Strategic API Calls to Retrieve the Cost Matrix in the Travelling Salesman Problem

Hesam Rashidi*[1], Mehdi Nourinejad[2], Matthew Roorda[1]
[1]Civil and Mineral Engineering, University of Toronto, Canada
[2]Civil Engineering, York University, Canada

1 INTRODUCTION

Routing in last-mile deliveries is typically formulated as a Travelling Salesman Problem (TSP), where an algorithm (e.g., nearest neighbour heuristic) uses a cost matrix, such as a travel time matrix, to determine a near-optimal or optimal tour starting and ending at a depot while serving all customers. The quality of TSP solutions depends on the routing algorithm and the accuracy and practicality of the input cost matrix. Common approaches to generating the cost matrix for last-mile routing include (1) Euclidean distances, (2) pre-trained machine learning models, and (3) third-party API services.

Euclidean distances are computationally efficient but do not capture real-world travel times or dynamic conditions such as traffic or closures. Pre-trained machine learning models offer more accurate estimates but require extensive historical data for training and do not reflect real-time traffic conditions. Third-party API services, such as Google Maps and Mapbox, provide near real-time travel time estimates. However, these services come with limitations, including rate limits on requests, matrix size restrictions, and variable pricing models based on the volume of data retrieved. For example, Mapbox allows a maximum of 10 input coordinates per call and 30 calls per minute, while Google Maps permits 25 input coordinates per call and up to 600 calls per minute, with a limit of 100 elements per server-side request (Google, 2025; Mapbox, 2025). These constraints can create bottlenecks in solving large-scale TSPs.

This study demonstrates that making strategic API calls can reduce costs and processing time while still achieving near-optimal solutions to the TSP. We developed a data-driven framework that operates without requiring pre-training, making it adaptable and applicable in near real-time scenarios. Rather than retrieving the travel times for all possible pairs of stops, which is often infeasible for large problem instances, the framework prioritizes the retrieval of travel times only for those pairs of stops that are most likely to be visited consecutively in the optimal TSP solution.

2 METHODOLOGY

Consider a complete digraph $G(V, A)$ representing a TSP instance, where $V = \{v_0, v_1, ..., v_n\}$ shows the set of depot and delivery points and an arc $a_{ij} \in A$ connects node v_i to node v_j for all $v_i, v_j \in V$. The travel time between the stops is given by a cost matrix $T = [t_{ij}]$, where t_{ij} denotes the travel time over arc $a_{ij} \in A$. Note that T is asymmetric, non-metric, and the travel time from any delivery point to itself is zero. A feasible TSP tour $s \in S$ is a sequence that starts at v_0 (i.e., depot), visits each node $v_i \in V \setminus \{v_0\}$ exactly once, and returns to v_0, where S denotes the set of all possible feasible tours. The optimal TSP tour, denoted by $s^* \in S$, is a feasible tour with minimal total travel time. Classical routing algorithms use a complete cost matrix (e.g., the travel time matrix T) to solve s^* for a TSP instance. In the scenario under study, it is assumed that the cost matrix T is constructed by making API calls to retrieve the travel times for the arcs. The following explains the definitions used in the developed framework:

Definition 1. *Sampling* an arc $a_{ij} \in A$ refers to making an API call to retrieve the travel time t_{ij}, which is then recorded in a travel time matrix denoted as $\hat{T}$. The matrix $\hat{T}$ represents the sampled version of the complete travel time matrix T, where only a subset of the arcs in A have their travel times retrieved from API calls and recorded. For arcs that have not been sampled, their travel times in $\hat{T}$ are imputed based on the available sampled values. The methodology used for imputing unsampled travel times is explained in the following section. If all arcs were to be sampled, then $\hat{T}$ would be identical to the complete cost matrix T.

Definition 2. The *true* length of a spanning subgraph $H \subseteq G$ (e.g., a tree or feasible TSP solution) refers to the sum of sampled travel times of the arcs in H and is shown by $L(H)$. Let $\hat{L}(H)$ represent the length of H computed using the travel times from $\hat{T}$, which may include a mix of sampled and imputed values. If all arcs in H have been sampled previously, then $\hat{L}(H) = L(H)$.

The objective is to construct $\hat{T}$ with a minimal number of sampled arcs such that a classical TSP solver can use it as the cost matrix to find a feasible sequence s, where the sampled length of s, $L(s)$, closely approximates the sampled length of the optimal sequence, $L(s^*)$. The process consists of three components: (1) the arc selection strategy, which determines which arcs to sample, (2) the imputation strategy, which estimates the missing values in $\hat{T}$ using the travel times of the sampled arcs, and (3) the stopping strategy, which establishes when $L(s)$ is sufficiently close to $L(s^*)$ without directly knowing $L(s^*)$. Figure 1 illustrates the four-step framework developed in this study, combining the mentioned components.

In Step-0, the framework initializes the problem instance by constructing the Euclidean distance cost matrix, $D = [d_{ij}]$, where d_{ij} represents the Euclidean distance between v_i and v_j, and calculates the bearing angles. The bearing angles represent the direction from one location to another, expressed in degrees relative to the north (i.e, 0°). These initial calculations are used for imputing unsampled arcs in $\hat{T}$ and to rank the arcs based on their likelihood of being part of the optimal TSP solution. We use the α-nearness method to rank the instance's arcs, $a_{ij} \in A$, based on their likelihood of being present in the optimal TSP solution constructed from the Euclidean distance matrix (Helsgaun, 2000). As a benchmark, we compare the performance of the α-nearness method against a random arc selection strategy.

In Step-1, a lower bound and an upper bound for the TSP instance are computed using D. The subset of arcs present in these bounds are sampled, which serve as the basis for constructing the initial $\hat{T}$. For the upper bound, we use the Nearest Neighbour (NN) heuristic, and for the lower bound, we use a spanning subgraph called a minimum one-tree (Held & Karp, 1970; Helsgaun, 2000). Steps-2 and -3 iteratively sample arcs and update $\hat{T}$ until the stopping criterion is met. In each iteration, in addition to the arcs sampled in the bounds, we sample an additional arc based on the ranking provided by the arc selection strategy in Step-0. An imputation function is employed to estimate the missing values in $\hat{T}$ using the sampled arcs. We impute the value of $\hat{t}_{ij}$, the travel time over an unsampled arc, using a linear model. This model is trained on the travel times of sampled arcs, with the Euclidean distance of the arc and its bearing angle as the independent variables. The loss function is defined as $loss = \sum_i \sum_j |t_{ij} - \hat{t}_{ij}|$. This loss function enables the imputation function to be trained using linear programming, which can be calculated in near real-time. As a benchmark, we compare the performance of the linear regression imputation strategy against a K-Nearest Neighbour (k-NN) imputation strategy (Pedregosa et al., 2011).

The stopping criterion strategy monitors changes in the *sampled* length of upper bound and lower bound between iterations. If the bounds stabilize, with their relative changes falling below a predefined threshold (e.g., 5%), they most likely contain the arcs present or close to being present in s^* (i.e., $\hat{T}$ is likely to have sampled the arcs close to being present in s^*). Once the threshold is reached, the framework performs 10 additional iterations to ensure stability before stopping. The bounds are computed using $\hat{T}$ as the cost matrix, except during initialization (Steps 0 and 1), where the Euclidean distance matrix (D) is used. Once the framework determines that the stopping criterion is met, the constructed $\hat{T}$ is passed to `OR-Tools` (Perron & Furnon, 2023) to solve the TSP instance. The sampled length of the resulting sequence is then used to evaluate the quality of the solution. This study was implemented using a Macbook Air with an eight-core Apple M2 chip and 16 GB of RAM.

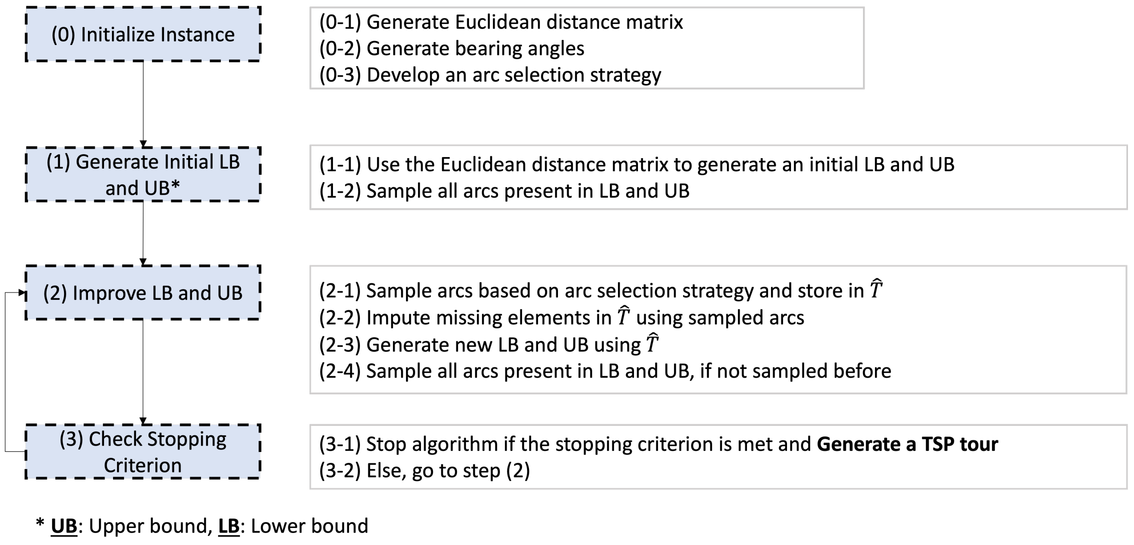

(0) Initialize Instance
(0-1) Generate Euclidean distance matrix
(0-2) Generate bearing angles
(0-3) Develop an arc selection strategy
(1) Generate Initial LB and UB*
(1-1) Use the Euclidean distance matrix to generate an initial LB and UB
(1-2) Sample all arcs present in LB and UB
(2) Improve LB and UB
(2-1) Sample arcs based on arc selection strategy and store in $\hat{T}$
(2-2) Impute missing elements in $\hat{T}$ using sampled arcs
(2-3) Generate new LB and UB using $\hat{T}$
(2-4) Sample all arcs present in LB and UB, if not sampled before
(3) Check Stopping Criterion
(3-1) Stop algorithm if the stopping criterion is met and **Generate a TSP tour**
(3-2) Else, go to step (2)
* **UB**: Upper bound, **LB**: Lower bound

Figure 1: The visual abstract of the developed framework.

3 Results and discussion

This study uses the open-source dataset from Amazon's Last-Mile Research Challenge, which contains 6,112 historically realized TSP instances in five cities in the United States. For descriptive statistics and more information about the data, refer to Merchán et al. (2022).

Figure 2 illustrates a sample problem instance. The line plots with markers illustrate the sampled values of the lower bound (blue), upper bound (red), and the TSP tour duration (black) computed by using $\hat{T}$ (i.e., $\hat{T}$ was used to generate the mentioned subgraphs and then their arcs were sampled to retrieve their true length). The dashed lines represent the true values of the lower bound, upper bound, and optimal TSP tour duration, calculated using the fully sampled travel time matrix T (these lines are not calculated as part of the framework and are for demonstration only). According to Figure 2, after 12 iterations, the quality gap between the solution generated by the framework and the optimal TSP solution computed using T is 5.3%. Notably, only 1,328 arcs are sampled for a problem size of 166 stops (i.e., 4.8% of arcs were sampled).

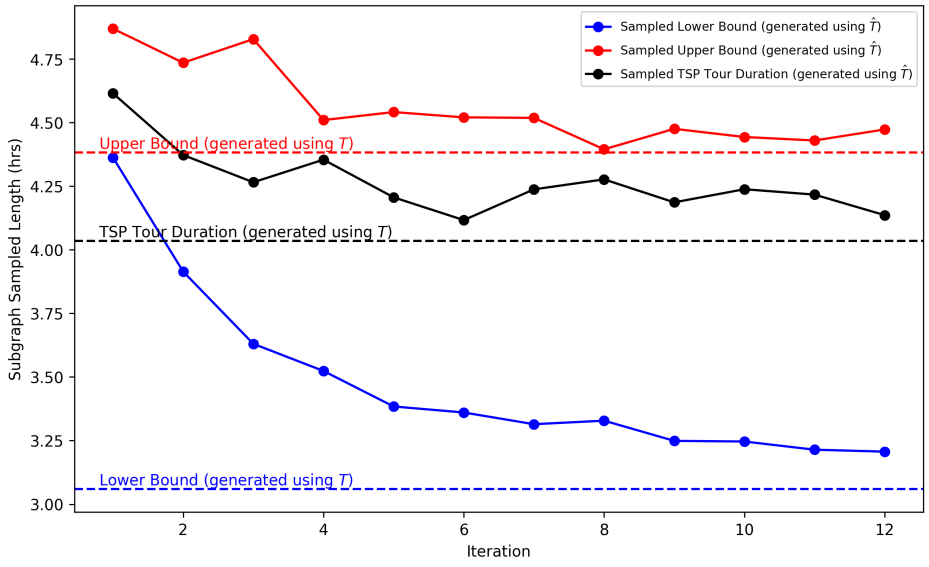
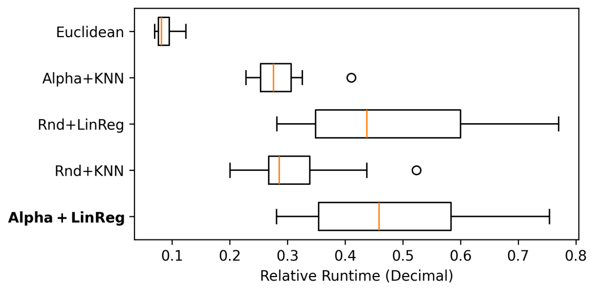


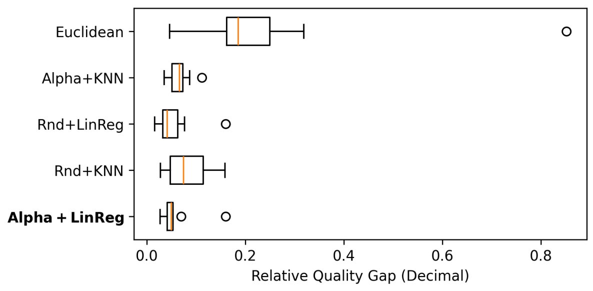
Figure 2: Framework applied to a sample problem instance.

The results of the algorithms are summarized in Figure 3, which compares their performance across three metrics: (a) runtime relative to the baseline, (b) quality gap compared to the baseline, and (c) the number of arcs sampled as a ratio of the total possible arcs. These results are based on a simulation of retrieving arcs from the Google Maps Travel Time API, taking into account the service limitations, such as rate limits and processing time, to reflect realistic constraints associated with API usage (Google, 2025). The baseline represents the case where the optimal TSP sequence is generated using the complete travel time matrix. The "Euclidean" algorithm uses the Euclidean distance matrix to generate the optimal TSP sequence, followed by sampling its arcs. The other algorithms explore variations of the proposed methodology, combining different arc sampling strategies (α-nearness and random selection) and imputation methods (linear regression and k-NN).

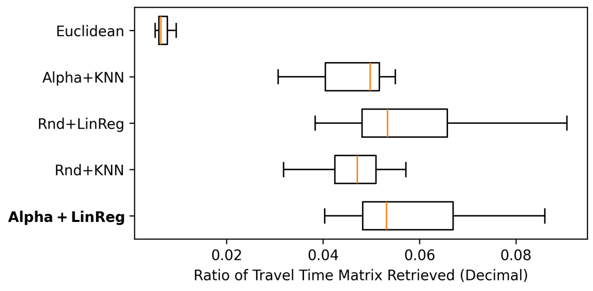
Figure 3a shows the runtime comparison of the algorithms relative to the baseline. The Euclidean algorithm is the fastest and requires sampling the least number of arcs (see Figure 3c), with a median runtime of 8.1% of the baseline. As expected, however, its performance in terms of quality is suboptimal. As shown in Figure 3b, the Euclidean approach has the largest quality gap among all algorithms, with a median value of 18.6%. Figure 3b demonstrates that all combinations of arc sampling strategies and imputation methods improve the quality gap compared to the Euclidean method. In the worst-case scenario, the *Rnd+KNN* method (random sampling with k-NN imputation) has a median quality gap of 7.5%, while the best results are achieved by the *Alpha+LinReg* method (α-nearness sampling strategy with linear regression imputation), with a median quality gap of 5.1%. These two cases respectively sampled a median of 4.7% and 5.3% of the entire travel time matrix. In terms of runtime, Figure 3a highlights that *Rnd+KNN* and *Alpha+LinReg* require a median of 28.6% and 45.9% of the runtime compared to the case where the complete travel time matrix is retrieved.



(a) Algorithm runtimes relative to the baseline.



(b) Algorithm quality gaps.



(c) Number of arcs sampled as a ratio of total possible arcs.

Figure 3: Comparison of algorithms based on (a) runtime, (b) quality gap, and (c) number of arcs sampled.

4 Conclusions

This study demonstrates that making strategic API calls can reduce costs and processing time while achieving near-optimal TSP solutions. We developed a data-driven framework that operates without the need for pretraining and can be applied in near real-time. Instead of retrieving the travel times for all pairs of stops, the framework selectively retrieves travel times only for the pairs of stops that are more likely to be visited consecutively in the optimal TSP solution. The baseline for comparison is the case where the full travel time matrix is retrieved, providing the optimal TSP solution. The proposed framework reduces the number of retrieved travel times to a median of 5.3% of the total matrix. This comes with a median quality gap of 5.1% and requires a median of 45.9% of the runtime compared to the baseline.

References

Google. (2025). Author. Retrieved from `https://developers.google.com/maps/documentation/distance-matrix/usage-and-billing?_gl=1%2A1lhccji%2A_up%2AMQ..%2A_ga%2ANTUyNjg0OTcyLjE3MzcyMzA1ODk.%2A_ga_NRWSTWS78N%2AMTczNzIzMDU4OS4xLjEuMTczNzIzMDU5NS4wLjAuMA..`

Held, M., & Karp, R. M. (1970). The traveling-salesman problem and minimum spanning trees. *Operations Research, 18*. doi: 10.1287/opre.18.6.1138

Helsgaun, K. (2000, 10). 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

Mapbox. (2025). Author. Retrieved from `https://docs.mapbox.com/api/navigation/matrix/`

Merchán, D., Arora, J., Pachon, J., Konduri, K., Winkenbach, M., Parks, S., … Merchán, M. (2022). Amazon last mile routing research challenge: Data set. *Transportation Science*. doi: 10.1287/trsc.2022.1173

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

Perron, L., & Furnon, V. (2023). *Or-tools*. Retrieved from `https://developers.google.com/optimization`