CIRRELT
1971 - 2021

CIRRELT-2021-30

Data-driven Vehicle Routing in Last Mile Delivery

Okan Arslan
Rasit Abay

August 2021

Bureau de Montréal
Université de Montréal
C.P. 6128, succ. Centre-Ville
Montréal (Québec) H3C 3J7
Tél : 1-514-343-7575
Télécopie : 1-514-343-7121

Bureau de Québec
Université Laval,
2325, rue de la Terrasse
Pavillon Palasis-Prince, local 2415
Québec (Québec) G1V 0A6
Tél : 1-418-656-2073
Télécopie : 1-418-656-2624

Data-driven Vehicle Routing in Last Mile Delivery

Okan Arslan[1, *], Rasit Abay[2]

1. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and, Department of Decisions Sciences, HEC Montréal, QC, Canada
2. School of Engineering and Information Technology, UNSW Canberra, Northcott Drive, Campbell, ACT, Australia

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. This method received the third place in the routing challenge.

Keywords: routing, last-mile, traveling salesman problem

---

* Corresponding author: okan.arslan@hec.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2021

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 3^{rd} or 4^{th} 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

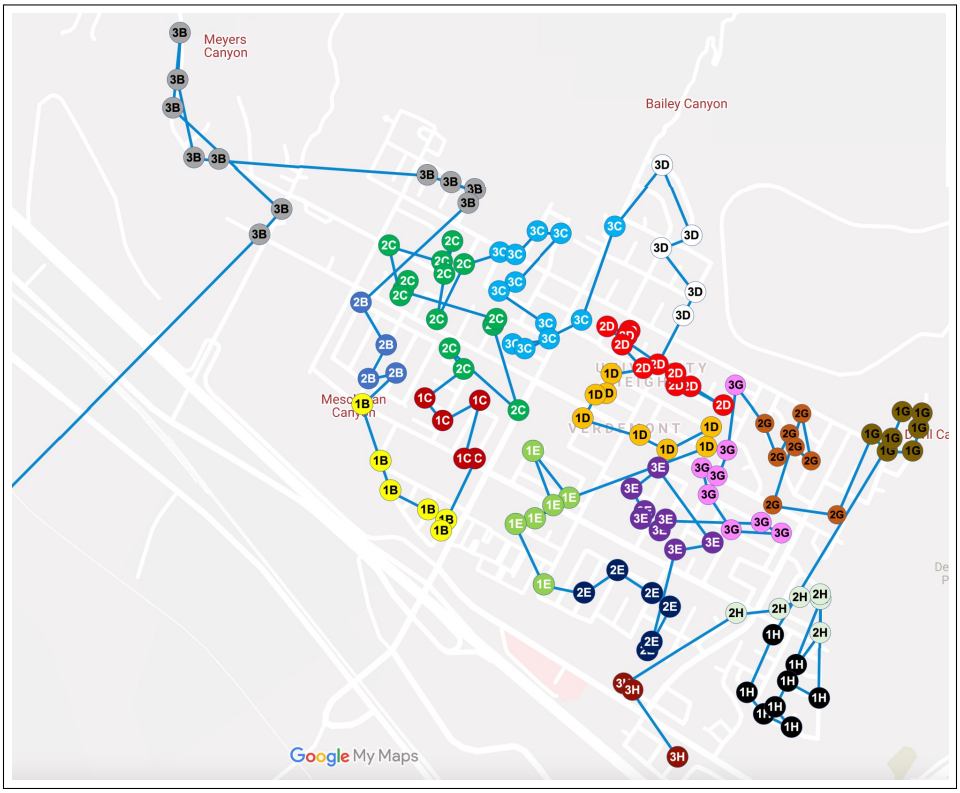

Figure 1: Map of actual route 'RouteID_62955f5e-57a2-4425-bfe6-6989d8dae565'. Stops are marked by their zone_ids and the route is shown in blue line. The depot is not shown on this map. Google Map data ©2021

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 $x_{ij}^k = 1$ if the kth 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 $\eta_{ij} = \sum_{k=1}^{k=4} x_{ij}^k$ to be the 'token difference of nodes i and j', which is a parameter to measure the dissimilarity between the zone_ids of the two given nodes.*

Having $\eta_{ij} = 0$ implies that nodes i and j share the same zone_id. Table 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 another stop in zone *A-1.3A*. In other words, the consecutive visited zones share 3 tokens 17.38 times on average, which corresponds to 11.9% of the average number of nodes. This case is particularly important and we now further investigate such zone changes.

Table 1: Average number of times that a token difference appears between two consecutive stops on actual routes

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

† Excluding the arcs from the depot

When the token difference equals 1 in Table 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.

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.

[*] https://en.wikipedia.org/wiki/Military_Grid_Reference_System

Table 2: Average number of times that two zone_ids are different only in i^{th} token for $i = 1, \ldots, 4$ (when the token difference equals 1 in Table 1).

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

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 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 as the actual route, neglecting the time windows insignificantly impacts the score. This modeling choice aligns with our modeling strategies presented in Section 1.

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

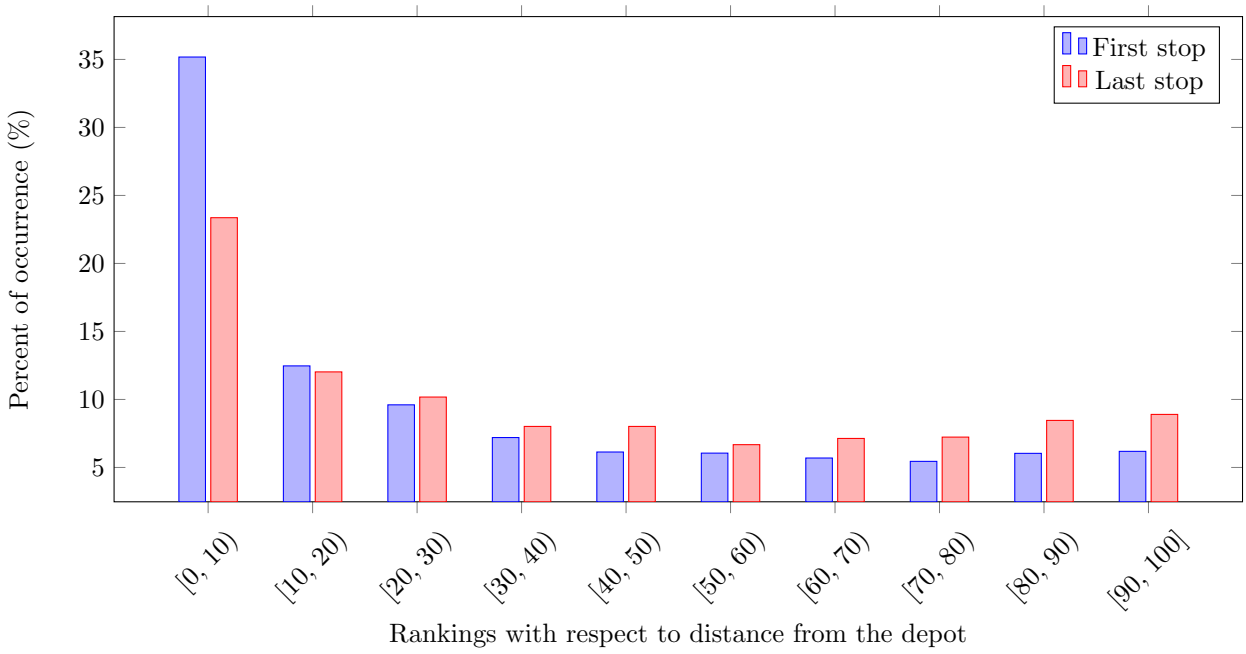

Figure 2: The first node and the last stops on the actual routes are not always the nearest two stops to the depot. This plot shows the ranking distribution of the first and last stops in terms of distance from the depot.

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 licencing 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 kth token of the *zone_id* of nodes i and j are different, and 0 otherwise. Consider graph $\hat{G} = (N, \hat{A})$, where $\hat{A} = A$ and the cost of arc (i, j) is equal to t_{ij}, which is defined as follows:

[‡] https://github.com/sugia/GA-for-TSP

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

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. 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 $maxNumOfTrial = 1$, $Npop = 50$ 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 $(b_1, b_2, a_1, a_2, a_3) = (0, 1, 1, 10, 100)$.

Algorithm 1: Route generating algorithm

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

Output: *proposed_sequences.json*

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

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.

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

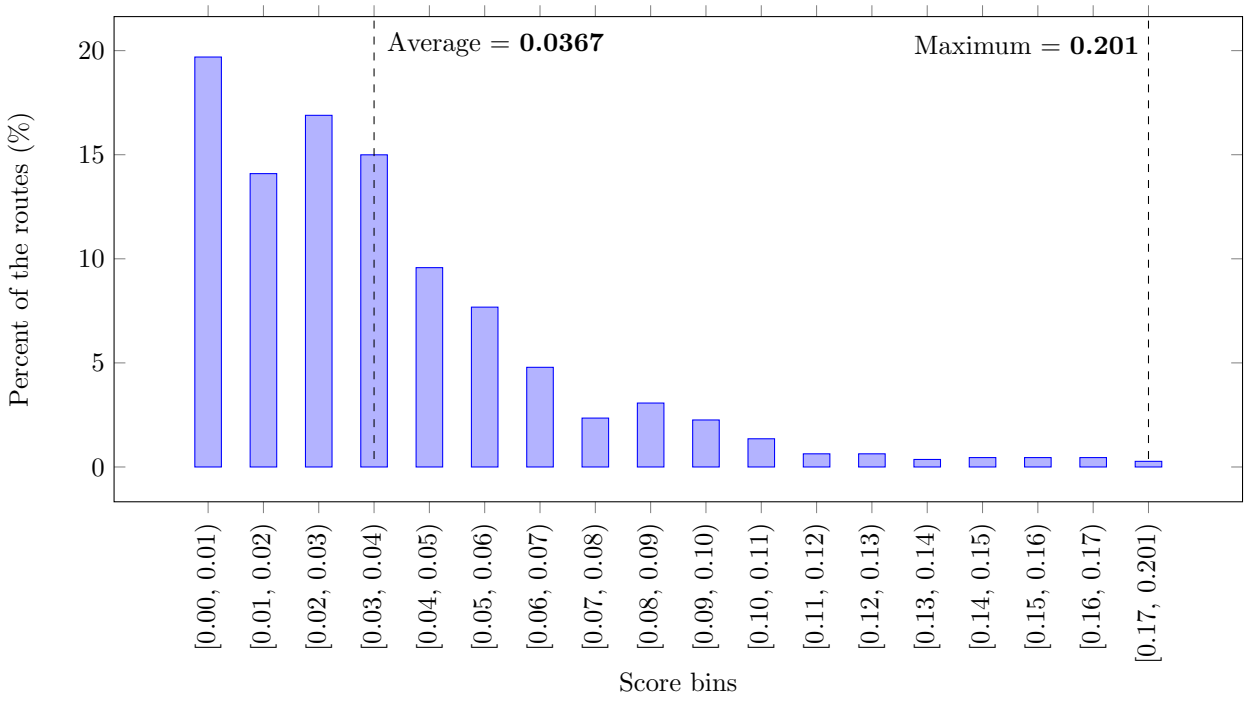

Figure 3: Score distribution on 1107 *high* routes without 'DELIVERY_ATTEMPTED' status on any stop. These routes share similar characteristics as in the model apply phase. The average score is **0.0367** and the maximum score is **0.201**.

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.
- The departure times change between 10:48:03 and 18:27:10.
- 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

Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2011). *The traveling salesman problem*. Princeton university press. doi:10.1515/9781400841103.

Bossek, J., Hoos, H. H., Kerschke, P., Kotthoff, L., Neumann, A., Neumann, F., Trautmann, H., & Wagner, M. (2021). Tsp algorithm selection. URL: https://tspalgsel.github.io last accessed on Jun 22, 2021.

Carrabs, F., Cordeau, J.-F., & Laporte, G. (2007). Variable neighborhood search for the pickup and delivery traveling salesman problem with lifo loading. *INFORMS Journal on Computing*, *19*, 618–632. doi:10.1287/ijoc.1060.0202.

Helsgaun, K. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*, 106–130. doi:10.1016/S0377-2217(99)00284-2.

Hore, S., Chatterjee, A., & Dewanji, A. (2018). Improving variable neighborhood search to solve the traveling salesman problem. *Applied Soft Computing*, *68*, 83–91. doi:10.1016/j.asoc.2018.03.048.

Jonker, R., & Volgenant, T. (1983). Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, *2*, 161–163. doi:10.1016/0167-6377(83)90048-2.

Jonker, R., & Volgenant, T. (1986). Transforming asymmetric into symmetric traveling salesman problems: erratum. *Operations Research Letters*, *5*, 215–216. doi:10.1016/0167-6377(86)90081-7.

Knox, J. (1994). Tabu search performance on the symmetric traveling salesman problem. *Computers & Operations Research*, *21*, 867–876. doi:10.1016/0305-0548(94)90016-7.

Kotthoff, L., Kerschke, P., Hoos, H., & Trautmann, H. (2015). Improving the state of the art in inexact tsp solving using per-instance algorithm selection. In C. Dhaenens, L. Jourdan, & M.-E. Marmion (Eds.), *Learning and Intelligent Optimization* (pp. 202–217). Cham: Springer International Publishing. doi:10.1007/978-3-319-19084-6_18.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, *59*, 231–247. doi:10.1016/0377-2217(92)90138-Y.

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, *21*, 498–516. doi:10.1287/opre.21.2.498.

Liu, S. (2014). A powerful genetic algorithm for traveling salesman problem. *arXiv preprint arXiv:1402.4699*.

Nagata, Y., & Kobayashi, S. (2013). A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing*, *25*, 346–363. doi:10.1287/ijoc.1120.0506.

Operations Research Group Bologna (2021). Tsp software. URL: http://or.dei.unibo.it/research_pages/tspsoft.html last accessed on Jun 22, 2021.

Pepper, J. W., Golden, B. L., & Wasil, E. A. (2002). Solving the traveling salesman problem with annealing-based heuristics: a computational study. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, *32*, 72–77. doi:10.1109/3468.995530.

Potvin, J.-Y. (1996). Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, *63*, 337–370. doi:10.1007/BF02125403.

Skiena, S. (2021). Stony brook algorithm repository, traveling salesman problem. URL: https://algorist.com/problems/Traveling_Salesman_Problem.html last accessed on Jun 22, 2021.

Wang, Z., Geng, X., & Shao, Z. (2009). An effective simulated annealing algorithm for solving the traveling salesman problem. *Journal of Computational and Theoretical Nanoscience*, *6*, 1680–1686. doi:10.1166/jctn.2009.1230.