Follow The Best: Crowdsourced Automated Travel Advice

Abdullah AlDwyish
The University of Melbourne
Melbourne, Australia
aldwyish@student.unimelb.edu.au

Egemen Tanin
The University of Melbourne
Melbourne, Australia
etanin@unimelb.edu.au

Shanika Karunasekera
The University of Melbourne
Melbourne, Australia
karus@unimelb.edu.au

ABSTRACT

Recently many navigation services have started to collect traffic data from drivers to enhance their services. The focus of their efforts are on real-time estimation of traffic conditions to offer better navigation services and help people avoid traffic incidents. However, using drivers as mere traffic sensors and aggregating their data disposes off any human knowledge and judgment that exists in their choices. Commuters in a city have a valuable knowledge about their city traffic patterns as a result of their daily commute over many years. In our work we take a different approach to navigation advice, we use individual driver trajectories to deduce travel choices. Our goal is to help drivers avoid traffic events by providing routes that bypassed congested routes during similar events in the past. We present a solution that combines graph search and trajectory search to find the fastest path taken in the past using historical trajectories under similar conditions in a city. We evaluate our approach through extensive experiments in realistic settings and experimental results show that our solution was able to find faster paths, avoiding the congested areas in cases where state-of-the-art methods resulted in drivers leading to routes through congested areas.

CCS CONCEPTS

•**Information systems** →*Location based services;*

KEYWORDS

Spatial Databases, Location-based Social Networks

ACM Reference format:
Abdullah AlDwyish, Egemen Tanin, and Shanika Karunasekera. 2017. Follow The Best: Crowdsourced Automated Travel Advice. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Melbourne, VIC, Australia, November 7–10, 2017 (MobiQuitous 2017),* 10 pages.
DOI: 10.1145/3144457.3144475

1 INTRODUCTION

Advances in mobile computing and sensor technologies have enabled drivers to record and share details of their trips in real-time. This has led to a new generation of smart data-driven navigation services. These navigation services collect traffic data from drivers in exchange for alerting them about traffic problems. For example, Waze.com (a Google company) is a navigation service with an estimated user base of more than 50 million users worldwide. In addition, many classical navigation services have adopted this crowdsourcing approach as the base for their navigation model. For example, TomTom (a leading navigation company) is now collecting data from more than 400 million drivers worldwide [12]. Crowdsourcing via mobile apps and GPS devices implemented by such companies have changed the way traffic data is collected and analysed and has become essential for enabling smart and ubiquitous cities. It was estimated [7] that such smart navigation services could save, globally, about $400 billion each year by increasing the utilisation and efficiency of transportation networks.

Current navigation services collect traffic information for real-time traffic estimation. With better traffic estimates, navigation systems can help users avoid traffic and give better route directions. However, despite such crowdsourcing efforts, there is still evidence that routes recommended by these navigation services are not good enough for many local drivers. A recent study [2] compares popular routes that are preferred by local drivers with routes recommended by navigation services (Google Directions[1]). The study shows that drivers commonly do not follow the routes provided by routing services, and that preferred routes are not necessarily the seemingly shortest route at the time of service.

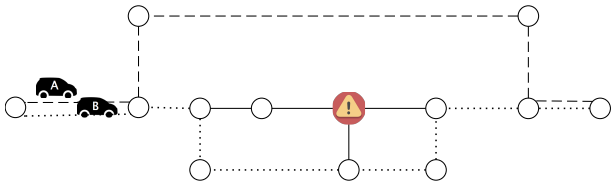
Local drivers use the road network on daily basis and have in-depth knowledge about the road network and its traffic patterns. Using drivers as mere traffic sensors and aggregating their data disposes off any human knowledge that exists in their actions. Typically, drivers share their data in the form of spatio-temporal trajectories, a sequence of time-stamped locations captured by GPS-enabled mobile devices. The techniques used by current navigation services deconstruct user trajectories to aggregate traffic information from different drivers per road segment. These techniques are good at estimating traffic levels through aggregation, but loses the knowledge available in complete individual trajectories. Therefore, actions and decisions made by individual drivers do not influence the navigation advice. For example, in Waze drivers collaborate to build a real-time estimate of the road network conditions by reporting their speed and location information. However, drivers' judgement and knowledge represented in their actions and turn decisions do not influence the advice Waze gives to other users.

In our work, we present an automated traffic management service to help drivers avoid traffic events using historical data from other drivers. Users subscribe to the service and when a traffic event occur, the service provide advice based on drivers actions during a similar traffic event in the past. We use a different approach for data-driven navigation that focuses on using local drivers' intelligence. We preserve the knowledge and behaviour of

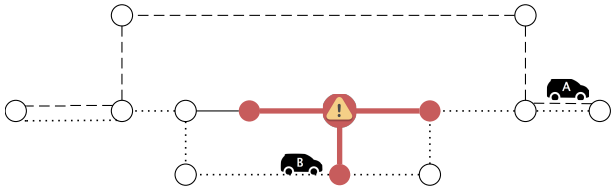
[1]Google has been crowdsourcing since 2008

MobiQuitous 2017, Melbourne, VIC, Australia

DOI: 10.1145/3144457.3144475



(a) Two drivers Alice (A) and Bob (B) planning to avoid an incident (caution mark). Bob will follow the dotted path as suggested by a state-of-the-art navigation system. Alice is familiar with the area and aware that incidents at this location spread to adjacent roads quickly so she decides to take the dashed route.



(b) Traffic propagates to adjacent routes and Bob is caught in traffic.

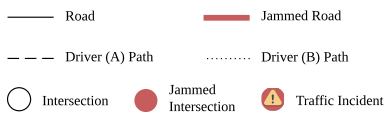


Figure 1: Local drivers can find better routes to avoid road incidents

drivers in the data and use historical trajectories without aggregation when giving driving advice. Our goal is to help users avoid traffic events by providing routes that were used by local drivers to avoid similar events in the past. We believe that drivers' knowledge about their local area, and drivers reaction to traffic events can improve navigation services. Expert drivers can foresee the evolution of traffic events and act according to their knowledge in the area before things evolve.

To illustrate, let us examine the scenario in Figure 1. Figure 1(a) shows two drivers Alice (A) and Bob (B) who are trying to avoid an incident (indicated by the caution mark). Bob is using a state-of-the-art navigation system, and the system advised him to take the dotted path to avoid the incident. However, by the time Bob reaches the intersection adjacent to the incident, he will end up getting stuck in traffic caused by the follow on effects of the incident on the neighborhood as shown in Figure 1(b). On the other hand, Alice being a local driver for many years knows that incidents on this road usually propagate to adjacent roads and decided to take an earlier detour to follow the dashed path and avoid the incident successfully as shown in Figure 1(b). This example, highlights local drivers' potential to foresee the evolution of events due to their knowledge in the area. If captured carefully, we believe this information can improve navigation services. It is now known there are benefits in predicting traffic events' impact to enhance navigation systems [9]. We propose a technique that uses historical trajectories to achieve this benefit.

In this paper, we consider providing efficient paths to drivers using past trajectories. The problem of finding efficient paths from a weighted graph has been studied extensively in the literature. However, the problem of finding a path from a graph using a set of historical trajectories has received little attention. We present a solution that combines graph search with trajectory search to find the fastest path that was taken to avoid a similar event in the past. The intuition behind our solution is simple, we can avoid an event by following the traces of the fastest driver from a similar situation in the past. Our work can be used independently or as part of a larger system to advise routes that can avoid traffic problems.

2 RELATED WORK

Most of the related work focus on popular route mining and finding drivers preferred routes. Only few studies suggest solutions to find efficient routes from past trajectories. Yuan et al. [14] proposed T-Drive, a system for route planning from taxi driver trajectories. Similar to our work, T-Drive uses past trajectories to estimate traffic conditions, however, T-Drive focus on estimating general traffic conditions (over a long period of time) while our method is event specific (focus on a short period of time). As a result, T-Drive does not distinguish between different traffic events and does not handle each traffic event independently. In addition, the system depends heavily on aggregation. The system assumes that all weekdays (from different weeks and months) share the same traffic conditions, and therefore project all weekdays trajectories into a single graph. Similarly, the system builds a graph for weekends and different weather conditions. The core data structure in T-Drive is a time-dependent graph, called a landmark graph. The landmark graph is used to represent and summarise the experience of taxi drivers. Each edge in the graph has a single 24 hour time travel profile estimated from the whole dataset. Also, each edge in the graph represents multiple roads (or road segments) in the actual road network. Therefore, T-Drive aggregates data from different trajectories that belong to different owners and different road segments into a single edge in the landmark graph.

Furthermore, a big part of the work done on route recommendation via historical trajectories is focused on discovering or learning the preferred routes of drivers. Popular routes mining algorithms aim to discover popular or most frequent routes taken by drivers. These algorithms provide routes that may be desirable for inexperienced users. However, they are not necessarily efficient for bypassing traffic events, and they may increase the travel cost which is not desirable for locals (e.g. take longer routes). Chen et al. [5] provide a framework that can find the most popular route from one location to another. Their work improve on sequential pattern mining where the popularity of a route is only measured by the number of trajectories. They suggest more accurate popularity measure that takes into account route destination. Their work can find the most popular route towards a destination. Luo et al. [6] argues the most popular routes change over time. So they developed popular route mining algorithm that can suggest different popular routes for different time periods. Chang et al. [4] argues that people prefer to take their own familiar routes and develop a framework to discover personalised routes from a driver own trajectory. Wei et al. [13] suggest algorithms to find popular routes from low-frequency trajectories. Ceikute and Jensen [3] proposed a system to enhance an existing routing service by using historical trajectories to find routes that are commonly used by local drivers.

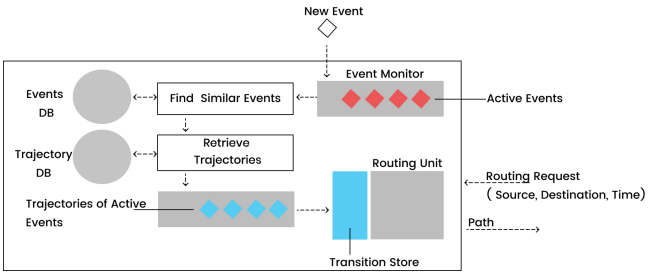


Figure 2: System Overview

The intuition is that routes that are used frequently by more drivers are preferred routes. Therefore, at the core of the system is a preference function that is used to score and rank routes based on: the number of trips associated with a route, temporal aspect of the trips and number of unique drivers. If the system can not find a route using trajectories, it will return the routes suggested by the external routing service.

Building on top of popular route mining methods, Su et al. [11] proposed CrowdPlanner: a crowd-based route recommendation platform. The main idea in CrowdPlanner is to get human feedback on routes generated by popular route mining algorithms. CrowdPlanner is not suitable for avoiding traffic congestion or to aid drivers in their daily commute due to following challenges. First, it requires explicit efforts from other human workers who may be in traffic themselves. Second, the contribution of human workers is limited by the candidate set offered by an automated system. CrowdPlanner tries to capture the knowledge of drivers by explicitly asking them about their preferred routes. The system uses existing popular or preferred route algorithms and web navigation services to generate a candidate set of alternative routes. Then, it attempts to evaluate the candidate set to select the best quality route. If the system is unable to select a route to recommend, it will generate a crowdsourcing task and explicitly ask human workers to evaluate the candidate set. Then the system will return the best route based on the feedback of human workers.

Furthermore, Bouros et al. [1] proposed efficient methods to speed up path queries on historical routes collection. The goal of the proposed algorithms is to speedup path queries in comparison with conventional graph search (BFS) by exploiting transitive information within routes. However, the methods in this work operate on routes made of point of interest (POI) and not detailed turn-by-turn directions like the routes we address in our work. Also, the indexing techniques do not handle the temporal dimension of the data and the search does not include any weight or cost of traversing the route. Therefore, these methods are suitable for exploring and finding new and interesting paths from route collections but are not be suitable for avoiding traffic events. The authors provided two search paradigms Route Traversal Search (RTS) and Link Traversal Search (LTS). RTS expands the search by considering all successor nodes, while LTS only considers the next link node (nodes that are shared by more than one route). Both paradigms terminate when they reach a route that contains or leads to the target, and therefore are faster than graph search.

Finally, Pan et al. [9] proposed ClearPath: a system that helps users avoid the impact of traffic incidents by predicting the future progression of traffic using road sensor data. Similar to our work, the system distinguish between different traffic events and the authors highlight the importance of including the event future impact in route planning. However, ClearPath does not collect data from drivers in a social network and the prediction is based on aggregated traffic data that is collected from traditional street sensors. Also, ClearPath alternative route advice is calculated using routing algorithms and it does not directly include local driver knowledge.

3 SYSTEM OVERVIEW

We propose a client-server architecture where the client is an application that can run on a modern mobile phone. Using the client, the user can ask for directions specifying the source and destination. The server has two components designed around two main functionalities: the Event Monitor and the Routing Unit. The system overview can be seen in Figure 2.

The Event Monitor goal is to keep track of active traffic events, and finding similar events from the events database. The server can receive event reports from traffic authorities. When the system receives an event report, it uses the events database to find the top most similar event from the past and keeps this information in the active event memory. The system will use the similar past event to retrieve historical trajectories from the same time as the event, and use those trajectories to build a transition model to be used by the routing unit later. In this work, we use a simple method for event matching and refine it in future work. To find similar events, we use hash indexing as there is easy division by (day, hour, location, etc.) that differentiates per event which trajectories are relevant. Therefore, we find a similar event, the first best match, and use it to retrieve a set of trajectories that are related to the event.

Once the server receives a routing request, it will determine if the user will encounter any of the active events along the way and pass the relevant transition model to the routing unit. The routing unit will initiate a graph search and use the transition model to assign costs to edges while searching for a path. The routing unit is composed of two main components: (i) transition store; and (ii) graph search component. **The transition store:** is an indexed data store with the goal of providing fast access to relevant transitions to be used during the search process. The transition store uses the connectivity of the graph as an indicator of locality. Specifically, the store uses the graph edges to access trajectories that pass through those edges at a certain time. Currently, the index is a basic memory mapping of edges to time-ordered transitions. We transform trajectories into transitions in a preprocessing step that is performed once for each event. Then, we group transitions based on the edge they pass. **The graph search component:** this component implements the leader following algorithm, a form of graph search that uses the transition store to retrieve and evaluate transitions for every edge.

4 ROUTING WITH HISTORICAL TRAJECTORIES

4.1 Preliminaries

Road Network: We represent the road network as a directed graph $G = (V, E)$ with a node set V of size n representing road intersections, and an edge set E of size m that represents road segments.

An edge $e = (u, v)$ where $u, v \in V$. Associated to each edge there is a weight function $w(e)$, that represents the cost of traversing this edge.

Path: A path P from node v_1 to v_k in G is a sequence of edges $e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow \dots e_{k-1}$ that connects the node set $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \dots v_k$. The weight of a path $w(P)$ is the sum of the weights for all edges in P. The shortest path between two nodes s and d is the path with minimum weight among all possible paths between s and d.

Traffic Event: Let X be a traffic event represented by the set $X = \{e, I\}$ for all $e \in E$ that are affected by the event X, and $I = \{i_t : t \in T\}$ is a time series that represents the impact of X on e during the time range of the event $\{T | T_s < T < T_e\}$ in terms of weights where T_s is the start time of the event, and T_e is the end time of the event, given increments of δ. Impact $i \in I$ is a value that indicates the increase in the weight $w(e)$. $E(X) \subset E$ is the set of edges that are impacted by the event.

Trajectory: A driver trip is collected in a GPS log. A GPS log is a sequence of timestamped GPS points $p_1 \rightarrow p_2 \dots p_k$. Each point $p = (latitude, longitude, t)$ consist of a coordinates pair and a timestamp $t \in T$. Using map-matching, points in a GPS log are transformed into trajectory Y by mapping the coordinates points to edges in E. Thus, a trajectory Y is a time ordered sequence of elements $Y = [y_1, y_2, \dots, y_k]$, where each $y = (e, [t_a, \dots, t_d])$ where $e \in E$ is the mapped edge, and t_a is the first timestamp that captures when e was entered and t_d is the last timestamp that captures when the edge was left. We use the notation $e(y)$ to indicate the edge e of element y, and $t_a(y)$ and $t_d(y)$ to indicate the arrival and departure time of edge $e(y)$ for element y. We assume that we are working with high frequency GPS logs that are enough to capture all edges in the graph. Thus, a mapped trajectory Y is a time ordered sequence of edges such that there exists a path $e(y_1) \rightarrow e(y_2) \dots e(y_k)$ in G. A trajectory set $A_{[t_s, t_e]}$ is a collection of trajectories $[Y_1, Y_2, \dots Y_j]$ that belong to the same time range $[t_s, t_e]$.

To avoid a traffic event, we need to find a similar traffic event from the past and follow the fastest drivers at that time to our destination. Drivers have a natural reaction to traffic; when drivers find about a traffic event, they strive to avoid it. Some of them may not succeed, but some drivers who have knowledge about the traffic pattern in the area will avoid it. Therefore, by looking at historical trajectories, we will identify the routes that can help us successfully avoid the event, and any future progression of traffic caused by the event.

However, when we search historical trajectories for a solution, we need to consider that the best route does not always map to a single trajectory. We cannot assume that all drivers in the past are going to the same source and destination as our queries. In other words, we need to consider routes that are formed from sub-trajectories. In these cases, the solution cannot be found by following a single driver, especially when the driver deviates from the query destination. Combining the trajectories of multiple drivers reveal better routes to take to avoid the event. Thus, combining several trajectories is necessary to find an optimal solution. In order to consider routes from sub-trajectories, we transform trajectories into transitions:

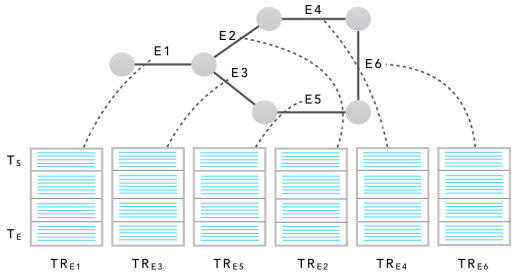


Figure 3: Transition Graph Example.

Transition: A transition is a tuple that represents a driver traversing from one edge to another. A transition $tr = \langle e, t_a, \lambda \rangle$ where $e \in E$, t_a is the arrival time and λ is the transition duration or travel time to another edge. Transitions are created from trajectories. Given a trajectory Y, we create the following transitions $\{(e(y_i), t_a(y_i), t_a(y_{i+1}) - t_a(y_i)) : y \in Y\}$, where $e(y)$ is the edge of element y, and $t_a(y)$ is the first timestamp for edge $e(y)$ in element y.

Transition Graph: Given the road network G and a trajectory set $A_{[t_s, t_e]}$, Transition Graph $\overline{G} = (\overline{E}, V)$ is a transition expanded version of G where for every edge $e \in E$ there is a new edge in $\overline{E}$ for every transition in the transition set $TR_{e,[t_s,t_e]}$ that group transitions which pass the same edge. Note that $\overline{G}$ is a temporal graph since edges in $\overline{E}$ are annotated with timestamps, therefore there might exist multiple edges between any two nodes in $\overline{G}$. Figure 3 shows an abstract example of a transition graph.

Transition Path: A transition path TP is a sequence of transitions $tr_1 \rightarrow tr_2 \rightarrow \dots tr_k$ such that $tr_1(e) \rightarrow tr_2(e) \rightarrow \dots tr_k(e)$ is a path in G and $t_a(r_{i+1}) > t_a(tr_i)$ for any $i \in [1, k-1]$. The cost of transition path TP is $\sum(\lambda(tr) | \forall\ tr \in TP)$.

Thus, based on the above definitions we can define our problem as finding the transition path that have minimal travel cost, we call this problem the Fastest Transition Path Query.

Fastest Transition Path Query: Given two nodes a source s and a destination d a transition graph $\overline{G}$ for the time range $[t_s, t_e]$, find the transition path TP with minimal travel time between s and d.

4.2 The Leader Following Algorithm

We present the Leader Following (LF) algorithm to solve the *Fastest Transition Path* Query. The LF algorithm modify Dijkstra's edge relaxation step to find and select the best transition while exploring nodes. The details of the algorithm is in Algorithm 1. The algorithm constructs paths with minimum travel time from source node to the remaining nodes, exploring adjacent nodes at each step until the fastest path to destination node d is found. The algorithm maintains a group of nodes in a priority queue Q to be processed. These nodes are the frontier of paths from the source node s that have been explored so far. The cost of each frontier is the sum of each transition travel time from the source node to the frontier. **Initiation:** first, the algorithm initiates a cost array $C[V] = \infty$ for all nodes and $C[s] = 0$. Node s is settled and added as frontier in Q. At each iteration, we extract the frontier u with the minimum cost and update the timestamp to $qt + C[u]$. **Edge relaxation:** for

every node v adjacent to u that has not been settled, we query the transition store to find the fastest transition for the edge (u, v) and the timestamp. A new cost from source is calculated using the transition travel time $C[u] + \lambda(tr)$. If this value is lower than the previous cost $C[v]$, then we update the cost of v to $C[u]+\lambda(tr)$ and set u as the predecessor node of v and tr as the predecessors transition. It is important to note here that each frontier in Q must maintains its own timeline. Otherwise, when the search backtrack or explore another path, it will use incorrect timestamp and retrieve transitions from a different period of time and this will effect the search results. Therefore, we calculate the timestamp used to query the transition store based on the cost of the frontier $C[u]$ and the query start time qt. This allows each frontier in Q to maintain its own timeline based on its cost so far. We call this mechanism *time stitching*, because we are stitching transitions together using their timestamp and travel time. Also, since we update the timestamp based on the travel time of the driver that we are following. Any other transitions from slow drivers will be filtered out and will not affect the search in the next steps. **Termination:** this process is repeated until the destination node is extracted from the queue. Finally, the minimum path can be recovered by running back from the destination node through stored predecessors untill the source is reached.

During edge relaxation, the transition store receives queries in the form of edge and timestamp $((u, v), t)$ and returns the transition tr with the minimum travel time during the time range $[t, t+r]$ where r is variable we use to control the time range of the query. In order to find the required transition, the store first select transitions with durations that overlap with the interval of the query. Then, it picks the transition with the minimum duration as detailed in listing 2.

To demonstrate how LF algorithm works, Figure 4 shows an example trajectory set (Figure 4(a)) and transition graph (Figure 4(b)). Let us assume that we have a query $q = (A, T, 2)$ going from node A to node T at time 2. At first, source node A is put into the queue Q. When A is extracted, the timestamp is set to 2, then the transition store is queried for $(A, B, 2)$ and $(A, C, 2)$ to evaluate the cost of B and C respectively. To answer the query $(A, B, 2)$ the transition store first filters all transitions for the edge (A, B) that are outside the time range $[2, 3]$. Then it will select the transition with minimum duration, in this case tr_3. Therefore, B will be added to the queue with the cost $C[B] = 0 + 1$. Similarly, C will be added to the queue with the $C[C] = 0 + 2$ coming from transition tr_5. Since B is the node with the smallest cost, B is extracted and the timestamp is updated to $2 + 1 = 3$ and the transition store is queried for $(B, D, 3)$. Then, D will be added to Q with the cost $C[D] = 1 + 1 = 2$ since the transition with the minimum duration under the time range $[3, 4]$ is tr_4. Next, C or D will be extracted, since both have the same cost they have equal chance to be extracted next. If D is extracted, the search is terminated. However, if C is extracted, a new cost for D will be calculated as $2 + 1 = 3$ which is greater than the cost stored for D. Therefore, the cost is ignored and D will not be updated. Finally, when D is extracted, the search is terminated with $A \rightarrow B \rightarrow D$ being the fastest path by following trajectory $Y_1 = (tr_3 \rightarrow tr_4)$.

Regarding the complexity of our algorithm, it is clear that our algorithm is more time consuming than Dijkstra due to the extra *findOrCreate* procedure we perform every time we relax an edge. The *findOrCreate* procedure performs two main operations sequentially: a filter operation and minimum operation. Let t be the average number of transitions per edge, then in worst case scenario each operation will be performed in $O(t)$. Therefore, we can assume that *findOrCreate* procedure complexity as $O(t + t)$ and can be simplified to $O(t)$. Furthermore, Dijkstra worst case complexity is known to be $O(m \log n)$ when implemented with a priority queue. Therefore, the complexity of our algorithm assuming t is the average number of traversals per edge is $O(mt \log n)$. However, it is important to note that t is dependent on the dataset used, specifically, the number of transitions.

Algorithm 1 The Leader Following Algorithm

```
1:  procedure QUERY(G = (V, E), s, d, qt)
2:      for v ∈ V do
3:          C[v] = ∞
4:      C[s] ← 0
5:      Q ← s
6:      while ¬Q.empty() do
7:          u ← Q.extractMin()
8:          if u == t then
9:              return u
10:         timestamp ← qt + C[u]
11:         for every node v adjacent to u not in Q do
12:             tr ← findOrCreate(u, v, timestamp)
13:             if C[u] + λ(tr) < C[v] then
14:                 C[v] ← c[u] + λ(tr)
15:                 v.tr ← tr
16:                 v.prev ← u
17:                 Q ← v
```

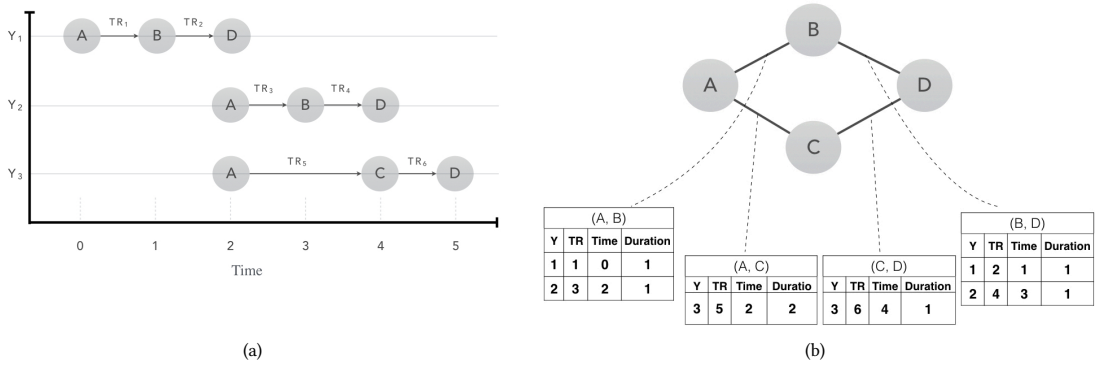
Algorithm 2 Finding transitions

```
1: procedure FINDORCREATE(u, v, t)
2:     TR_(u,v) ← store[u, v]
3:     tr ← min_λ{tr ∈ TR_(u,v) | t < (t_a(tr) + λ(tr)) ∧ (t + r) > t_a(tr)}
4:     if tr ≠ φ then
5:         return tr
6:     else
7:         return length(u, v)/speed(u, v)
```

4.3 Extending the Base Algorithm

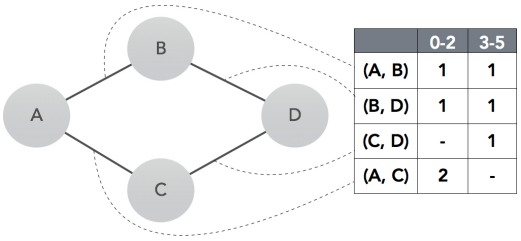
The LF algorithm is designed to be precise using realistic trajectories. However, keeping individual trajectories for many years will result in terabytes of data and this will increase the cost of storage and maintenance, also will make the search more expensive. For this reason, we present the snapshot routing algorithm which uses an alternative cost model for situations where only sparse trajectories are available, or if storing and using terabytes of individual trajectories is expensive for a particular company.

In this algorithm, we still operate on the same temporal graph used in LF algorithm. The difference here is the data store, instead



(a) (b)

Figure 4: From trajectories to a transition graph. A detailed example for the leader following algorithm.



	0-2	3-5
(A, B)	1	1
(B, D)	1	1
(C, D)	-	1
(A, C)	2	-

Figure 5: Timetable created using trajectories from Figure 4(a) with interval = 3 seconds.

of using the detailed transitions, we use a snapshot of the event created from event-specific trajectories only. The Snapshot Routing algorithm (SNP) uses trajectories to create a timetable for each edge in the road network that maps arrival time to a travel cost. Travel costs are aggregated from trajectory points that belong to the same time interval and pass the same edge. For example, assume we have edge e and the time interval is five minutes, then $timetable_e$ will contain the average travel cost for every five minutes aggregated from trajectories that pass e. Figure 5 shows an example of transforming the trajectories from Figure 4(a) into a time table with time interval equal to three seconds. We traverse the graph in a manner similar to LF algorithm, we keep the timestamp stitching mechanism (querying timetables based on the current timestamp). However, instead of querying the transition store, we query the edge related timetable and update the cost accordingly. The key here is that the timetables are created from event-related trajectories, we do not mix trajectories from other time frames to calculate the travel cost. Therefore, the travel costs still capture event spatio-temporal growth or evolution.

To compare the snapshot routing to the transition store in terms of size, let us assume that we have a trajectory set $Y_{[t_{min}, t_{max}]}$ which falls under the time range $[t_{min}, t_{max}]$. Let the time interval/rate be Δt and number of edges in the graph $|E|$. Then, we need $O(|E| * \frac{t_{max} - t_{min}}{\Delta t})$ pages to store time cost aggregates. The size of the timetables in this cost model depends on the time interval and the time range of the trajectory set. However, the size in the transition store depends on the number of transitions. To show this, assume that trajectories in Y all have the same length L_Y, then the transition store would require $O(Y * L_Y)$ pages to store all transitions. Therefore, the storage requirements for the transition store increase, when the number and length of trajectories increases. While, the timetables depends only on the time interval and range of the data, which is much smaller event-wise.

Although SNP reduces the size of the data and is less expensive than LF, it is not suited for finding personalised or individualised advice. By using aggregates, we lose the information about individual trajectories, and thus loss of this data will make profile based and other improvements to the system impossible in the future.

5 EXPERIMENTAL STUDY

Our main objective in this evaluation is to study the effectiveness of our proposed algorithms, LF and SNP, under different scenarios. Our goal is to show that our methods are effective at helping drivers avoid traffic events. We use the *Average Travel Time* (ATT) as the measure to evaluate the effectiveness of our approach. Although computation time is not our main focus in this paper we evaluate the performance of our algorithms using *Average Query Time* (AQT) as well. We compare our approach against two other methods: fastest path (FP) and real-time (RT) based methods. Both methods use Dijkstra as graph search approach and travel time as edge weight, but they differ in the way they estimate the travel time. The FP method is the conventional speed limits approach where the travel time is calculated based on the maximum allowed speed of the road. This is a basic approach and does not get traffic updates. The RT method estimates the speed of each road based on the average speed of vehicles traversing the road at a given time interval, and then computes the fastest path based on the estimated speeds. The RT method estimates and updates graph weights every five minutes based on speed samples collected during the past five minutes. Note that the FP approach represents the worst case scenario, since the weight is static and is not affected by traffic. Thus, this approach is not aware of any speed drop caused by the traffic event and will send vehicles to the event if the fastest path of the vehicle pass by the event. In addition, to test whether blocking the event location is enough to re-route from the event, we use a variation of FP approach (FP2) that is aware of the speed drop at the event location and re-route vehicles.

5.1 Evaluation Methodology

In our evaluation we use a microscopic traffic simulator called SMARTS [10]. The simulator is used to test our method as well as to generate synthetic trajectories. The simulator employs multiple traffic models (such as car-following and lane-changing), and the simulated vehicles obey various road rules. Thus, the simulator is capable of generating trajectories of realistic driver behaviour.

Simulation scenario: Using SMARTS, we run a simulation scenario using real road networks extracted from OpenStreetMaps (OSM) [8]. The simulation scenario is designed to test the effectiveness of a given routing method in helping drivers avoid traffic events. In this simulation scenario, we create a set of test vehicles that go through an event, and use a given routing method to re-route test vehicles. Then, we measure the simulated travel time using the trajectory of test vehicles. The simulation scenario is designed to be focused on the surrounding area of the traffic event. We fix the simulation time to one hour and the simulation region to the suburb that includes the event location. During the simulation, a traffic event will be activated on a selected road, and this will reduce the road speed according to the given event impact parameter. The speed drop caused by the traffic event is visible to all routing methods (except for FP) and when test vehicles are released into the road network they are re-routed using the routing method under test (FP, FP2, RT, LF, SNP). During the simulation, trajectories for test vehicles are recorded and exported. We run this scenario for the five routing methods under test, and use the trajectories of test vehicles to measure and compare the methods.

Test vehicles generation: We generate 100 test vehicles for each experiment. Test vehicles in the simulation are generated using the following steps. First, we randomly select source and destination pairs. Then, using the fastest path routing method we assign a route for each pair. Each route must go through the event location, or we replace the source-destination pair with a new one and find a new route. Next, each test vehicle is assigned a start time selected based on the experiment parameters (see Section 5.2).

Historical trajectory generation: We also use the simulator to generate trajectories needed by the LF method. To generate these trajectories we use a simulation scenario similar to the above, except that the event is not visible to vehicles and we do not re-route vehicles. First, we randomly select source and destination pairs from the nodes in the road network. Then, using the FP method we assign a route for each pair. Each route must have at least three or more turns, otherwise we replace the source and destination pair with a new one. Then, we run the simulator given the generated routes and export the trajectories. In this process, we do not control or alter the paths to avoid the event, and vehicles will always follow the fastest path at given time. We have considered including smart driver behaviour. However, since we do not have local drivers' data, we did not want to bias the experiment with unrealistic smart choices which may have a global view of traffic. Therefore, if there is a vehicle that is going towards the traffic event, it will get stuck and will not be re-routed. We thus rely only on having trajectories with different sources and destinations and our algorithm work by patching these trajectories to find out how to bypass the event. In real life we may get even better results as people do make better choices. Table 1 shows statistics about a historical trajectory set generated for one of our experiments.

Table 1: An example of synthetic historical trajectories used in one of our experiments.

Statistic	Value
Duration of Simulation	1 hour
Number of Trajectories	10155
Average Transition Time	11 seconds
Average Transition Distance	39.5 meters
Average Trip Time	367.68 seconds
Average Trip Distance	1356.79 meters
Average Trip Speed	3.6 m/s

Table 2: Experiment Settings.

Parameter	Default	Range
Event Location	Inner City	{Inner City, Outer Suburb}
Event Impact	%90	{50%, 90%}
Traffic Volume	1000	{500, 1000}
Peak Time	peak hour	{peak hour, off-peak hour}
Event Start Time	5	{5, 10, 15}
Release Time	8	{3, 8, 13}

Table 3: Details of the road network used in each map.

Map Name	Nodes	Edges	Area	Density
Inner City	3213	4293	2.4 km^2	1788.6
Outer Suburb	5697	9437	18 km^2	524.3

Finally, the procedure for evaluating a single experiment involves running the simulator six times. All six simulations are run according to the simulation scenario above, and they share the same conditions and parameters (Section 5.2) per experiment except for the routing method. We repeat each experiment five times and take the average ATT.

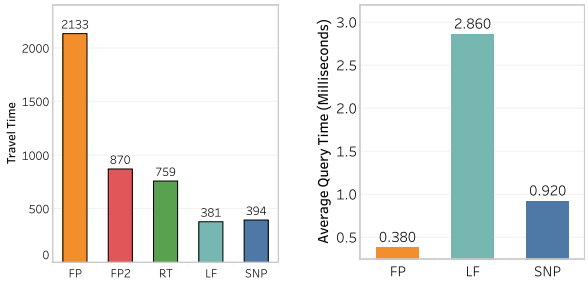
5.2 Evaluation Parameters

To evaluate our methods, we use six parameters that control and change how the traffic event propagates. The parameters are (i) Event Location, (ii) Event Impact, (iii) Traffic Volume, (iv) Peak Time for Event Location, (v) Event Start Time and (vi) Release Time of Test Vehicles. The settings for all experiments parameters are presented in Table 2 and explained below in details.

Event Location: We selected two different locations in two different areas. The areas were chosen from OSM of Victoria, Australia, to represent different road network layouts. The suburbs are: Melbourne City Centre (Inner City) and Ringwood (Outer Suburb). Table 3 lists the network characteristics for each suburb.

Event Impact: In these experiments, we vary the severity of the event impact. The impact variable has two levels: low and high. The impact of a traffic event is represented as a percentage of loss in speed for a specific road. For example, if an event have 90% impact on a road with max speed of $40km$ the road speed will be reduced to $4km$.

Traffic Volume: We test our method under two levels of traffic congestion: light and heavy. The traffic level affects the number



(a) (b)

Figure 6: (a) ATT values for the incident on inner city location. (b) Performance results for FP, LF and SNP using the inner city map.

of random vehicles in the simulation. The number of random vehicles remains constant during the simulation. For example a traffic volume of 1000 implies that there will be 1000 background vehicles at any time during the simulation.

Peak Time: While the traffic volume parameter controls the number of background vehicles, background vehicles are randomly distributed around the map (they have random source and destination), and they are not guaranteed to pass through the event location. This parameter helps us represent scenarios like rush hour traffic. If peak time is true, the flow through the event is increased by increasing the number of background vehicles that are headed to the event.

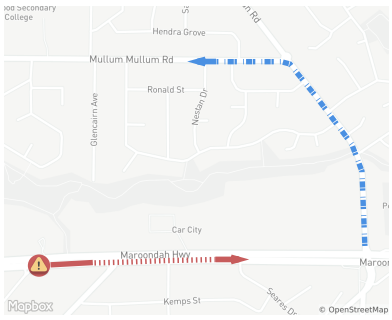
Event Start Time & Release Time of Test Vehicles: The event start time parameter controls when the event is triggered and activated during the simulation time. For example, when event start time is five minutes, the event will be activated and take effect after five minutes of the simulation start. Similarly, the release time parameter controls when the test starts. The release time indicates the time a vehicle gets released into the road network by the simulator. This is relative to the event start time. For example, if the release time is eight, test vehicles will be released into the simulation 8 minutes after the event start time.

6 RESULTS

In this section, we use a real world setting to test the scenario we presented in the introduction, where Bob was not able to escape the event impact even after being re-routed. In particular, we wanted to see how much of time saving we would achieve with our approach. In this experiment, the simulation was performed on Melbourne City Centre, with the event starting at minute 5, test vehicles are released after 8 minutes, with the event impact set to 90% and the traffic volume set to maximum of 1000 vehicles during peak time.

The result for this experiment is shown in Figure 6(a). The figure shows that the FP method has the longest ATT (2133 seconds) as expected. Also, the figure shows that FP2 has reduced the ATT to 870 seconds, which means blocking the event location help part of test vehicles but not all of them as the high ATT indicates that some vehicles still experienced a significant delay. Similarly, the RT method has ATT close to FP2 (759 seconds), which shows that

(a) Traffic incident for the Inner City location.



(b) Traffic incident for the Outer Suburb location.

Figure 7: The locations of traffic incidents used in our experiments, and examples of their future propagation. In 7(a), the traffic propagation will spread and block the closest alternative road (Flinder St.). In 7(b) the alternative road (Mullum Rd) is far from the event location and safe from future traffic propagation.

using the RT method has a small improvement over blocking the event location. Furthermore, the figure shows that LF and SNP methods reduced ATT significantly with ATT of 381 and 394 seconds for LF and RT respectively. To show the improvement our methods achieved over the RT method we present, in Table 4, the number of test vehicles that had better travel time using our methods and the time reduction achieved by our methods over RT. The table shows that LF method reduced the travel time for 329 vehicles by 47.5% on average. Similarly, the SNP method reduced the travel time for 321 by 46.6% on average.

The RT method did not perform well in this experiment because the propagation of traffic event was faster than the update rate of the RT method. This leads the congestion to spread to alternative routes used by the RT method and blocked all the vehicles re-routed via these alternative routes. This is visible in Figure 7(a). The figure shows the map of Melbourne City, the event is located on Collins St. (indicated by the caution mark) and the dashed arrows show the direction of traffic propagation. The RT method updates the graph every five minutes and routes vehicles based on data collected from the past five minutes. However, the RT method

Table 4: Improvement achieved by LF & SNP over RT.

Method	Number of Vehicles	Time Reduction
LF	329	47.5%
SNP	321	46.6%

is not aware of changes in traffic between updates nor can it estimate the future. In this case, the RT method was not proactive about traffic propagation towards Flinders St. and re-routed some vehicles via Flinders St. which lead these vehicle to get stuck in traffic again.

Furthermore, to show the runtime performance for our methods, Figure 6(b) presents the average query time (AQT) for FP, LF and SNP using the city map. The measurements focus on the query operation for the LF and SNP algorithms. The figure shows that LF method takes more time than SNP or FP. Nonetheless, the results show that the AQT for the LF method is 2.8 milliseconds which is acceptable to answer user queries. Also, AQT for the SNP method is 0.9 millisecond which is third the time of LF method. As we mentioned before, our algorithm is expected to be slower than Dijkstra, however, the results show that it is still within the accepted quality of service.

6.1 Elasticity

In this section, we study the effectiveness of our methods when the behaviour of the event changes based on the parameters shown in Table 2.

6.1.1 Effect of Event Impact. In this set of experiments, we vary the event impact parameter. Figure 8(a) shows ATT when we change the event impact. We notice that test vehicles do not experience any significant delays when the impact is 50%. This is clear because the ATT value for test vehicles when they go through the event without re-routing (FP method) is 564 seconds. This indicates that re-routing vehicles is not necessary since this setting does not disrupt the traffic significantly to cause congestion. On the other hand, increasing the event impact to 90% introduces a a considerable delay for test vehicles and in turn increases the ATT value for FP significantly (1830 seconds). In this case, the event causes congestion and will delay some vehicles that are being re-routed by the RT and FP2 methods.

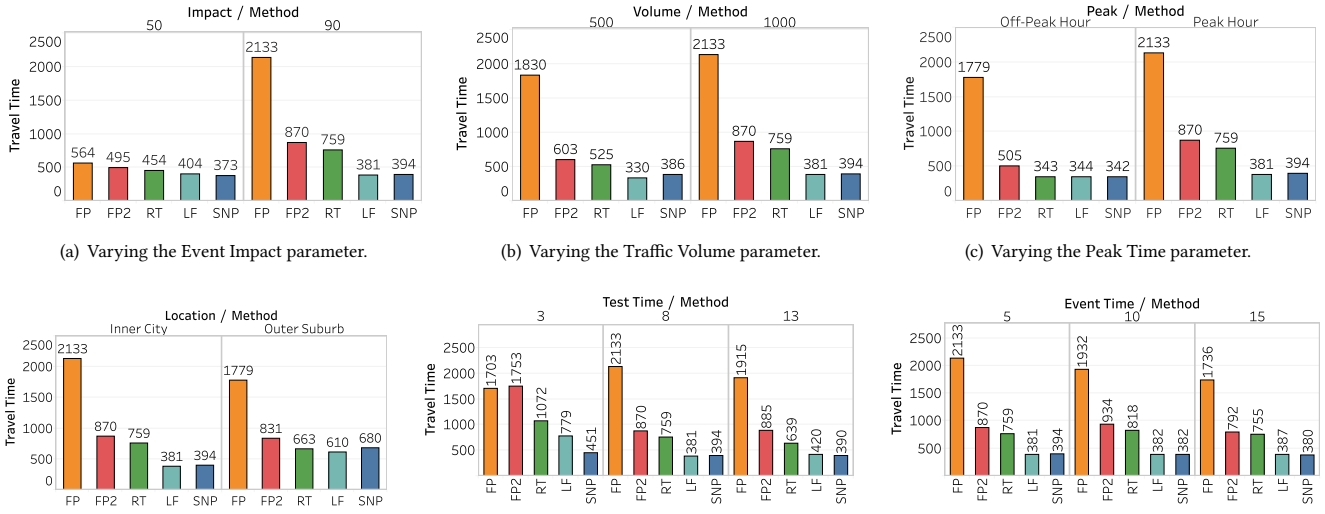
6.1.2 Effect of Traffic Volume. In this experiments, we reduce the traffic volume from 1000 to 500. Figure 8(b) shows ATT values when we change the traffic volume parameter. We notice that decreasing the traffic volume from 1000 to 500 slightly decrease ATT overall. This decrease occurs because the number of vehicles going towards the event has decreased. However, the change is small and suggests that traffic volume does not have a dramatic effect on the event as the event impact parameter does, unless a high volume of traffic is sent to the event directly.

6.1.3 Effect of Peak Time. We use the Peak Time parameter to increase the flow to the event and concentrate the vehicle distribution around the area. Figure 8(c) shows the ATT during Peak Hour and Off-Peak Hour. The figure shows that during off-peak hour, we can see that the RT and LF are almost equivalent in terms of ATT. This suggests that both methods managed to reduce ATT and successfully re-route test vehicles away from the event impact. During peak-hour, the increased flow to the event causes the congestion to propagate blocking nearby alternative routes and introducing significant delays to FP2 and RT. This is visible in the increase of ATT during peak-hour where ATT is increased by approximately 416 seconds for RT, while our method only experienced slight increase about 40 seconds. This indicates that our method has helped the drivers, who got stuck by the RT method during peak-time.

6.1.4 Effect of Event Location. In this experiment, we change the location of the event. Event location plays a big role on how the event behaves. Figure 8(d) shows the results of our experiments with two locations, Inner Melbourne City and Ringwood Suburb. The figure shows higher ATT overall for Ringwood. Furthermore, the figure shows that RT, LF and SNP for the Ringwood location is almost similar with ATT of 663, 610 and 680 seconds respectively. The reason for this is that traffic propagation does not reach the alternative route due to event being located on the middle of a long highway in Ringwood (as shown in Figure 7(b)). The capacity and speed of the highway makes traffic propagation slow (compared to the speed of a tertiary street in the city). Consequently, traffic propagation takes a long time to reach the roads used by the alternative routes.

6.1.5 Effect of Test Vehicle Release Time. In this experiments, we vary the Release Time parameter to test our methods during different phases of the event lifetime. Figure 8(e) shows ATT values when we change the test time to (3, 13). When the test vehicles are released after 3 minutes from the event start, the results show that FP method experience less ATT (compared to 8 minutes), a possible reason for this is that the event have just started and traffic has not yet started to build up. Therefore, vehicles that arrive first at the event location will experience shorter delays. We can also notice that the LF method has higher ATT values than when the release time is 8 or 13. This happens because at the start of the event there is no traffic congestion and the vehicles start to queue up slowly. In this initial period of the event, the delay experienced by vehicles varies, and this affects the LF method since it relies on following the fastest driver. However, this variation does not affect SNP method because it uses the average travel time instead of following the fastest driver. Furthermore, we can see that FP method experience shorter ATT when test vehicles are released after 13 minutes. This is probably because test vehicles were released earlier than other experiments and therefore spent more time waiting in congestion. Other than this, releasing the test vehicles after 13 minutes have similar trends to 8 minutes.

6.1.6 Effect of Event Start Time. In this experiment, we vary the start time of the event during the simulation. Figure 8(f) shows the ATT values when we change the start of the event during the simulation from minute 5 to minute 10 and 15. The results show that the ATT for FP2 is (870, 934, 792) seconds, and ATT for RT is (759, 818, 755) seconds when the event time is (5, 10, 15) respectively. While the ATT for LF is (381, 382, 387) and the ATT for SNP is (394, 382, 380) seconds. These results confirm that reduction of time achieved by our methods persist even when we change the



(a) Varying the Event Impact parameter.

(b) Varying the Traffic Volume parameter.

(c) Varying the Peak Time parameter.

(d) Changing the Event Location.

(e) Varying the Test Vehicle Release Time parameter.

(f) Changing the Event Start Time.

Figure 8: ATT under different parameter values.

time of the event. Thus, different start times have no major impact on the event behaviour or performance of our methods.

7 CONCLUSION

We presented two route recommendation algorithms (LF and SNP) that uses past trajectory data to help users avoid traffic problems. The algorithms can find the fastest route from past trajectories between two locations and a departure time. The LF algorithm is a form of graph search which uses a historical trajectories every time it relaxes an edge to find the fastest transition left by drivers. The LF algorithm evaluates routes that are constructed from complete trajectories or sub-trajectories, concatenation of different drivers trajectories. Our work can fallback to traditional road speed limit if there is a gap in the data. This ensures that LF algorithm can find a path even when there is no data for a particular road.

We presented an experimental evaluation done using micro-traffic simulation with multiple realistic scenarios. We reported on the effectiveness of our approach measured by the simulated travel time. In our study, we compared our approach with classical FP (speed road limits) method and real-time traffic analysis based (RT) method. The experiments show that the RT approach cannot estimate the future propagation of a traffic event. Therefore, it could lead drivers to congestion. Our method was able to suggest alternative routes that avoid the evolution of the traffic event and achieved up to 47.5% reduction in travel time on average. In addition, the RT method requires continues communication and frequent updates which is costly for both users and servers. Our method does not require updates and works as well as the RT in most situations.

In the future, we plan to focus on event similarity and matching. Currently, we assume a simple matching that uses trajectories from a similar event. We will work on more improved matching technique and add support for aggregating or combining trajectories from multiple similar events.

REFERENCES

[1] Panagiotis Bouros, Dimitris Sacharidis, Theodore Dalamagas, Spiros Skiadopoulos, and Timos Sellis. 2012. Evaluating path queries over frequently updated route collections. *IEEE TKDE* 24, 7 (2012), 1276–1290.

[2] Vaida Ceikute and Christian S. Jensen. 2013. Routing service quality - Local driver behavior versus routing services. In *IEEE MDM*, Vol. 1. 97–106.

[3] Vaida Ceikute and Christian S Jensen. 2015. Vehicle routing with user-generated trajectory data. In *IEEE MDM*, Vol. 1. IEEE, 14–23.

[4] Kai-Ping Chang, Ling-Yin Wei, Mi-Yeh Yeh, and Wen-Chih Peng. 2011. Discovering personalized routes from trajectories. In *ACM SIGSPATIAL*. 33–40.

[5] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. 2011. Discovering popular routes from trajectories. In *IEEE ICDE*. 900–911.

[6] Wuman Luo, Haoyu Tan, Lei Chen, and Lionel M Ni. 2013. Finding time period-based most frequent path in big trajectory data. In *ACM SIGMOD*. 713–724.

[7] Carl-Stefan Neumann. 2015. Mckinsey Report. (2015). http://www.mckinsey.com/industries/capital-projects-and-infrastructure/our-insights/big-data-versus-big-congestion-using-information-to-improve-transport [Online; accessed 16-May-2017].

[8] OpenStreetMap. 2017. OpenStreetMap Foundation. (2017). http://www.openstreetmap.org/ [Online; accessed 16-May-2017].

[9] Bei Pan, Ugur Demiryurek, Chetan Gupta, and Cyrus Shahabi. 2015. Forecasting spatiotemporal impact of traffic incidents for next-generation navigation systems. *Knowledge and Information Systems* 45, 1 (2015), 75–104.

[10] Kotagiri Ramamohanarao, Hairuo Xie, Lars Kulik, Shanika Karunasekera, Egemen Tanin, Rui Zhang, and Eman Bin Khunayn. 2016. SMARTS: Scalable Microscopic Adaptive Road Traffic Simulator. *ACM TIST* 8, 2 (2016), 26.

[11] Han Su, Kai Zheng, Jiamin Huang, Hoyoung Jeung, Lei Chen, and Xiaofang Zhou. 2014. Crowdplanner: A crowd-based route recommendation system. In *IEEE ICDE*. 1144–1155.

[12] TomTom. 2014. TomTom Online Navigation Services. (2014). http://www.tomtom.com/lib/doc/14Q3_US_A5hr.pdf [Online; accessed 16-May-2017].

[13] Ling-Yin Wei, Yu Zheng, and Wen-Chih Peng. 2012. Constructing popular routes from uncertain trajectories. In *ACM SIGKDD*. 195–203.

[14] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. 2013. T-drive: Enhancing driving directions with taxi drivers' intelligence. *IEEE KDE* 25, 1 (2013), 220–232.