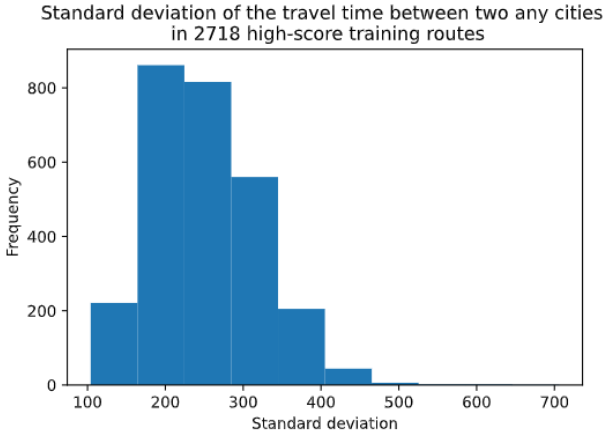

Figure A.3: Distribution of travel time in the high-score routes in the training data

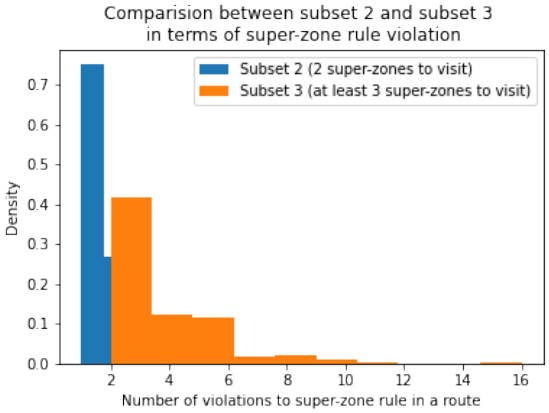

Figure A.4: Difference in importance of super-zone rule adherence between route set with 2 and at least 3 super-zones to visit

References

Blank, Z. (2021). What drivers can learn from amazon's efficiency and route optimization. URL: https://www.getstraightaway.com/blog-posts/what-drivers-can-learn-from-amazons-efficiency-and-route-optimization.

Cook, W. J., Applegate, D. L., Bixby, R. E., & Chvátal, V. (2007). *The Traveling Salesman Problem*. Princeton University Press. URL: https://doi.org/10.1515/9781400841103. doi:10.1515/9781400841103.

Schrijver, A. (2005). On the history of combinatorial optimization (till 1960). In *Discrete Optimization* (pp. 1–68). Elsevier. URL: https://doi.org/10.1016/s0927-0507(05)12001-5. doi:10.1016/s0927-0507(05)12001-5.