TrajRoute: Rethinking Routing with a Simple Trajectory-Based Approach – Forget the Maps and Traffic!

Maria Despoina Siampou, Chrysovalantis Anastasiou, John Krumm, Cyrus Shahabi
Dept. of Computer Science, University of Southern California, Los Angeles, United States
{siampou, canastas, jkrumm, shahabi}@usc.edu

2025 26th IEEE International Conference on Mobile Data Management (MDM) | 979-8-3315-2569-9/25/$31.00 ©2025 IEEE | DOI: 10.1109/MDM65600.2025.00040

***Abstract*—The abundance of vehicle trajectory data offers a new opportunity to compute driving routes between origins and destinations. Current graph-based routing pipelines, while effective, involve substantial costs in constructing, maintaining, and updating road network graphs to reflect real-time conditions. In this study, we propose a new trajectory-based routing paradigm that bypasses current workflows by directly utilizing raw trajectory data to compute efficient routes. Our method, named TrajRoute, uniquely "follows" historical trajectories from a source to a destination, constructing paths that reflect actual driver behavior and implicit preferences. To supplement areas with sparse trajectory data, the road network is also incorporated into TrajRoute's index, and tunable parameters are introduced to control the balance between road segments and trajectories, ensuring a unified and adaptable routing approach. We experimentally verify our approach by comparing it to an existing online routing service. Our results demonstrate that as the number of trajectories covering the road network increases, TrajRoute produces increasingly accurate travel time and route length estimates while gradually eliminating the need to downgrade to the road network. This highlights the potential of simpler, data-driven pipelines for routing, offering lower-maintenance alternatives to conventional systems.**

***Index Terms*—routing engines, trajectory-based routing, urban mobility**

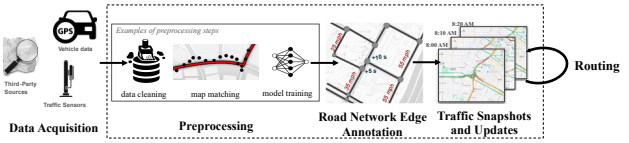
I. Introduction

The recent widespread adoption of location-acquisition technologies, such as GPS data from vehicles, has generated vast amounts of user trajectory data [10], [16]. For instance, companies such as Tesla and Google have access to comprehensive real-time trajectory datasets that their users continuously provide. This has offered a foundation for improving navigation services, where trajectory data are used to annotate the road network edges with more realistic travel time estimates [7]. As more data becomes available, these edges are continuously updated to reflect real-time conditions. A typical pipeline used by routing services, shown in Figure 1a, involves several cost-intensive preprocessing steps. One common step is map matching, where GPS data is aligned with the road network [19]. The network is then dynamically updated with this information, generating traffic snapshots that enable accurate path recommendations. In an effort to deliver personalized routing, a number of research studies have also focused on utilizing these trajectory data to extract more insights such as preferences, patterns, and popular paths which are also used to annotate the graph edges, allowing for more customized route recommendations [3]–[5], [8], [14], [17], [20], [25]. However, maintaining these graph-based pipelines is costly due to the need for continuous data collection, processing, and frequent updates to reflect real-time traffic estimates, rendering the entire process resource-intensive.

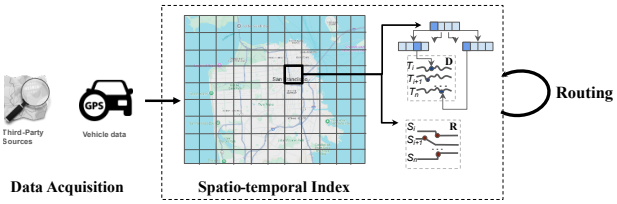
With the increasing availability of large-scale and up-to-date trajectory datasets and their current utilization of current routing services, we have identified a new opportunity: the ability to extract routes directly from historical raw data, eliminating the need for costly graph construction and ongoing maintenance, streamlining the process and leveraging the richness of real-world data. To this end, we propose a new approach for routing, namely TrajRoute, that directly leverages raw trajectory data for routing decisions. We demonstrate that accurate route recommendations can be derived by following historical trajectories. TrajRoute's approach is straightforward yet effective: given an origin, a destination, and a start time, the system utilizes historical trajectories to compute the optimal route in terms of travel time. If a complete historical trajectory between the origin and destination is unavailable, TrajRoute combines segments from multiple trajectories to construct a continuous path. Consider it analogous to hopping from one vehicle to another to reach the final destination.

TrajRoute is designed for scenarios where there is access to extensive historical trajectory data. One key advantage of TrajRoute is that it eliminates the need for the costly preprocessing steps of typical routing pipelines, that repeat continuously as the road network's edge weights to reflect real time conditions. As shown in Figure 1a these steps include tasks like map matching and converting trajectories into travel times for road segments. The steps within the dashed box are reprocessed regularly to account for current traffic conditions and as new data becomes available. In contrast, TrajRoute bypasses these updating steps by working directly with raw historical trajectory data. To retrieve relevant trajectories efficiently, TrajRoute uses a spatio-temporal grid-based index, allowing it to find trajectories that best match a user's query. The approach performs optimally when there is consistently a segment of a trajectory near the current position and within the present time frame. For scenarios where a direct path to the destination is not entirely available from historical data, TrajRoute integrates adjacent road segments to bridge the gaps, ensuring that users will always receive a valid route as

2375-0324/25/$31.00 ©2025 IEEE
DOI 10.1109/MDM65600.2025.00040



(a) Overview of a typical pipeline of current routing services. The steps in the dashed box are repeated as newer data becomes available.



(b) Our proposed TrajRoute pipeline.

Fig. 1: Comparison of pipelines: (a) current routing services and (b) our proposed pipeline.

a query answer. We note that if no trajectories pass through these roads, conventional routing methods won't have access to real-time travel times for those segments either. Figure 1b, presents the proposed framework.

We experimentally verify TrajRoute on a real trajectory dataset and compare the routes generated with those from Azure Maps[1] in terms of route accuracy and travel time precision. The results confirm that TrajRoute consistently delivers reliable and precise routing recommendations. One of the primary advantages of TrajRoute is its ability to inherently integrate routing features that are implicitly encoded in the trajectories, such as variable traffic speeds, intersection-related delays, and individual driver preferences, which are typically challenging and expensive to incorporate in standard graph-based routing systems. We note, that the focus of TrajRoute is not to completely discard the road network, but rather to eliminate the heavy-weight processes of map-matching, traffic inference, and graph edge updating. TrajRoute achieves this effectively when supplied with abundant up-to-date trajectory data. Additionally, it is simple in tuning and maintenance, requiring only three parameters: the grid size, which determines the granularity of trajectory indexing, and two additional parameters that adjust the reliance on historical trajectory data versus traditional road network segments. This setup ensures a flexible and effective routing solution.

We would like to clarify two potential misconceptions about our work. First, TrajRoute should not be misunderstood as discarding the road network entirely. Instead, our focus is on eliminating the repetitive updates to edge weights (e.g., travel times). TrajRoute utilizes trajectory data to estimate travel times whenever available and relies on static road segment attributes only in the absence of real-time data, a limitation shared by all methods. Second, TrajRoute is specifically designed for organizations with access to extensive trajectory data (e.g., Tesla, Google), rather than for academic or public/open-source contexts. Without trajectory data, real-time traffic information, and consequently time-dependent routing, becomes infeasible, underscoring the relevance of our approach for industry-focused applications.

II. Related Work

Current routing approaches. The vehicle routing problem has attracted significant attention over the years. Traditionally, this problem has been approached using graph-based models where the road network is represented by nodes (intersections or destinations) and edges (travel paths). Path-finding algorithms such as Bellman-Ford, Dijkstra, A*, and bidirectional A* are commonly employed to determine the shortest path between two nodes [15], [22]. To more effectively direct the search, algorithms like ALT [12], [13] have been developed. The ALT algorithm pre-selects a group of nodes called landmarks, and the shortest distances from every node in the network to these landmarks are calculated and stored in advance. During routing, a bidirectional A* search utilizes these pre-computed distances to estimate a lower bound distance between nodes for the heuristic computation. To accelerate computations, several pre-processing-based methods have also been proposed in the literature [1], [6], [9], [11], [24], [26], such as Contraction Hierarchies (CH) [11], which simplifies the graph by removing less important nodes, that once removed, would not affect the shortest path distances between the other, and more important, nodes. Our approach diverges from these traditional methods by not relying solely on graph-based representations of road networks.

Trajectory-based routing approaches. Conventional routing methods typically focus on finding the lowest cost paths based on distance or travel time. However, using historical trajectory data has opened up possibilities for more personalized or context-aware routing suggestions [3]–[5], [8], [14], [17], [18], [20], [23], [25]. For example, Chang et al. [4] utilized historical trajectory segments to build a road network, recommending routes based on the most frequently traveled road segments. Ceikute et al. [3] proposed a sophisticated scoring function of historical routes, considering both the frequency of traversals and the distinct number of drivers that took each route, but also temporal factors such as whether the travel occurred during peak traffic hours. Guo et al. [14] developed a routing preference model that first clusters the vertices of the original road network into regions, creating a region graph. Then, routing preferences are learned from the available historical trajectories that connect some region pairs. These preferences are transferred to similar region pairs that are not connected by trajectories, ensuring that the approach will work even when trajectories are not available. Our approach differs from these methods in two distinct ways. First, unlike previous studies, we do not extract insights from the trajectory data and do not model scoring functions to rank trajectories based on these insights. Instead, we leverage the raw data directly to guide routing decisions. By directly

[1]https://azure.microsoft.com/en-us/products/azure-maps

following previous historical trips, we are implicitly recommending routes that might be more preferred by the drivers. Second, the aforementioned approaches still depend on a graph-based framework, while our approach is grid-based. Our innovations are designed to greatly reduce the maintenance cost of previous routing approaches.

To the best of our knowledge there is no previous work that directly utilized trajectories to extract routes from an arbitrary origin to a destination. One approach close to ours was presented in [2], where raw trajectory data were utilized for constructing isochrone and reverse isochrone maps. While our focused problem is different (single source to a single destination), this previous approach, though, fails when trajectory data does not fully cover that area.

III. The TrajRoute Approach

A. Index Structure

One of the main advantages of TrajRoute is its ability to operate directly on trajectory data. This is enabled by a grid-based index that quickly retrieves trajectories near a query's start and destination locations. An empty grid G over the area of interest, where each cell has a uniform edge length of n meters. Each trajectory T is a sequence of GPS measurements, $T = \langle (p_1, ts_1), \ldots, (p_{|T|}, ts_{|T|}) \rangle$, where $p_i = (lat_i, lon_i)$ are coordinates and ts_i is the timestamp. Measurements are mapped to grid cells based on their coordinates. This structure enables efficient querying and tolerates minor GPS inaccuracies, eliminating the need for map matching.

Since traffic conditions vary throughout the day, the relevance of trajectories for routing depends on the time and day of travel. For instance, during rush hours, drivers may favor longer but faster routes to avoid congestion, whereas at night, highways are often preferred due to lighter traffic. To ensure realistic route suggestions and accurate travel time estimates, TrajRoute filters trajectories based on their timestamps to match the query's time and day. Specifically, a B-tree is built over the timestamps of trajectories within each non-empty grid cell. During query processing, only trajectories that fall within the specified temporal window w_{time} are considered. This enables TrajRoute to retrieve time-relevant trajectories without the need to create, maintain, or update explicit traffic snapshots of the road network.

B. Identifying Neighbors

In traditional graph-based routing methods, neighbors are typically defined as nodes directly connected by an edge to the current node being explored. In TrajRoute, the grid structure offers an intuitive way to define neighbors. To that end, we consider two types of neighbors: the next GPS measurement in the current trajectory, which captures the natural, sequential movement along a continuous path, and nearby GPS measurements from other trajectories that are mapped to the same cell.

Formally, given a trajectory dataset D, the index G, and a query tuple $Q = (p_{\text{OR}}, p_{\text{DEST}}, t)$, we define the neighbors Nbr of a node [2] p_k of trajectory $T \in D$, as follows:

$$\begin{aligned} Nbr(p_k) = &\{p_{k+1} \mid k < |T|\} \cup \\ &\{p_m \mid p_{m-1} \in \text{cell}(p_k, G), \\ &\quad T' \neq T, |ts_{p_m} - ts_{p_k}| \leq w_{time}\} \end{aligned} \quad (1)$$

Here, p_{k+1} represents the next GPS measurement on T, if it exists. This captures the natural progression along a continuous path, and no temporal constraints are imposed on this transition. Furthermore, p_m represents the m_{th} measurement of another trajectory $T' \in D$ whose GPS measurement p_{m-1} exists in the same cell as p_k and was recorded at a close time and same date as p_k.

This formulation allows TrajRoute to continue exploring a trajectory by following the next point while also considering alternative routes by examining other trajectories whose GPS measurements are spatially nearby.

C. Movement Cost

Each neighbor is associated with a *movement cost*, which influences its priority for processing and contributes to the overall cost of the final route. Here, these costs are naturally encoded in the dataset.

Let us assume that we are currently exploring node $p_k \in T$, and let us define p_m as one valid neighboring node. As mentioned before, we consider two types of neighbor transitions. The first type is moving along the same trajectory. In this case, we can move to the next GPS point of the same trajectory, $p_m = p_{k+1}$ and the movement cost is based on their timestamp difference, that is $|ts_{p_{k+1}} - ts_{p_k}|$.

The second type of neighbor transition is switching to a different trajectory T'. For this transition to be valid, the previous point in T', p_{m-1}, must lie in the same grid cell and within a temporal window w_{time} as p_k. For this, a constant transition cost from T to T', called τ_c, is applied, which depends on the size of the grid cell. Intuitively, larger cells require higher transition costs, while for smaller cells, τ_c can be as low as zero. After switching to T', the movement cost is computed as the time difference between p_m and p_{m-1}, that is, $\tau_c + |ts_{p_m} - ts_{p_{m-1}}|$.

Overall, the movement cost is formally defined as:

$$C_{\text{traj}}(p_k, p_m) = \begin{cases} |ts_{p_{k+1}} - ts_{p_k}| \text{ if } p_k, p_m \in T, m = k+1 \\ \tau_c + |ts_{p_m} - ts_{p_{m-1}}| \text{ if } p_k \in T, p_m \in T' \end{cases} \quad (2)$$

D. Fallback to Road Network

1) Road Network Integration: TrajRoute finds the optimal path from an origin to a destination by leveraging trajectories stored in D, assuming they provide sufficient spatial and temporal coverage. However, in cases where trajectory coverage is insufficient, the system must seamlessly switch between following available trajectories and using the underlying road network to bridge gaps. This transition ensures that a complete path is always returned without introducing additional overhead to the pipeline.

[2] In this context nodes are represented by points in our dataset.

To enable this, we index road segments S using the same grid structure G. Each segment S represents a portion of the road network between two consecutive junctions or endpoints and is modeled as a polyline $\langle p_1, p_2, \ldots, p_{|S|} \rangle$, where each $p_i = (lat_i, lon_i)$ denotes an intermediate point. Each segment also has an associated speed limit v_S. Unlike trajectories, road segments lack temporal information; thus, the B-tree index on timestamps applies only to trajectories, not road segments.

2) Adjusting Neighbors and Movement Costs: With a unified indexing approach, extending the notion of neighbors to consider both trajectory and road points becomes straightforward. For road points, the next point on the polyline can be considered a natural neighbor, similar to traveling along road segments. Additionally, like trajectory neighbors, road points within the same grid cell can also be considered neighbors.

The movement cost between road nodes is associated with the distance traveled and can be defined in two cases. First, for movement along the same road segment S, the cost is determined by the distance between consecutive points on the polyline, divided by the speed v_S of S. Second, for transitions to a different road segment S', the movement cost is a combination of a transition cost τ_c and the distance-based movement cost on the new segment.

Formally, the cost of movement C_{road} between two nodes that belong to road segments can be expressed as:

$$C_{\text{road}}(p_k, p_m) = \begin{cases} \frac{\text{dist}(p_k, p_{k+1})}{v_S}, \text{if } p_k, p_m \in S, m = k+1 \\ \tau_c + \frac{\text{dist}(p_{m-1}, p_m)}{v_{S'}}, \text{if } p_k \in S, p_m \in S' \end{cases} \tag{3}$$

where *dist* is the Haversine distance between the nodes.

The overall cost across neighbors can be consistently defined across both datasets: a point can progress along its path following the next GPS measurement in a trajectory or the next point in a road segment while also considering nearby points from other trajectories or road segments within the same cell.

The overall cost function C_{base} that accounts for all the different types of transitions is defined as:

$$C_{base}(p_k, p_m) = \begin{cases} C_{\text{traj}}(p_k, p_m), & \text{if } p_m \in T \\ C_{\text{road}}(p_k, p_m), & \text{if } p_m \in S \end{cases} \tag{4}$$

3) Denoting Preference: While our cost function computes travel time costs based on the neighbor type, it inherently favors road segments over trajectories. This is because the trajectories reflect real-time traffic and road conditions, and thus, they naturally exhibit higher costs. On the other hand, the road segments are associated with theoretical costs, which assume a constant average speed, underestimating the actual travel time, especially during peak traffic periods.

To account for this, we introduce a new parameter, $r_{\text{penalty}} \geq 0$, to penalize the cost of moving to road neighbors over trajectory neighbors. To that extent, we update the cost of movement C_{adj} from a node p_k to p_m as follows:

$$C_{adj}(p_k, p_m) = \begin{cases} (1 + r_{\text{penalty}}) C_{base}(p_k, p_m), \ p_m \in S \\ C_{base}(p_k, p_m), \text{ otherwise} \end{cases} \tag{5}$$

While prioritizing the exploration of trajectory routes over road segments is effective, it is also more realistic and practical to prioritize continuous trips over trajectory segment transitions. This mirrors actual driving behaviors and helps preserve the temporal continuity of real-world trips. To achieve this, we introduce a second parameter, rw, which represents the *continuity reward* for maintaining a continuous trip on the same historical trajectory. Now, final cost of movement C between nodes p_k and p_m can be defined as:

$$C(p_k, p_m) = \begin{cases} e^{-rw} C_{adj}(p_k, p_m), \ p_k, p_m \in T, m = k+1 \\ C_{adj}(p_k, p_m), \text{ otherwise} \end{cases} \tag{6}$$

The final cost function improves TrajRoute's ability to generate routes that are not only cost-effective but also more closely aligned with typical driving patterns.

E. Query Answering

Given the neighbor identification process and the cost functions, any pathfinding algorithm could be employed to extract the routing path from an origin to a destination. In our implementation we chose to utilize the A* search algorithm with heuristic function $h(p_i) = \frac{dist(p_i, Q.p_{DEST})}{v_{max}}$, where $dist$ is the haversine distance and v_{max} the maximum allowable speed in the area of interest, i.e., the maximum speed limit in San Francisco is 70 mph. This heuristic always underestimates the actual travel cost of reaching $Q.p_{DEST}$, which makes it admissible. More tight heuristics could also be applied [9].

IV. Experimental Evaluation

A. Experimental Setup

1) Dataset and Parameter Selection: We utilize a real-world taxi trajectory dataset collected in San Francisco, USA[3] [21]. The dataset contains over 11M records, corresponding to more than 1M trajectories. We remove trajectories that traveled less than 500 meters or less than 2 minutes during pre-processing. We used OpenStreetMap[4] to extract the San Francisco area's road network and road segment travel speed limits. The dataset has a total of 27,279 roads. The trajectory dataset has a spatial coverage of 99.0% of the extracted roads. Regarding the parameter selection, we set the grid size to $n = 100$ meters. Given that this is a relatively small size, we set the transition cost to 0. Lastly, we look at trajectories occurred 30 mins before or after the query time and set the threshold for identifying the destination point d_{thres} to 100m.

[3]https://stamen.com/work/cabspotting/

[4]www.openstreetmap.org

2) Evaluation and Metrics: For evaluation, we generate a set of origin-destination queries $\mathcal{Q}$ by randomly sampling start and end points from the trajectories in our dataset. The timestamp of the start point is used as the departure time. To evaluate the quality of the routes provided by TrajRoute, we compare the route length and travel time cost to those obtained from Azure Maps. Given that there are multiple route alternatives for each origin-destination pair, we believe that comparing travel time and route length provides more meaningful insights than directly comparing the similarity of the routes. For this comparison, we query the Azure Maps API [5] and set the departure time to match the one in our queries so that the results from Azure reflect accurate traffic conditions. For each comparison, we calculate the mean absolute error (MAE) for both the route length and travel time. The average route length for our query routes is 5.4 km.

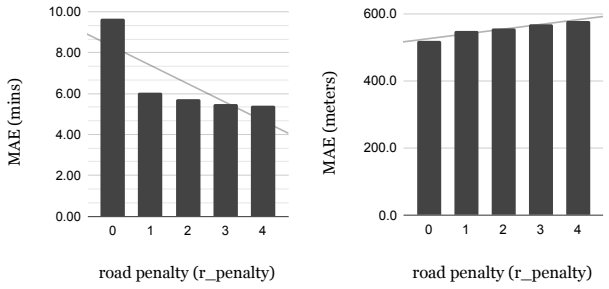
B. Quality of the generated paths

In this section, we show that the quality of the routes generated by TrajRoute is comparable to or better than those produced by Azure Maps, which incorporates real-time and historical data and relies on a significant pre-processing and updating pipeline. We show this by analyzing the impact of TrajRoute's tunable parameters, the road penalty $r_{penalty}$, and the continuity reward (rw) on route quality. By varying these parameters, we generate different routes and compare them against those provided by Azure Maps to assess their quality.

We begin by focusing on the effect of the road penalty, $r_{penalty}$, by testing values in the range $[0, 4]$. This range reflects the trade-off between real-world travel conditions, as captured by trajectories, and the idealized conditions reflected in the road network. Figure 2 shows the results for different values of $r_{penalty}$. As the road penalty increases, discouraging the use of road segments and encouraging greater reliance on historical trajectories, we observe a decreasing trend in the mean absolute error for travel time estimation (i.e., the estimated time for drivers to reach their destinations) in Figure 2a. This indicates that relying more on historical trajectories allows TrajRoute to better account for real-world traffic patterns, leading to more accurate travel time predictions. These predictions align more closely with those from Azure Maps, which incorporates both real-time and historical traffic data into its calculations. We also notice that improvements diminish when $r_{penalty} > 3$. This is likely because 75% of the trajectories were collected during off-peak times, meaning that for this portion of the dataset, TrajRoute can effectively prioritize trajectories without requiring to excessively penalize the use of road segments.

Additionally, we report the distance differences between the routes generated by TrajRoute and Azure Maps to demonstrate that the routes generated by our approach are reasonable (i.e., similar in length). In this case, we observe a slight increase in path distance as $r_{penalty}$ increases, as shown in Figure 2b. This can be explained by the fact that as TrajRoute relies more heavily on trajectories, it may select routes that are slightly longer but preferred by drivers. These small deviations (less than 600 meters) might increase the total trip distance but reflect more realistic driver choices.

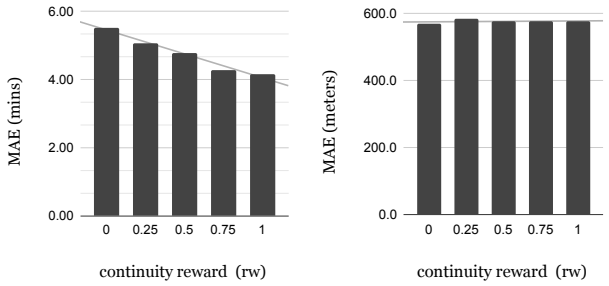
[5]https://www.microsoft.com/en-us/maps/azure/azure-maps



(a) MAE of route travel time (b) MAE of route distance

Fig. 2: Results for different r_{penalty} values $\in$ [0, 4].

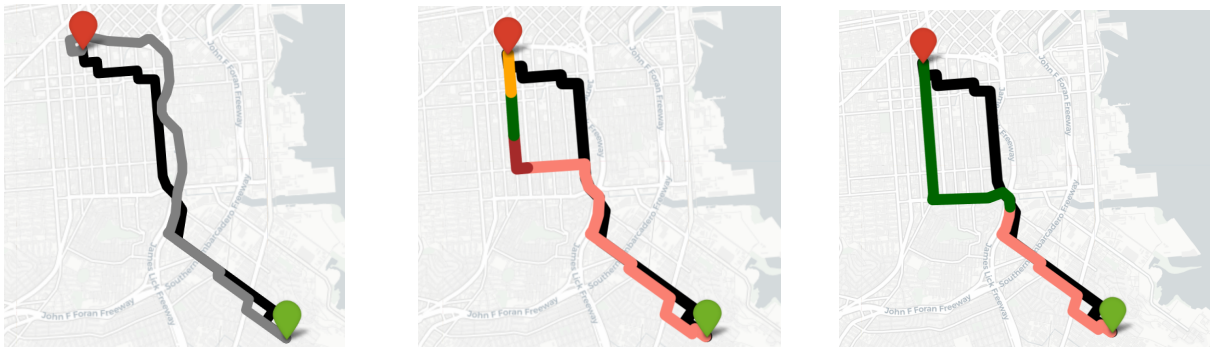
We also look at the impact of $r_{penalty}$ on query execution time. As shown in Figure 5a, as the penalty increases, there is a gradual rise in average execution time. This increase is expected, as higher penalties force TrajRoute to rely more heavily on trajectory data, resulting in a larger search space and more paths to evaluate. However, the increase in execution time is relatively small, suggesting that TrajRoute can manage the shift from road segments to trajectories without introducing significant delays ($<$ 1 second with our modest hardware).



(a) MAE of route travel time (b) MAE of route distance

Fig. 3: Results for different rw values $\in$ [0, 1].

Next, we examine the effect of the continuity reward (rw), which encourages following the same trajectory over switching between different trajectories or road segments. For this experiment, we set $r_{penalty} = 3$. As shown in Figure 3a, we again observe a decrease in the MAE for time travel cost as rw increases. This suggests that following the same trajectory provides more accurate time estimates. This observation is intuitive, as sticking to a single trajectory closely mimics real-world driver behavior and allows TrajRoute to capture the real-time conditions of a specific trip. By doing so, TrajRoute is able to implicitly encode how a driver navigates and adjusts to conditions along the route, leading to more accurate travel time calculations that align even closer with those of Azure Maps, which computes routes that are sensitive to estimated traffic speeds. At the same time, we observe in Figure 3b that the MAE for path distance remains relatively stable across all



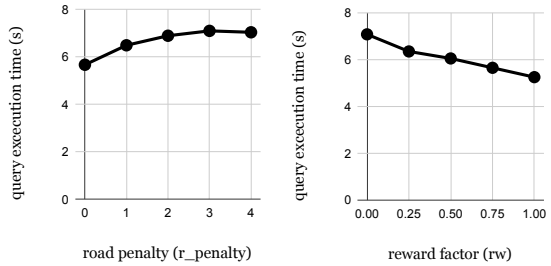
(a) Route for $r_{penalty} = 0$ and $rw = 0$. TrajRoute ETA: 10 mins.

(b) Route for $r_{penalty} = 3$ and $rw = 0$. TrajRoute ETA: 16.21 mins.

(c) Route for $r_{penalty} = 3$ and $rw = 0.75$. TrajRoute ETA: 19.53 mins.

Fig. 4: Example of routes computed by TrajRoute for different $r_{penalty}$ and rw values. The departure time for this query is 06:01 PM, and Azure Maps ETA is 21 mins. Grey color represents road segments, black color represents the route generated by Azure Maps. Other colors represent different trajectories.

values of rw. This stability suggests that while the reward encourages staying on the same trajectory, it does not cause significant deviations in the overall route distance. The small values of rw ensure that the approach avoids blindly following trajectories without considering optimal paths, maintaining reasonable route distances. Regarding performance for the rw, Figure 5b shows a steady decrease in average query execution time as the reward increases. Since continuity rewards encourage the system to stick to the same trajectory rather than switching between multiple trajectories or road segments, the search space is reduced, and fewer paths need to be explored, leading to faster query execution times.



(a) Impact of $r_{penalty}$ on avg. query execution time.

(b) Impact of rw on avg. query execution time.

Fig. 5: Average query execution time.

C. Visual Evaluation

In this section, we present real-world examples of the routes computed by TrajRoute for different values of road penalty, $r_{penalty}$, and continuity reward, rw. When both $r_{penalty}$ and rw are set to 0, as shown in Figures 4a, TrajRoute primarily generates paths using road segments. As explained earlier, this is expected because road segments are associated with theoretical costs, assuming a constant average speed, which often underestimates actual travel time. We further observe that the generated routes tend to favor highways, where higher speed limits contribute to faster-estimated travel times. As a result, the ETA estimates from TrajRoute differ more from those of Azure Maps. We reiterate that our comparison assumes Azure Maps has access to up-to-date traffic data, which makes this comparison somewhat unfair. If Azure Maps lacked trajectory data for those road segments, it would also default to predefined speed limits, potentially generating the same ETA and path as TrajRoute.

When $r_{penalty}$ is increased to 3, as shown in Figures 4b, TrajRoute relies more heavily on trajectories. This shift improves the accuracy of the ETA estimates, as trajectories better capture real-world traffic patterns. As demonstrated in Figure 4c, by setting rw to 0.75, TrajRoute encourages continuity by favoring single trajectories, reducing the frequency of switching between different trajectories. This further improves ETA estimates, bringing them even closer to those of Azure Maps, as the system follows real-world driving behavior.

V. Conclusion

In this paper, we introduced TrajRoute, a novel trajectory-based routing approach that eliminates the costly preprocessing steps employed by current navigation pipelines by directly leveraging raw trajectory data to compute efficient routes. Our experiments show that, with sufficient and up-to-date trajectory data, TrajRoute produces realistic routes with travel times and distances closely matching those from Azure Maps. This demonstrates TrajRoute's ability to implicitly capture real-world factors such as traffic patterns, intersections, traffic lights, and stop costs, all while requiring low maintenance costs. Without up-to-date and sufficient trajectory data, both our approach, TrajRoute, and standard navigation systems like Azure Maps and Google Maps, which rely on trajectory data to infer traffic and travel times for road segments, would experience degraded performance. In the future, we plan to deploy TrajRoute within an end-to-end navigation pipeline. We acknowledge that for the final step of a real pipeline, we need to map-match TrajRoute's recommended path. However, we would like to emphasize that this map-matching of the final results is significantly less computationally intensive compared to map-matching all incoming collected trajectories at the start of the pipeline and as new trajectory data becomes available.

REFERENCES

[1] Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *Experimental Algorithms: 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proceedings 10*, pages 230–241. Springer, 2011.

[2] Chrysovalantis Anastasiou, Chao Huang, Seon Ho Kim, and Cyrus Shahabi. Time-dependent reachability analysis: A data-driven approach. In *2019 20th IEEE International Conference on Mobile Data Management (MDM)*, pages 138–143. IEEE, 2019.

[3] Vaida Ceikute and Christian S Jensen. Vehicle routing with user-generated trajectory data. In *2015 16th IEEE International Conference on Mobile Data Management*, volume 1, pages 14–23. IEEE, 2015.

[4] Kai-Ping Chang, Ling-Yin Wei, Mi-Yeh Yeh, and Wen-Chih Peng. Discovering personalized routes from trajectories. In *Proceedings of the 3rd ACM SIGSPATIAL international workshop on location-based social networks*, pages 33–40, 2011.

[5] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. Discovering popular routes from trajectories. In *2011 IEEE 27th International Conference on Data Engineering*, pages 900–911. IEEE, 2011.

[6] Theodoros Chondrogiannis and Johann Gamper. Pardisp: A partition-based framework for distance and shortest path queries on road networks. In *2016 17th IEEE International Conference on Mobile Data Management (MDM)*, volume 1, pages 242–251. IEEE, 2016.

[7] Jian Dai, Bin Yang, Chenjuan Guo, Christian Søndergaard Jensen, and Jilin Hu. Path cost distribution estimation using trajectory data. *Proceedings of the VLDB Endowment*, 10(3):85–96, 2016.

[8] Daniel Delling, Andrew V Goldberg, Moises Goldszmidt, John Krumm, Kunal Talwar, and Renato F Werneck. Navigation made personal: Inferring driving preferences from gps traces. In *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems*, pages 1–9, 2015.

[9] Ugur Demiryurek, Farnoush Banaei-Kashani, Cyrus Shahabi, and Anand Ranganathan. Online computation of fastest path in time-dependent spatial networks. In *International Symposium on Spatial and Temporal Databases*, pages 92–111. Springer, 2011.

[10] Zhiming Ding, Bin Yang, Yuanying Chi, and Limin Guo. Enabling smart transportation systems: A parallel spatio-temporal database approach. *IEEE Transactions on Computers*, 65(5):1377–1391, 2015.

[11] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7*, pages 319–333. Springer, 2008.

[12] Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *SODA*, volume 5, pages 156–165, 2005.

[13] Andrew V Goldberg, Haim Kaplan, and Renato F Werneck. Reach for a*: Efficient point-to-point shortest path algorithms. In *2006 Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 129–143. SIAM, 2006.

[14] Chenjuan Guo, Bin Yang, Jilin Hu, and Christian Jensen. Learning to route with sparse trajectory sets. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1073–1084. IEEE, 2018.

[15] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[16] Jilin Hu, Bin Yang, Chenjuan Guo, and Christian S Jensen. Risk-aware path selection with time-varying, uncertain travel costs: a time series approach. *The VLDB Journal*, 27:179–200, 2018.

[17] Julia Letchner, John Krumm, and Eric Horvitz. Trip router with individualized preferences (trip): Incorporating personalization into route planning. In *AAAI*, pages 1795–1800, 2006.

[18] Xiucheng Li, Gao Cong, and Yun Cheng. Spatial transition learning on road networks with deep probabilistic models. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 349–360. IEEE, 2020.

[19] Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 336–343, 2009.

[20] Kayur Patel, Mike Y Chen, Ian Smith, and James A Landay. Personalizing routes. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 187–190, 2006.

[21] Michal Piorkowski, Natasa Sarafijanovoc-Djukic, and Matthias Grossglauser. A Parsimonious Model of Mobile Partitioned Networks with Clustering. In *The First International Conference on COMmunication Systems and NETworkS (COMSNETS)*, January 2009.

[22] Ira Pohl. Bi-directional search, machine intelligence 6. *Edinburgh University Press, Edinburgh*, 127:14, 1971.

[23] Dimitris Sacharidis, Panagiotis Bouros, and Theodoros Chondrogiannis. Finding the most preferred path. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–10, 2017.

[24] Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys (CSUR)*, 46(4):1–31, 2014.

[25] Bin Yang, Chenjuan Guo, Yu Ma, and Christian S Jensen. Toward personalized, context-aware routing. *The VLDB Journal*, 24:297–318, 2015.

[26] Andy Diwen Zhu, Hui Ma, Xiaokui Xiao, Siqiang Luo, Youze Tang, and Shuigeng Zhou. Shortest path and distance queries on road networks: towards bridging theory and practice. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 857–868, 2013.