ANTIGONE: Accurate Navigation Path Caching in Dynamic Road Networks leveraging Route APIs

Xiaojing Yu*, Xiang-Yang Li†, Jing Zhao‡, Guobin Shen§, Nikolaos M. Freris†, Lan Zhang†
*†*School of Computer Science and Technology, the CAS Key Laboratory of Wireless-Optical Communications, University of Science and Technology of China*, Hefei, China.
*yxjing@mail.ustc.edu.cn, †{xiangyangli, nfr, zhanglan}@ustc.edu.cn,
‡§*Alibaba Group*, Shanghai, China. ‡jing.john.zhao@gmail.com, §shen.sgb@alibaba-inc.com

IEEE INFOCOM 2022 - IEEE Conference on Computer Communications | 978-1-6654-5822-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOM48880.2022.9796817

***Abstract*—Navigation paths and corresponding travel times play a key role in location-based services (LBS) of which large-scale navigation path caching constitutes a fundamental component. In view of the highly dynamic real-time traffic changes in road networks, the main challenge amounts to updating paths in the cache in a fashion that incurs minimal costs due to querying external map service providers and cache maintenance. In this paper, we propose a hybrid graph approach in which an LBS provider maintains a dynamic graph with edge weights representing travel times, and queries the external map server so as to ascertain high fidelity of the cached paths subject to stringent limitations on query costs. We further deploy our method in one of the biggest on-demand food delivery platforms and evaluate the performance against state-of-the-art methods. Our experimental results demonstrate the efficacy of our approach in terms of both substantial savings in the number of required queries and superior fidelity of the cached paths.**

I. Introduction

Geographic information such as navigation paths, that is, the shortest paths corresponding to a set of origin-destination (OD) pairs, along with their corresponding travel times is the key information used for location-based services (LBS). Real-time accurate map services are instrumental for real-time decision-making in a wide spectrum of online applications, for example, package dispatching to couriers, route selection of drivers, carpooling, etc. [1]. This motivates an ever increasing demand for fine-grained geographic information.

Apart from a few giant LBS platforms such as Google, Apple, and Bing, that have sufficient resources to maintain and compute the paths on their own [2], it is common for an LBS platform, e.g., a small-size startup company to outsource the shortest path computation to external map service providers like Google Maps or Gaode [3] via querying their route APIs. However, such queries can be quite expensive; for instance, a given on-demand food delivery (OFD) company may need to spend more than one million US dollars yearly for such a service [1], [4]. As a consequence, it is imperative for LBS platforms to maintain a cache of navigation paths.

During the peak periods of LBS platforms, the average volume of path queries becomes especially large (e.g., over one million per second for an OFD platform in peak hour). With a cache of small capacity, the LBS platform's response speed will be limited by the query-per-second (QPS) threshold of the route APIs, which could result in non-responsiveness and thus very poor quality-of-service (QoS). To reduce the

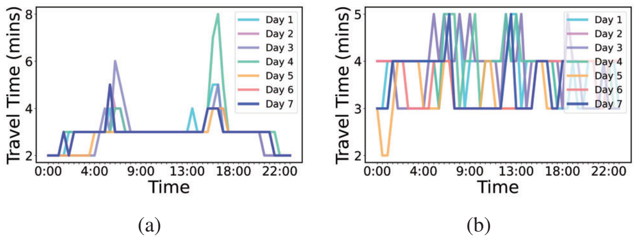


Fig. 1. Examples of travel time for two OD pairs obtained from route APIs with (a) regular patterns and (b) irregular fluctuations.

external map service cost and response latency, LBS platforms usually adopt a cache with large capacity for storing the set of all predetermined OD routes of interest. For each LBS request, the platform visits the local cache and uses the latest route to generate its response instead of querying the API in an attempt to bear very low response delay and no cost due to querying the map service. This differs from the classical caching paradigm that would advice to typically store only a small subset of OD paths based on their visit frequency [5].

In the face of variability in traffic dynamics, it is indispensable to update the cache by querying the route APIs so as to maintain fidelitous navigation path information. On account of the high volumes of data pertaining to a large-scale map and in view of the querying costs, it would be prohibitive and impractical to aim for maintaining up-to-date information in the bulk of the cache. To illustrate the standard practice regarding frequency of updating the cache, we have collected data from one large LBS company to find that more than 50% of the paths were not updated for more than *six months*, which incapacitates the provision of accurate navigation paths. Out-of-date inaccurate route paths largely affect the business of the LBS provider, via an increase in the experienced travel times compared to the best achievable ones. We illustrate with an example from the OFD scenario, where the platform gives users and couriers the estimated time of arrival (ETA) by leveraging the OD cache. In this setting, when ETA is less than the actual arrival time, users are dissatisfied because the delivery exceeds the expected time; in contrast, when the delivery can be completed before the ETA, the food might not be ready when the courier arrives at the restaurant.

There has been an increasing interest in navigation path caching. The majority of prior work on the subject aim to: a) maximize the hit-rate, i.e., the number of cache hits divided by the number of requests made to the cache, and b) minimize the delay and cost incurred by querying the APIs [5], [6]. In [7], the authors devised a query processing method by leveraging the *subpath property* proposed in [8] to improve cache look up performance and cache space utilization. However, as mentioned in the prequel, the small cache size used does not suffice to meet the stringent real-time requirements pertaining to large-scale LBS platforms. Li and Yiu [4] studied how to reduce the query response time while offering accurate query results for range queries and K-nearest neighbors queries; nonetheless, this approach is not suitable for our chief objective, i.e., shortest path queries. Their setup assumes that the local host holds the same graph as the map provider, and the routes obtained within a short duration (10 minutes in their setting) still provide accurate information to answer current queries. We observe that there exist evidently regular patterns of travel time obtained from the route API in most road segments (cf. Fig. 1); we specifically capitalize on this in designing our cache update solutions. Another challenge lies in the fact that, in reality, the LBS cannot obtain accurate maps from map providers: the map provider adds noise to the GPS coordinates in the returned navigation paths to prevent the leakage of accurate map information. Last but not least, note that shortest paths are calculated based on edge travel times. Notwithstanding, given the cache capacity limitations alongside the fact that query costs are invariable to the length (i.e., the number of edges) of the path, it would be unbearable to simply query edges in the road graph. Our proposed methods explicitly target to optimize the distillation edge travel time information using an adaptive path query mechanism.

In the face of all the aforementioned challenges and opportunities, we propose a suite of real-time cache updating strategies for navigation paths called **ANTIGONE** (Accurate Navigation paTh cachInG in rOad NEtworks) which aim to minimize the query costs while maintaining high fidelity of the paths in the cache. We construct a *distance-preserving hybrid graph* (that is, a compressed graph of the road network which approximately preserves the distances between the nodes [9]) from massive noisy navigation paths as the basic information structure underlying the cache update mechanism. We propose three dynamic graph maintenance algorithms each better suited for different settings namely: a) an Upper Confidence Bound (UCB) based algorithm, (Alg. 1) with $O(|\Psi||\overline{\pi}|)$ computational cost and $O(|\Psi|)$ storage cost, for platforms that have limited storage resources ($|\Psi|$ is the number of OD pairs in the cache and $|\overline{\pi}|$ is the average number edges per navigation path); b) a predictor-based method (Alg. 2) that updates the dynamic graph leveraging a combination of forecasting the real-time travel times based on historical information and queries from the route API; c) an extended update algorithm especially tailored for the OFD scenario (Alg. 3) where the LBS platform may aggregate geo-tagged data (which provides an opportunity to obtain the latest navigation path): this is especially useful for the scenario that navigation paths may not be in alliance with real paths (e.g., many couriers violate the traffic regulations during delivering).

The proposed solutions have been deployed at one of the world's largest OFD platforms, to efficiently manage a cache containing billions of navigation paths. Our experiments using a large-scale dataset from the real-life application demonstrate that our approach outperforms state-of-the-art solutions in terms of both the number of required queries and the fidelity of the cached paths. In specific, ANTIGONE achieves at least 2x higher accuracy using only 0.1x querying costs of the commonly used state-of-the-art strategy.

The main contributions of this paper enlist:

- We propose mechanisms for updating a large-scale navigation path cache by leveraging the regularity of travel time patterns.
- We propose an approach to construct a hybrid graph based on large volumes of noisy navigation paths, as a compression mechanism of the primitive Big Data pertaining to the road network state description.
- We propose three dynamic graph maintenance algorithms each better suited for different infrastructure settings.
- We evaluate our method using data from one of the world's largest OFD platforms. Our experimental results attest to significant improvement in terms of higher accuracy and lower costs over the current state-of-the-art.

II. Problem Statement

In this section, we provide a formal definition of the cache updating problem and further introduce related concepts and models regarding the OD cache updating system.

A. Dynamic Road Network

Points of interest (POI), e.g., residential or commercial locations, as well as intersections with heavy traffic, constitute the vertices of the road network graph; each navigation route between two vertices that is not passing through other vertices is an edge, whose weight is the travel time. The reality of frequent changes in traffic conditions is captured by a time-varying edge weight set. To sum up, the dynamic road network is modeled as a directed connected weighted graph, denoted by $G_t = (V, E_t, W_t)$, where V is the (fixed) vertex set, E_t is the edge set (possibly time-varying due to occasional additions or deletions of edges), and W_t represents the time-varying edge weight set (weights being the travel times that may feature high variability). Given two vertices corresponding to an origin and destination, $o, d \in V$, let $\pi_{G_t}(o, d)$ denote the shortest path in graph G_t, i.e., the path with minimum sum of weights of the edges comprising the path, denoted as $w(\pi_{G_t}(o, d))$.

B. Query & OD Cache

As mentioned above, apart from a few giant LBS platforms that have sufficient resources to maintain and compute the paths on their own [2], it is common for an LBS platform to outsource the shortest path computation to external map service providers. In this work, we consider the shortest path query at timestamp t, i.e., the path with shortest travel time for a chosen OD pair. The query response contains two types of information: (1) the navigation *path* between o and d, i.e., $\pi_{G_t}(o,d)$ and (2) the *travel time* of every edge in $\pi_{G_t}(o,d)$. Although the weight returned by map service providers may not be exactly equal to the ground truth, it still remains the most efficient source for encapsulating dynamic traffic conditions. The LBS platform must pay for the query, and we assume that the price of any such query is the same.

To cut down the cost and response time due to map service queries, LBS companies usually cache previous queries of all usable OD pairs in an *OD Cache* system. We use Ψ to denote the set of OD pairs stored in the OD cache. Each OD pair $\{o,d\}$ in Ψ is equipped with a parameter $\eta(o,d)$ which measures how frequently $\{o,d\}$ is queried, i.e., it is defined as the ratio of the number of queries for the OD pair divided by the total number of queries. The navigation path between $\{o,d\}$ stored in the OD cache at timestamp t is denoted as $\psi_t(o,d)$, and its corresponding travel time is denoted as $l(\psi_t(o,d))$.

C. Problem Definition

Due to the highly dynamic nature of traffic in dense urban areas, the OD cache needs to be updated frequently enough to accurately reflect the dynamic traffic changes. The QPS limit of the map provider along with the available total query cost budget of the LBS platform are captured by a constraint that the number of queries to the map service provider is limited during some fixed time-window. The main intent of the cache maintenance system lies in resolving the information inconsistency between cached OD weights and those pertinent weights to the map service provider, i.e., to keep the OD cache information $\pi_\Psi(o,d)$ as close as possible to the corresponding real-time information on the external map service provider $\pi_G(o,d)$. We let $\Delta(o,d)_t := |w(\pi_{G_t}(o,d)) - l(\psi_t(o,d))|$ denote the difference between the path times in the cache and map provider at timestamp t.

The formal definition of the problem that we consider in the paper is as follows. We define the *inconsistency error* of the OD cache as the weighted average of the OD differences, with weights given by the path frequencies:

$$\gamma_t = \sum_{\{o,d\}\in\Psi} \eta(o,d)\cdot\Delta(o,d)_t. \tag{1}$$

We aim to design a method for updating Ψ over a period of T time units so as to minimize the average inconsistency error $\frac{1}{T}\sum\gamma_t$ subject to a fixed available budget of queries.

III. Solution Overview

Most studies in the field of navigation path caching have only focused on leveraging the road network [4], [5]. However, in reality, LBS cannot obtain accurate maps from map providers: for the sake of information protection, the map providers adopt anti-crawler mechanisms by adding noise to the GPS coordinates in the returned navigation paths to prevent the leakage of accurate map information. Besides, in view of our cache capacity limitations (given the huge number of edges in the road network), the query costs generated by updating individual edges would be unbearable. In Section IV, we propose our solution to build a hybrid graph with smaller number of vertices and edges than the original road network, as the integral component of our cache representation and update mechanism. Travel times are affected by real-time traffic conditions. For example, rush hours and traffic accidents significantly increase the extent of traffic congestion and, consequently, travel times. LBS may be unable to estimate travel times of road segments accurately based solely on historical responses. In specific, the key challenge is to exploit the historical travel times to effectively trade-off cache accuracy for incurred query costs. In Section V, we propose three dynamic graph maintenance algorithms each better suited for different resource settings. Next, we demonstrate how to use the riders' trajectory as the update seeds for the OFD scenario in Section VI.

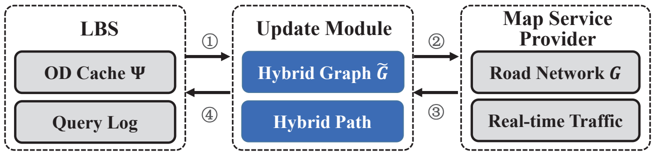


Fig. 2. Workflow of ANTIGONE.

The workflow of our proposed solution is illustrated in Fig. 2 and comprises the following four steps:

Step 1: The LBS platform initially projects all navigation trajectories into the hybrid graph to obtain *hybrid paths* according to vertex partitioning (cf. Sec. IV) to reduce the storage and computational overhead.

Step 2: The update module selects OD pairs from the graph maintenance algorithms and sends requests to the map service provider.

Step 3: The map service provider responds navigation paths and travel times based on the real-time traffic in the road network.

Step 4: The hybrid graph is updated based on the responses and stores the latest navigation paths and corresponding travel times in the OD cache.

IV. Model Construction

In this section, we demonstrate our method to build the hybrid road network; this constitutes a distinctive attribute of our system design that aims to reduce the computation overhead.

A. Road Network Reconstruction

A geographic waypoint in the road network can be a turning point, a road end, or a cross point. The navigation trajectory is stored in the cache as a sequence of GPS points, where each point represents a geographic waypoint. Due to the noise added, a geographic waypoint may correspond to one of hundreds of actual GPS coordinates. Therefore, the approach we adopt considers reconstructing the road network from cached navigation paths; this is an especially challenging problem in light of the large number of OD pairs (in the order of several billions for middle-sized cities in the OFD scenario). A considerable amount of literature has focused on the problem of building the road network from trajectories using techniques such as trajectory clustering [10] or pattern mining [11]. Davies et al. [12] proposed a low complexity Kernel Density Estimation (KDE)-based map construction algorithm: it first computes the path density for each grid cell and, subsequently, capitalizes on this to infer a center line representation of the contour of the density map. We regard the navigation paths as trajectories and adopt the graph generating algorithm proposed in [12] which generates a small map with fewer number of vertices and edges.

B. Hybrid Graph Construction

The vertices in the road network graph represent two kinds of locations: POIs and waypoints. In contrast, the vertices in OD pairs are all associated only with POIs. This simple fact motivates the introduction of a *hybrid graph* with a substantially smaller number of vertices compared to the original road network, in order to allow for manageable computational overheads.

Formally, we denote the hybrid road network as $\tilde{G}_t = (\tilde{V}, \tilde{E}, L_t)$. $\tilde{V}$ is a partition of V, i.e., each vertex $\tilde{v} \in \tilde{V}$ is a subset of V. There are two types of vertices in $\tilde{V}$: POI vertices and waypoint vertices. The hybrid POI vertices in $\tilde{V}$ only contain POIs in the road network, and, similarly, the hybrid waypoint vertices only contain waypoints. This setting is to facilitate the transformation of the original navigation path to the corresponding path on the hybrid graph, which we call *hybrid path*. $\tilde{E}$ is the edge set, in specific, two hybrid vertices are connected with an edge if any two original vertices therein are connected in the road network. We use $l_t(\tilde{e}) \in L_t$ to denote the weight of edge $\tilde{e}$, which corresponds to the latest travel time between the hybrid points. The graph is dynamic in that the weights L vary over time primarily due to variability of the traffic conditions but also because edges in $\tilde{E}$ may be added or deleted due to road developments.

The central issue of generating the hybrid graph is to produce the partition of vertices. Due to the random noise in the GPS points, we have observed that points of the road network are denser at junctions. A natural choice to create the partition of V is to invoke a density-based clustering algorithm such as DBSCAN [13]. However, our problem differs from traditional graph compression. Recall that the objective is to keep the navigation distance in the hybrid graph as close as possible to the one in the original graph. To this end, although applying clustering methods on all the vertices directly can produce a small number of hybrid nodes, this does not guarantee accurate preservation of navigation distances. For this purpose, we propose a Group Clustering method to generate the hybrid graph (for its efficacy in terms of distance preservation) by dividing waypoints and POI vertices and adopting different clustering goals respectively; cf. Fig. 3.

Hybrid waypoint vertices: We apply the DBSCAN clustering algorithm on all waypoints which serves to filter out the noise. It is worth mentioning that it is quite costly to acquire the navigation distance between any two points querying from the route API. In view of the fact that the road network structure can be captured well by use of grid lines in many cases of interest (for example, in city centers), we use the Manhattan distance as the distance metric for clustering the vertices. Fig. 4(a) shows the average gap (mean absolute error) as a function of the navigation distance when clustering using three different distance metrics: this illustrates the advantage of using the Manhattan distance. The GPS coordinate of a given hybrid waypoint vertex is taken as the mean of the coordinates among vertices in its corresponding cluster. Here, we keep every outlier vertex obtained from DBSCAN and regard it as a hybrid vertex. Then, we simply connect the hybrid waypoint vertices that have edges between their corresponding original vertices in each cluster.

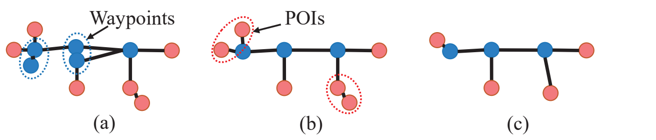


Fig. 3. An illustration of hybrid vertex generation: (a) we first cluster the waypoint vertices that correspond to the same geographic waypoint (e.g., road junction); (b) we cluster POI vertices that are close to each other to a hybrid vertex (e.g., residential community); (c) the hybrid graph generated.

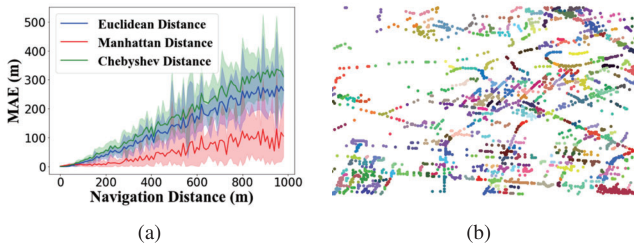


Fig. 4. Vertex partitioning: (a) Error for distance metrics; (b) Illustration of the vertex partition.

Hybrid POI vertices: After partitioning the POI vertices into hybrid vertices, edges between POI vertices within hybrid POI vertex are removed from the hybrid graph, i.e., these edges will not be updated. As a result, it is required to keep the length of the eliminated edges as small as possible, which is different from waypoint vertex clustering. We adopt the K-means-based clustering method proposed in [14] which achieves the best performance and construction speed in our setting. In each cluster, we use the median of the coordinates among vertices as the GPS coordinate of the resulting hybrid POI vertex, as this attains the minimum sum of Manhattan distances of those excluded edges.

After obtaining the vertices and edges of the hybrid graph, we use the travel time of the latest navigation path between two hybrid vertices as the weight of that edge. Since the navigation path is needed in services, we further maintain the original navigation trajectory as an attribute of the hybrid edge.

C. Hybrid Path Production

We project original paths onto the hybrid graph based on the above clustering results, i.e., the partition of vertices, to generate *hybrid paths*. The transition from hybrid paths to navigation trajectories is rather straightforward by substituting the navigation trajectory of hybrid edges into the hybrid paths.

The hybrid paths require significantly less storage space than the original navigation paths. Moreover, the original navigation paths may feature a significant level of inconsistency as compared with the latest information due to the invoked update strategy. Because of the different update delays, two distinct historical navigation paths which pass the same two points, might have different road segment and travel time. We seek to alleviate such inconsistencies by only keeping the latest road segment information in the hybrid edge.

V. Dynamic Graph Maintenance

In this section, we introduce the process to maintain the dynamic graph.

A. Edge Weight Updating

As mentioned in Sec. II-B, by querying the latest navigation path for a given OD pair from the map service, it is possible to obtain the latest edge weights pertaining to the path by projecting the navigation path onto the hybrid graph. In light of querying fees paid to the map service provider, the main objective of graph maintenance is to keep the weights in $\tilde{G}$ as compatible as possible with the latest weights in G while minimizing the incurred query costs. Recall that updating the cache frequently is costly, which we capture by periodically querying the map API and then updating the OD cache, using a query budget of B queries per round (i.e., a total budget of $B \cdot T$).

A central question is how to attest to the benefit of choosing a given OD pair, i.e., the set of edges contained in the returned navigation path, versus another one. In several cases of practical interest, the weights (travel times) change slightly, whence it is plausible to hypothesize that the hybrid paths are less likely to change and only the weights of the edges in the path are different. This, in turn, implies that we may infer edges comprising each OD pair directly from the locally cached hybrid graph. the problem of select OD pairs for minimizing the aggregated OD differences, can be cast as set cover problem, which is NP-Hard [15]. For this problem, a greedy algorithm that sequentially selects OD pairs with the largest marginal gain typically outputs a good approximate solution [16].

Unfortunately, in cases of severely abrupt changes of travel times, it may be infeasible to deduce the shortest paths from the local hybrid graph. A reasonable approach to tackle this issue could be to extract some patterns of weight change from historical information. Previous studies have verified that the change of traffic times is generally not that irregular, and the traffic fluctuation period in most road segments is usually one day [17]. This motivates the use of forecasting, which we consider in Alg. 2. When it is not possible to predict the traffic time of every edge based on historical data with high accuracy (e.g., due to limitations in storage and computational overheads), the historical queried weights may serve as useful proxies to select the OD pairs, as in our model-free approach in Alg. 1.

Algorithm 1: Model-free ANTIGONE

```
1 Initialize: For each $e_i \in \tilde{E}$, observe latest weight $w_{i,0}$
    from Oracle, set $p_{i,0} = \frac{1}{|\tilde{E}|}$. Generate $\tilde{G}_0 = (\tilde{V}, \tilde{E}, L_0)$,
    where $L_0 = \boldsymbol{w}_0$.
2 for $t = 1$ to $T$ do
3     $(\mathcal{Q}, \tilde{G}_{t+1}) = OD\text{-}PairQuery(\boldsymbol{p}_t, \tilde{G}_t, B)$.
4     Obtain $\boldsymbol{p}_{t+1}$ via Eq. (2).
```

Algorithm: OD-PairQuery

```
  Input: Edge probability $\boldsymbol{p}_t$, hybrid graph $\tilde{G}$, and Query
         budget $B$
  Output: Updated edge set $\mathcal{Q}$ and corresponding hybrid
          graph $\tilde{G}$
1 $\mathcal{Q} = \emptyset$.
2 for $j = 1$ to $B$ do
3     Compute $p(o, d)$ for $\{o, d\} \in \Psi$ via Eq. (3).
4     Select $\{o_j, d_j\} \backsim \boldsymbol{p}_t$.
5     Observe latest weight of edge set $\mathcal{Q}'$ in the
        navigation path of selected $\{o_j, d_j\}$ by querying
        from the Oracle.
6     Update $\tilde{G}$ by setting $l_i = w_{i,t}$ for $e_i \in \mathcal{Q}'$, and set
        $p_{i,t} = 0$.
7     $\mathcal{Q} = \mathcal{Q} \cup \mathcal{Q}'$.
8 return $\mathcal{Q}$, $\tilde{G}$
```

Model-free Update Algorithm: Firstly, we consider the simpler case that we only query a single edge each time instead of OD pairs with multi-hops. We do so to illustrate the design for this simpler case, which we proceed to extend to multi-hop OD pairs next. We model the problem as a budget-limited Multi-Armed Bandit (MAB) problem [18], where a) each edge is treated as an arm, b) the change of edge weights (as captured by the squared difference between the queried and local travel times) is seen as the reward, and c) each query of external route API is seen as the act of pulling a corresponding arm. In this model, B arms will be played in each round. The Upper Confidence Bound (UCB) [18] algorithm is widely used to solve the traditional MAB problem and a number of extensions leverage UCB to estimate the unknown reward and balance the regret subject to budget constraints [19]. We propose an UCB-based algorithm (Alg.1) to maintain the dynamic graph querying edge weights from route API.

We introduce $\alpha_{i,t}$ to denote the importance of an edge, which is calculated by the number of paths that traverse it. A natural design concept that we adopt instructs that the edge query frequency be increasing in: a) the observed edge weight change and b) its importance. Here, we use $|w_{i,t} - l_i|^2$ to represent the reward of updating edge i and $\hat{\mu}_{i,t}$ to represent the average reward observed from edge i. Next, we use $p_{i,t}$ to denote the probability of selecting edge i at time t, which is updated as:

$$p_{i,t} \propto \alpha_i \left(\hat{\mu}_{i,t} + \sqrt{\frac{2 \ln t}{n_{i,t}}} \right), \tag{2}$$

Algorithm 2: Predictor-based ANTIGONE

```
1 Initialize: For each $e_i \in E$, observe latest weight $w_{i,0}$
    from Oracle, let $p_{i,0} = \frac{1}{|E|}$. Build edge weight
    predictors from historical data. Generate
    $\tilde{G}_0 = (\tilde{V}, \tilde{E}, L_0)$, where $L_0 = \boldsymbol{w}_0$.
2 for $t = 1$ to $T$ do
3     Compute $\hat{\boldsymbol{w}}_t$ from edge weight predictors.
4     Set $L_t = \hat{\boldsymbol{w}}_t$.
5     $(\mathcal{Q}, \tilde{G}_{t+1}) = OD\text{-}PairQuery(\boldsymbol{p}_t, \tilde{G}_t, B)$.
6     Update predictors based on $L(\mathcal{Q})$ in $\tilde{G}_{t+1}$.
7     Obtain $\boldsymbol{p}_{t+1}$ via Eq. (2).
```

where $n_{i,t}$ is the number of pulls of arm i until time step t. We use $\propto$ to ignore the normalization constant required to make $\boldsymbol{p}$ (the stacking of $p_{i,t}$) a probability mass function (PMF) and follow the same abbreviation in the remainder of the paper, for ease of exposition.

We proceed to describe the extension to selecting OD pairs (Alg. *OD-PairQuery*). In reality, the cost of querying a path (comprising a number of edges) is the same as querying a single edge, which would output the weights of individual edges. For this reason, we target to devise a path querying mechanism based on the following observations. First, it is advantageous to choose paths with long edge sequences for covering more edges. One option would be to treat each OD pair as an arm and capitalize on the same method discussed above. Nonetheless, the number of OD pairs may be really large to the extent that the initialization step of inserting OD pair information in the cache may incur an unbearable query cost. Given the probability of selecting edges in Eq. 2, we use $p(o,d)$ to denote the probability of selecting OD pair $\{o,d\}$, which is defined as:

$$p(o,d) \propto \sum_{e_i \in \pi_{\tilde{G}_t}(o,d)} p_{i,t}, \tag{3}$$

where $\pi_{\tilde{G}_t}(o,d)$ is the shortest path stored in the OD cache. Algorithm *OD-PairQuery* describes the detailed procedure of our path selection method. In a given round, we query paths sequentially. We temporally set $p_{i,t} = 0$ for each e_i that has been updated in one round and update $p(o,d)$ using Eq. (3) without re-computing the shortest path, in order to reduce the computational overhead at step 3. At Step 4, we choose the OD pairs following $p(o,d)$.

Alg. 1 describes the procedure of our model-free graph maintenance algorithm. It requires $O(|\Psi|)$ storage cost and a per-round computational cost of $O(|\Psi|\overline{|\pi|})$, where $\overline{|\pi|}$ is the average edge number over the navigation paths stored in the OD cache.

Predictor-based Update Algorithm: There has been extensive work on predicting travel times [20], [21]. The pattern of travel time may be irregular [21], therefore, in some cases, it is impossible to achieve accurate prediction only based on historical data. Nevertheless, this is rarely the case and, for stable edges (which were found to be at least 80% of the total in our datasets), we can leverage simple predictors to estimate the real-time travel times and use the predicted value as the proxy of the ground truth to avoid the cost of querying from the map service provider. Following the same selection rules mentioned above, in this case an edge will be queried with higher probability if the predictor is less "accurate". We use the squared prediction error $|w_{i,t} - \hat{w}_{i,t}|^2$ to represent the reward of updating edge i, where $\hat{w}_{i,t}$ denotes the estimated value from the predictor, and apply Eq. (2).

The algorithmic description is exposed as Alg. 2. The storage cost of Predictor-based ANTIGONE is dominated by the size of the LSTM predictor [21] and the historical data needed for prediction. In our practice, it is not necessary to build a predictor for each edge, and we resort in reducing the number of needed predictors by clustering edges according to their change pattern (e.g., we can record the day-long travel time of edges and cluster the edges based on the travel times). Thus, the computational complexity is dominated by steps 3 and 6 in Alg. 2 and is of the order of generating the predicted edge weights and updating the predictors. In practice, we use a data pool to collect the edge weights obtained from route APIs and then update the predictors to save the computational overhead.

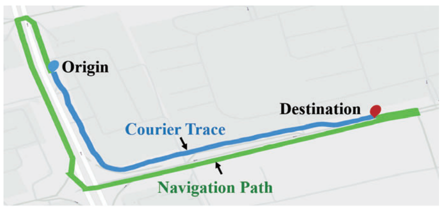
B. Hybrid Path Updating

In view of the storage cost and computational overhead pertaining to hybrid paths, we efficiently maintain them as shortest path trees. Instead of updating the hybrid paths by computing the shortest path tree for all OD pairs, which would require very large storage cost, we rather refresh all shortest paths whenever a new edge update becomes available. We adopt the dynamic shortest path algorithms proposed in [22] to maintain shortest paths as hypergraphs, which achieve a low computational overhead when the dynamic change is large and affects many edges.

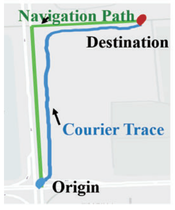
Moreover, we introduce the notion of *Simplified Update*. Instead of re-calculating the shortest path in the hybrid graph, we assume all paths stay approximately invariant after an edge update, i.e., only the weights of edges change dynamically. In the event that the path distance after recalculating is too large compared with the original path distance, the shortest path between a pair of nodes may switch to a new shorter path. Thus, we only update the paths with shorter re-calculated distance than the original distance instead of scanning the whole graph. We opt to update the accurate shortest paths of all OD pairs in the cache after τ rounds of simplified updates, considering the trade-off between the computational overhead and performance.

VI. Application to the OnlineFood Delivery Problem

In this section, we introduce the application of our method to the OFD problems, for which we apply modifications targeted to deal with the practical challenges pertaining to this real-life scenario.



(a) Courier rides in the opposite direction (i.e., not complying to the driving regulations)



(b) Courier rides along the path with deviations

Fig. 5. Two illustrative navigation paths and courier traces.

Couriers mostly deliver food by motorcycle or electric bicycle, as these are much more flexible than cars in avoiding congestion. The travel times are highly dependent on the driving behaviors of the couriers, which is difficult to estimate considering the diversity of driver behaviors [23]. A positive attribute is that there is lower congestion in bike lanes (when available) compared with the driveway. Hence, the navigation paths constitute an important factor to consider when OFD platforms make planning decisions such as selection of the couriers.

As the bike lane infrastructure changes frequently, the navigation path between two nearby locations will also change. In order to deliver a package on time, the courier will choose the shortest path based on their familiarity with the area. However, in some areas that are poorly supervised by traffic control, couriers tend to violate the traffic rules so as to reach their destination faster and/or minimize fuel consumption. This phenomenon tends to be the rule rather than the exception in the real-life, albeit the fact it is by no means encouraged by the platform. It is hard to distinguish whether couriers violate traffic rules when there is a mismatch between the courier trace and the navigation path; Fig. 5 illustrates two commonplace cases. Therefore, every update of the navigation path cache should be queried from the map service provider so as to guarantee the security of couriers.

Here, we extend our cache update model to account for these particularities of OFD; the weight of an edge is taken as the length of the navigation path between two points instead of the travel time.

A. Courier Traces

Many LBS platforms usually generate and aggregate geo-tagged data, which provides an opportunity to estimate the latest navigation paths. We introduce the detailed process to obtain the latest navigation path from courier traces in the following.

We obtain the day-long moving trajectories of all available couriers from the courier mobile devices. Since courier traveling patterns typically feature a large extent of flexibility and variability, a courier may detour during a task due to various reasons (e.g., help other couriers) so that the resulting sensing data may turn quite noisy. Even with full information of the delivery tasks, it is challenging to discover legible traces from origin to destination. Getting on and off typically occurs in two cases: when the courier departs from its origin and

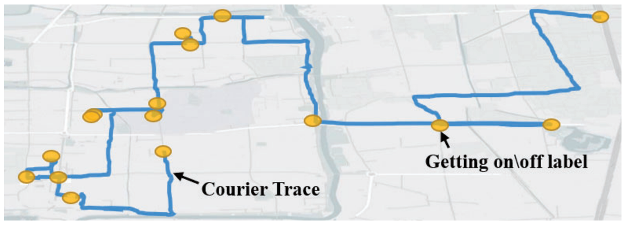


Fig. 6. An example of courier daylong trajectory with getting on-off labels.

Algorithm 3: ANTIGONE for OFD

```
1 Initialize: For each $e_i \in E$, observe latest weight $w_{i,0}$
    from Oracle, set $p_{i,0} = \frac{1}{|E|}$. Generate $\tilde{G}_0 = (\tilde{V}, \tilde{E}, L_0)$,
    where $L_0 = \boldsymbol{w}_0$. Set $\theta \in (0, 1)$.
2 for $t = 1$ to $T$ do
3     Compute $\lambda_i$ from courier traces and set
      $\hat{\mu}_{i,t} = \frac{\lambda_{i,t}}{\max_j \lambda_{j,t}}$.
4     Obtain $\boldsymbol{p}_t$ via Eq. (4).
5     $(\mathcal{Q}, \tilde{G}_{t+1}) = OD\text{-}PairQuery(\boldsymbol{p}_t, \tilde{G}_t, B)$.
```

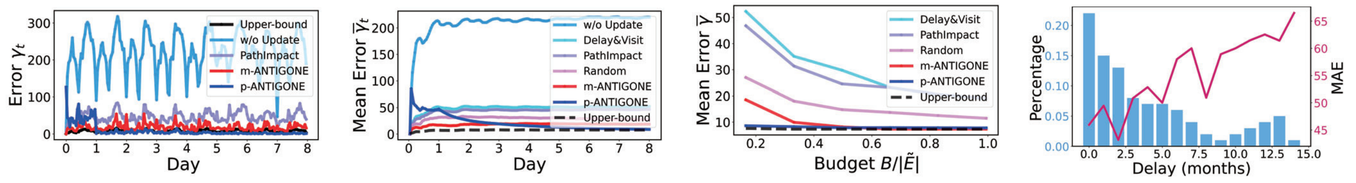
when he/she reaches the destination. We adopt an activity recognition LSTM-based neural network as proposed in [24], with inertial measurement unit (IMU) signals as inputs and outputs the labels of whether the courier is riding. Besides, we set the GPS speed threshold for more accurate judgment, i.e., when the riding speed is less than 2.1 m/s, we label the courier as getting off the bike. Here, we take the shortest path from a starting point to a destination as the one chosen by the courier. After trajectory segmentation, we cut off outliers by calculating the point-to-point travel speed in a trajectory based on the time interval and distance between a point and its successor, cf. [25].

In some cases, couriers choose different navigation paths depending on one's particular preference or other random factors instead of seeking the shortest path. To obtain the dominant navigation patterns, we collect courier traces over the past day and adopt ST-DBSCAN [26] for trajectory clustering due to its ability to discover clusters according to spatial and temporal values. We calculate the absolute difference between the courier trace length and the old navigation path length, and use $\lambda_{i,t}$ to represent this difference for edge i at time t.

B. Update Algorithm

We introduce an extended update algorithm to maintain the navigation path cache in OFD, cf. Alg. 3. We obtain the real-time traffic information from couriers at Step 3, which is much more accurate than predicting based on historical data. We introduce $\hat{\mu}_{i,t} = \frac{\lambda_{i,t}}{\max_j \lambda_{j,t}}$ to represent the reward obtained from courier traces. Next, we introduce $\beta_{i,t} = \frac{\tau_{i,t}}{\max_j \tau_{j,t}}$ to record the update delay of edge i where $\tau_{i,t}$ is the edge update delay period. In order to reduce the impact of inaccurate rider data, we adopt $p_{i,t}$ in consideration of both the importance and update delay of the edge, as follows:

$$p_{i,t} \propto \alpha_i \left(\theta\beta_{i,t} + (1-\theta)\hat{\mu}_{i,t}\right), \tag{4}$$



(a) (b) (c) (d)

Fig. 7. Performance of all tested methods in terms of (a) error at time t; (b) mean cumulative error at time t (defined as $\bar{\gamma}_t := \frac{1}{t}\sum_{i=1}^{t}\gamma_i$); (c) mean cumulative error over all update periods (defined as $\bar{\gamma} := \frac{1}{T}\sum_{t=1}^{T}\gamma_t$). (d) illustrates staleness and distance error of paths in the original cache.

where we use $\theta \in (0,1)$ to control the number of backup queries. When the rider's behavior is very accurate, we choose a smaller θ. In contrast, when the traces are noisy or the number of traces is small, we choose a larger one θ. In different parts of the city, we can choose different θ in view of how many couriers operate in that area (θ was selected equal to 0.3 in our experiments).

VII. Experimental Results

In this section, we evaluate the performance of ANTIGONE in terms of path accuracy and query costs.

A. Dataset used for Experiments

We use the navigation paths of a medium-sized city (which covers 6,000+ square kilometers) from historical results queried from a leading map provider along with corresponding courier traces, which is the online cache used at a leading LBS company. The evaluation dataset contains more than 1.5 million navigation paths that comprise of more than 0.7 million GPS points and 1.2 million links. The average length of navigation paths is 4,175.7 meters. We collected more than 5,000 daylong trajectories from more than 1,000 couriers for navigation distance estimation. The histograms of update delay and corresponding mean absolute error (MAE) of the length of navigation paths in the cache are illustrated in Fig. 7(d), which reveals that a large number of paths have not been updated in several months.

B. Evaluation Metrics

Due to the large amounts of data, it is not plausible to re-query all the cached navigation paths to obtain the latest travel times. We thus chose to store 10% of all raw paths in the cache, and acquired the latest navigation path and travel time from a leading map provider as the ground truth. We use the error defined in Eq. 1 as the evaluation metric for accuracy.

C. Benchmarks Used

We compare our update strategy with the following state-of-the-art methods:

- **Delay&Visit**: In each round, query OD pairs with most visits or with update delay exceeding a threshold (threshold=6 hours in our setting); this is the baseline method currently deployed in the OFD platform.
- **PathImpact**: Zhang et al. [2] developed a cache update mechanism by leveraging the *path weight*, as generated using the impact and valid-time of each path (impact is captured with respect to other intersecting paths, and the valid-time is the time period that the path is accurate in the dynamic road network). We compare ANTIGONE with PathImpact using greedily querying OD pairs with the largest path weight as in [2] ($\theta = 0.5, t_{\max} = 12$).
- **Random**: Random selection achieves great performance for many problems in large-scale dynamic graphs [27]. We compare ANTIGONE with the random method that queries OD pairs uniformly at random in each round as a means to update the cache.

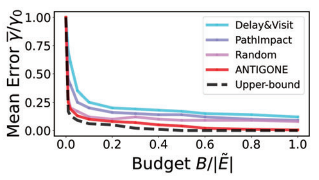
In addition, we compare versus the **Upper-bound** method, in which the probabilities are updated using perfect information and then ANTIGONE is run, as well as the worst case, i.e., **w/o Update**.

D. Comparative Perfomance Validaiton

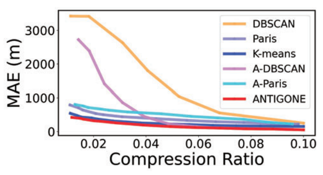
The update duration of a round is one hour and we collect the results for 8 days. Fig. 7(a) gives an illustration of the real-time error of the OD cache. Here, we refer to the model-free method (Alg. 1) as m-ANTIGONE, and to the predictor-based method (Alg. 2) as p-ANTIGONE. Fig. 7(b) plots the mean cumulative error over time, which reveals that model-free ANTIGONE outperforms all baselines. The error of predictor-based ANTIGONE increases at an initial stage and then decreases to achieve the best performance among all methods after 3 days; this is attributed to the lag required to build effective predictors from streaming historical data.

We compare the impact of the query budget, normalized by the number of hybrid edges, i.e., $\frac{B}{|\tilde{E}|}$. Fig. 7(c) illustrates that m-ANTIGONE decreases the mean error by 35.4% and p-ANTIGONE by 18.4% over the selected baselines when $\frac{B}{|\tilde{E}|} = 0.1$. Random selection also achieves formidable performance; a probable reason for that is edges that are common in many paths will be selected at a higher frequency during the sampling process, while other edges are also updated randomly.

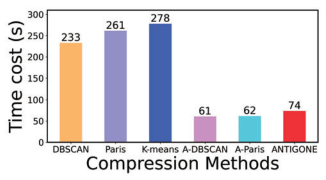
We further compare the performance of ANTIGONE in OFD. Here we refer to the method (Alg. 3) simply as ANTIGONE, We adopt the riding state classification model to segment daylong trajectories of couriers, which achieves 97% precision. Leveraging courier traces to estimate the latest navigation path yields an MSE of distance between our outputs and the navigation paths from API of 49.69m, which underlines the merits of courier data utilization. As shown in Fig. 8(a), the ANTIGONE achieves the best performance among all baselines.



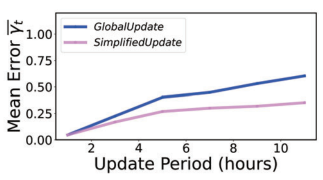
(a) Mean error of cache in OFD



(b) Distance error



(c) Time cost



(d) Simplified Update

Fig. 8. Evaluation of ANTIGONE for OFD (a) and each proposed technical component (b,c,d).

We also study the effectiveness of each proposed technical component of ANTIGONE separately, as follows:

1) Model Construction: We measure the MAE of length of navigation paths in the hybrid graph over variable compression ratio, defined as $CR = \frac{|\tilde{E}|}{|E|}$. We compare the model construction algorithm in ANTIGONE with the DBSCAN based method in [13], K-means [14], and Paris [28]. In ANTIGONE, we obtain a generated road network as described in Sec. IV-A (MAE=57.7m, CR=17.4%). We compare ANTIGONE with different clustering methods for POI hybrid vertex generation, denoted as A(DBSCAN) and A(K-means). As shown in Fig. 8(b), ANTIGONE achieves the lowest error over all other methods. Besides, Fig. 8(c) demonstrates the savings in computing time of the construction framework of ANTIGONE.

In view of the advantage of using Manhattan distance (Fig. 4(a)), we selected a hybrid graph with 8,445 hybrid edges (CR=0.07 and MAE=101.5m). This choice achieved 90% coverage rate of the entrance points of the residential communities that we manually marked. After projecting more than 1.5 million raw paths onto the hybrid graph, we only need to maintain 12,000+ hybrid paths. The MAE of the error of inferring the raw paths from hybrid paths is 6.94m, which accounts for a very small fraction of the average length (=4,175.7m) of cached navigation paths; this demonstrates the efficacy of our graph-based methods.

2) Dynamic Graph Maintenance: We test the performance for longer update period (with fixed budget of 10% of the total edge number). Fig. 8(d) shows a sublinear increase of the mean error of ANTIGONE with the update gap. On the other hand, a larger update period will reduce the instants of re-calculating the shortest paths, thus leading to computational savings. In our online system, the *SimplifiedUpdate* step saves a lot of computational overhead compared with the global update. Our implementation combines ANTIGONE with *SimplifiedUpdate* during the period gap. Fig. 8(d) further shows that *SimplifiedUpdate* reduces the mean error of the cache, which renders *SimplifiedUpdate* the method-of-choice.

VIII. Related Work

Due to the substantive discrepancies associated with navigation path caching versus traditional caching, there are, to the best of our knowledge, no publicly available solutions that could be adopted directly in our scenario. We overview work related to each part of our proposed solutions in what follows.

There are methods for generating the road network [29] and discovering the route path based on trajectory data [10]. Wang et al. [30] proposed a real-time model for estimating the travel time using tensor decomposition based on vehicle trajectories. Zhao et al. [14] proposed a method consisting of Douglas-Peucker (DP)-based compression and density-based clustering to improve the clustering performance of ship trajectory data; this is similar to our method, but we adopted ST-DBSCAN [26] and fast-DTW [31] to reduce the computational burden.

Road network compression [32] and trajectory compression [33] achieve reduction of storage and computational cost. Hendawi et al. [34] proposed a road network simplification algorithm to improve the accuracy of performing the map-matching operation on the produced simplified road network graph. Ali et al. [35] utilized a hybrid aggregation and compression technique and integrated it with the query processing pipeline. For the clustering algorithm in graph compression, we also compare our work with the hierarchical graph clustering algorithm named Paris proposed in [28], which is parameter-free and automatically determines the number of clusters based on the hierarchy structure.

The update strategy of traditional storage limited cache focuses on maximizing the use of space and reducing the miss rate [6]. Thomsen et al. [7] proposed an effective shortest path caching scheme for web search that estimates the benefit of caching a specific shortest path and employs a greedy algorithm for placing most beneficial paths in the cache. Zhang et al. [36] studied the advantages of sharing the direction route information among navigation paths and proposed a greedy algorithm to find the best set of waypoints. Another related technique was proposed in [4], which studied how to reduce the response time of queries while offering accurate query results for range queries and K-nearest neighbors queries (but not our chief objective, i.e., shortest path queries). However, as mentioned in the prequel, the small cache size used does not suffice to meet the stringent real-time requirements pertaining to large-scale LBS platforms.

Acknowledgments

Xiang-Yang Li is the corresponding author. The research is supported by National Key R&D Program of China 2018YFB0803400, China National Funds for Distinguished Young Scientists with No.61625205, China National Natural Science Foundation with No. 61822209, No. 62132018, No. 61932016, Key Research Program of Frontier Sciences, CAS. No. QYZDY-SSW-JSC002.

References

[1] Z. Wang, K. Fu, and J. Ye, "Learning to estimate the travel time," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 858–866.

[2] D. Zhang, A. Liu, Z. Li, G. Jia, F. Chen, and Q. Li, "Effective shortest travel-time path caching and estimating for location-based services," *World Wide Web*, vol. 22, no. 2, pp. 455–475, 2019.

[3] D. Zhang, C.-Y. Chow, Q. Li, X. Zhang, and Y. Xu, "SMashQ: Spatial mashup framework for k-NN queries in time-dependent road networks," *Distributed and Parallel Databases*, vol. 31, no. 2, pp. 259–287, 2013.

[4] Y. Li and M. L. Yiu, "Route-Saver: leveraging route APIs for accurate and efficient query processing at location-based services," *Transactions on Knowledge and Data Engineering*, vol. 27, no. 1, pp. 235–249, 2014.

[5] Y. Zhang, Y.-L. Hsueh, W.-C. Lee, and Y.-H. Jhang, "Efficient cache-supported path planning on roads," *Transactions on Knowledge and Data Engineering*, vol. 28, no. 4, pp. 951–964, 2015.

[6] N. Beckmann, H. Chen, and A. Cidon, "LHD: Improving cache hit rate by maximizing hit density," in *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018, pp. 389–403.

[7] J. R. Thomsen, M. L. Yiu, and C. S. Jensen, "Effective caching of shortest paths for location-based services," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2012, pp. 313–324.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[9] A. Sadri, F. D. Salim, Y. Ren, M. Zameni, J. Chan, and T. Sellis, "Shrink: Distance preserving graph compression," *Information Systems*, vol. 69, pp. 180–193, 2017.

[10] Z. Chen, H. T. Shen, and X. Zhou, "Discovering popular routes from trajectories," in *27th International Conference on Data Engineering*. IEEE, 2011, pp. 900–911.

[11] L. Cao and J. Krumm, "From GPS traces to a routable road map," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Sdvances in Geographic Information Systems*, 2009, pp. 3–12.

[12] J. J. Davies, A. R. Beresford, and A. Hopper, "Scalable, distributed, real-time map generation," *IEEE Pervasive Computing*, vol. 5, no. 4, pp. 47–54, 2006.

[13] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "DBSCAN revisited, revisited: why and how you should (still) use DBSCAN," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

[14] X. Shen, W. Liu, I. Tsang, F. Shen, and Q.-S. Sun, "Compressed k-means for large-scale clustering," in *Thirty-first AAAI conference on artificial intelligence*, 2017, pp. 2527–2533.

[15] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

[16] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, "An analysis of approximations for maximizing submodular set functionsII," in *Polyhedral Combinatorics*. Springer, 1978, pp. 73–87.

[17] J. Van Lint and H. J. van Zuylen, "Monitoring and predicting freeway travel time reliability: Using width and skew of day-to-day travel time distribution," *Transportation Research Record*, vol. 1917, no. 1, pp. 54–62, 2005.

[18] L. Tran-Thanh, A. Chapman, A. Rogers, and N. Jennings, "Knapsack based optimal policies for budget–limited multi–armed bandits," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 1, 2012, pp. 1134–1140.

[19] G. Gao, J. Wu, M. Xiao, and G. Chen, "Combinatorial multi-armed bandit based unknown worker recruitment in heterogeneous crowdsensing," in *IEEE Conference on Computer Communications (INFOCOM)*, 2020, pp. 179–188.

[20] S. E. Jabari, N. M. Freris, and D. M. Dilip, "Sparse travel time estimation from streaming data," *Transportation Science*, vol. 54, no. 1, pp. 1–20, 2020.

[21] Y. Duan, L. Yisheng, and F.-Y. Wang, "Travel time prediction with LSTM neural network," in *19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2016, pp. 1053–1058.

[22] J. Gao, Q. Zhao, W. Ren, A. Swami, R. Ramanathan, and A. Bar-Noy, "Dynamic shortest path algorithms for hypergraphs," *Transactions on Networking*, vol. 23, no. 6, pp. 1805–1817, 2014.

[23] L. Zhu, W. Yu, K. Zhou, X. Wang, W. Feng, P. Wang, N. Chen, and P. Lee, "Order fulfillment cycle time estimation for on-demand food delivery," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2571–2580.

[24] O. Steven Eyobu and D. S. Han, "Feature representation and data augmentation for human activity classification based on wearable IMU sensor data using a deep LSTM neural network," *Sensors*, vol. 18, no. 9, p. 2892, 2018.

[25] Y. Zheng, "Trajectory data mining: an overview," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, no. 3, pp. 1–41, 2015.

[26] D. Birant and A. Kut, "ST-DBSCAN: An algorithm for clustering spatial–temporal data," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.

[27] D. Kostic, A. Rodriguez, J. R. Albrecht, A. Bhirud, and A. Vahdat, "Using random subsets to build scalable network services." in *USENIX Symposium on Internet Technologies and Systems*, 2003, p. 19.

[28] T. Bonald, B. Charpentier, A. Galland, and A. Hollocou, "Hierarchical graph clustering using node pair sampling," in *14th International Workshop on Mining and Learning with Graphs (MLG)*, 2018.

[29] Y. Zhang, J. Liu, X. Qian, A. Qiu, and F. Zhang, "An automatic road network construction method using massive GPS trajectory data," *ISPRS International Journal of Geo-Information*, vol. 6, no. 12, p. 400, 2017.

[30] Y. Wang, Y. Zheng, and Y. Xue, "Travel time estimation of a path using sparse trajectories," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 25–34.

[31] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[32] A. M. Hendawi, A. Khot, A. Rustum, A. Basalamah, A. Teredesai, and M. Ali, "COMA: Road network compression for map-matching," in *Proceedings of the 16th IEEE International Conference on Mobile Data Management*, vol. 1, 2015, pp. 104–109.

[33] R. Song, W. Sun, B. Zheng, and Y. Zheng, "PRESS: A novel framework of trajectory compression in road networks," *Proceedings of the VLDB Endowment*, vol. 7, no. 9, pp. 661–672, 2014.

[34] A. Hendawi, J. A. Stankovic, A. Taha, S. El-Sappagh, A. A. Ahmadain, and M. Ali, "Road network simplification for location-based services," *GeoInformatica*, vol. 24, no. 4, pp. 801–826, 2020.

[35] A. Khoshgozaran, A. Khodaei, M. Sharifzadeh, and C. Shahabi, "A hybrid aggregation and compression technique for road network databases," *Knowledge and Information Systems*, vol. 17, no. 3, pp. 265–286, 2008.

[36] D. Zhang, C.-Y. Chow, A. Liu, X. Zhang, Q. Ding, and Q. Li, "Efficient evaluation of shortest travel-time path queries through spatial mashups," *GeoInformatica*, vol. 22, no. 1, pp. 3–28, 2018.