IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 8, 2018, accepted June 12, 2018, date of publication July 4, 2018, date of current version July 30, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2852787

T-Reduce: Route-Aware Mobile Trajectory Data Reduction in Transportation Management Systems

NAMGU KWON[1], JIBUM KIM[1], SUNHO LIM[2], (Senior Member, IEEE), JINSEOK CHAE[1], (Member, IEEE), AND HEEMIN PARK[3], (Member, IEEE)

[1]Department of Computer Science and Engineering, Incheon National University, Incheon 22012, South Korea
[2]T[2]WISTOR: TTU Wireless Mobile Networking Laboratory, Department of Computer Science, Texas Tech University, Lubbock, TX 79409, USA
[3]Department of Software, Sangmyung University, Cheonan 31066, South Korea

Corresponding author: Jinseok Chae (jschae@inu.ac.kr)

This work was supported by a Incheon National University Research Grant in 2014.

ABSTRACT Due to the increasingly popular e-commerce and rapidly growing global market of delivery services, transportation management system (TMS) has been playing a critical role in minimizing transportation delay and cost as well as improving reliability. However, how to efficiently keep track of running vehicles and judiciously store their corresponding trajectories become a key issue in the presence of scarce wireless bandwidth and limited storage space. In this paper, we propose a novel mobile trajectory reduction scheme, called *T-Reduce*, to reduce the size of trajectory data stored in a TMS server. We first develop a path refinement operation to adjust raw location update data transmitted from vehicles to a given road topology as close as possible to improve their location accuracy. Second, we develop a route matching operation consisting of three sub-operations, route extraction, route recognition, and trajectory generation, for the server to identify and extract a set of routes by decomposing the received location update data. Then, the server can store each trajectory as a set of corresponding route *ids* instead of storing entire location update data. We set up a small-scale testbed, implement the proposed scheme with a real-world trace data collected from a logistic company, and conduct extensive experiments for performance evaluation and comparison. The experimental results show that the proposed scheme can significantly reduce the average trajectory error and route information up to 26.4% and 88%, respectively, compared to that of two prior trajectory-based and corner-based approaches. The proposed approach can also achieve up to 5.72 times cost efficiency compared to the prior approaches.

INDEX TERMS Location update, mobile trajectory, route information, transportation management system.

I. INTRODUCTION

A transportation management system (TMS) is designed to support efficient delivery of logistics from source to destination. The primary goals of TMS are to plan and optimize terrestrial transportation routes and track vehicles in a real-time manner to minimize transportation delay and effective cost but improve transportation reliability. Thanks to increasingly popular e-commerce, the global market of delivery services is rapidly growing more than 24.2 trillion in 2015 which is 8.4% increase compared to 2013, and 8.5% annual increase is expected by 2018 [1]. The number of vehicle registrations for terrestrial transportation is increased from 2.5 to 3.4 million at 2000 and 2015, respectively in South Korea [2]. Due to the explosive demands of logistics information in the presence of limited road infrastructure, it is envisioned that the TMS plays a critical role in assisting the collaboration among shippers, carriers, and customers as well as contributing economy growth.

One of key functionalities in operating the TMS is to keep track of running vehicles (later in short, vehicles). Each vehicle periodically (e.g., every few seconds) updates its current location using a customized device or smartphone equipped with an on-board global positioning system (GPS) and sends the location to a TMS server (later in short, server) wirelessly through a 3G/4G network or road side unit (RSU) [3]. Then the server builds the trajectories of vehicles based on the

series of location update data, and plans a set of alternate routes in case of time-varying traffic, changes of delivery destination, or any road accident and construction. The server can store all the received location update data for the purpose of recording delivery transactions, optimizing transportation routes, and reducing the number of delivery rounds. As the number of vehicles increases, however, frequent transmissions of location update data may waste scarce wireless bandwidth and incur a non-negligible storage cost at the server. In addition, if a vehicle needs to share or advertise its current location in a real-time fashion, it often continuously transmits its location data in a form of streaming data but this requires sufficient wireless bandwidth and ample storage space. Several transportation apps for smartphone broadcast the location data to near-by customers for sharing, e.g., Gettaxi [4], Lyft [5], or Sidecar [6]. Thus, the volume of location data streaming from multiple vehicles to the server could quickly be huge.

To address these issues, off-line approaches [7]–[10] have been proposed but they primarily focus on the reconstructing trajectories with reduced errors. On-line approaches without knowledge on the trajectories [11]–[15] also have been proposed based on the time-, distance-, or prediction-based location update. But they may suffer from redundant location updates and trajectory estimation errors because of the time-varying speeds of vehicles and inherent road topologies. In fact, it is not trivial to obtain the knowledge of trajectories during the run time [16]–[18] in practice. In light of these, we propose a mobile trajectory data reduction scheme, called *T-Reduce*, not only to judiciously decrease the number of location update data transmitted from multiple vehicles but to significantly reduce the volume of trajectories stored at the server. Our contribution is briefly summarized in three-fold:

- We first develop a path refinement operation and convert raw location update data into usable data in the TMS. The quality and accuracy of location update data are improved by removing redundant or deviated from road, adjust to road, and increase path density.
- Second, we develop a route matching operation including three sub-operations: route extraction, route recognition, and trajectory generation. We deploy a 2-dimensional tree to efficiently access the routes and quickly determine whether newly identified routes and existing routes are matched. We manage each trajectory as a set of smaller path segments and store its route *ids* instead, rather than storing entire location data involved in the trajectory.
- Third, we setup a small-scale testbed and implement the proposed approach using a real-world dataset collected from a logistic company. We also implement the conventional approaches, trajectory-based (TB) and corner-based (CB), for performance evaluation and comparison.

We conduct extensive experiments for performance evaluation and analysis in terms of average trajectory error, efficiency in route recognition, accumulated file size, and cost efficiency. The experiment results show that the proposed method is able to significantly reduce the storage for trajectories by minimizing the number of location update data transmitted from vehicles to the server compared to the prior schemes. Moreover, the original path can be successfully reconstructed with smaller errors than that of prior schemes. The remainder of paper is organized as follows. The prior approaches are summarized and analyzed in Section II. The proposed system model and trajectory data reduction method are presented in Sections III and IV, respectively. Section V is devoted to extensive experiments, performance comparison and analysis, and discussion of potential implementation issues. Finally, we conclude the paper in Section VI.

II. RELATED WORK

There are two major approaches to reduce the amount of storage space at the server: on-line and off-line. We can further divide the on-line approach based on whether prior knowledge on the trajectories or area (e.g., map, road networks, etc.) is utilized.

A. OFF-LINE APPROACHES

Off-line approaches use all the location information of a trajectory and pursue a globally optimized solution, but they may suffer from the communication and storage cost in recording location changes for entire trajectories. In [7], the first off-line approach (DP) is proposed for a simplified trajectory with the reduced number of locations. Given a trajectory composed of multiple line segments, the DP finds the furthest located location from the approximating line segment that connects start and end locations of the segment. By comparing the distance between the furthest location and the approximating line segment with a tolerance threshold, the DP recursively reduces the number of locations in the trajectory. In addition to the DP approach, Qian and Lu [8] try to preserve the spatio-temporal properties of the trajectories. In this approach, they succeeded in preserving the spatio-temporal information of the trajectories, but the compression ratio of the locations is worse than that of the DP.

Since all the location information of a trajectory is available in the off-line approach, Imai and Iri [9] propose an optimal algorithm that minimizes the number of locations of the approximating trajectory when an error bound is given, or minimizes the approximation error when the number of locations of the approximating trajectory is constrained. The algorithm first constructs a directed acyclic graph, where a shortest path is sought. Although the algorithm can guarantee an optimal solution, it requires global location information and may incur high computational overhead for long trajectories. In order to reduce errors and noises introduced while acquiring GPS data from the vehicles, wavelet analysis has been adopted to filter errors and reconstruct trajectory data at the server side [10]. This approach does not focus on reducing storage size but only on reconstructing trajectories with less errors.

B. ON-LINE APPROACHES WITHOUT KNOWLEDGE ON THE TRAJECTORIES

Unlike off-line approaches, on-line approaches selectively send location update to a server, where the trajectory approximation is processed in a real-time manner and thus, the communication and storage cost can be saved. They further consider whether a vehicle is aware of past trajectories.

If a vehicle has no knowledge of the past trajectories, it periodically updates and sends its time-varying location information to the server, called time-based location update [12]. In the distance-based location update [11], [12], however, each vehicle updates its location information based on the distance between the current and the last locations where the location updates have been sent to the server. Although the distance-based approach generally shows good performance, it may incur redundant location updates to the server if the original trajectory is simple, such as a straight line. Prediction-based location update approaches are also proposed by observing vehicle's moving direction [13], [14]. The linear dead reckoning is applied to predict the trajectory of moving vehicles by on-line, but it requires to deploy a complex prediction model. Park *et al.* [15] propose an on-line trajectory reduction method by sending location update to the sever only when the current approximating trajectory exceeds a predefined trajectory error. This approach shows good performance in terms of the number of locations, but it may choose sample locations that are not located at roadway corners or intersections as long as the error of approximating trajectory does not exceed the given error bound.

C. ON-LINE APPROACHES WITH KNOWLEDGE ON THE TRAJECTORIES

If a vehicle is aware of the area or past trajectories, there are more chances to reduce trajectory data using approximation. In [16], a Map-Matching algorithm is developed by using historical positioning information to decide whether vehicle's precise location is within the actual roads. But this approach does not consider how to reduce the communication cost nor how to exploit the route history. Kim *et al.* [17] use map data, extract corner points located at the intersections using the Harris corner detection algorithm [19], and improve the accuracy of trajectory, compared to other on-line approaches. Zheng *et al.* [18] deploy a road network to better approximate trajectories from low-sampling-rate trajectories. It could be possible to reduce trajectory data based on the prior knowledge on the trajectories. But it is not trivial (unless it is impossible) to obtain the knowledge in a systematic way. Thus, the server may need to pre-process and store all the information of area of interests before conducting the reduction operation.

In summary, a great deal of research effort has been devoted to minimize the communication and storage cost but maximize the accuracy of trajectories. However, little attention has been paid in how to efficiently process a series of raw trajectory data streamed from multiple vehicles, extract route data, reduce trajectory data, and resolve any practical implementation issue in the realm of TMSs.

III. SYSTEM MODEL

Each vehicle equips a built-in navigation system integrated with a GPS or a smartphone running an app and executes computing and communication operations. Since each vehicle is powered by its own built-in battery, energy conservation does not become an issue. Each vehicle periodically updates the current location and communicates with a server through a 3G/4G network or a RSU. The vehicle movement is restricted by underlying fixed roads with speed limits and traffic lights. Here, the RSU can be mounted on the top of a signal light, a road lamp, a gas station, or an intersection, and it is connected with a wired network and operates as a gateway for vehicles to connect to the Internet [3].

When a vehicle updates its current location k, it records a point, $p_k =< (x_k, y_k), t_k >$, where both x_k and y_k are the coordinates of a point and t_k is the update time. A *path* i, pt_i, is defined as a sequence of points, $pt_i = \{p_k\}$, where p_k is any updated point between source and destination. The point of source (s) and destination (d) can be denoted as p_s and p_d, respectively. Given a path, we construct an undirected connected graph, $g(v, e)$, where v and e are an updated point (e.g., p_k) and a road, respectively. We also use a directed graph only when it is necessary to clearly show the direction of vehicles. In this paper, we implicitly assume a two-way road topology and do not consider an one-way for the sake of simplicity. The connected graph can be decomposed into a set of biconnected components, in which each biconnected component is a maximal biconnected subgraph. A *route* is also defined as a biconnected component in the graph. A route i, r_i, is a subset of path and consists of at least more than two points, and it can be expressed as, $r_i = \{p_k| < (x_k, y_k), t_k >\}$. Here, p_k is any updated point between p_s and p_d. In addition, a *trajectory* is defined as a set of routes, where the vehicle follows the routes. A trajectory j, tr_j, can be expressed as, $tr_j = \{r_i\}$, where r_i is any route formed the path.

For example, an undirected connected graph consists of 13 points and five routes as shown in Fig. 1. A set of points, S, included in the routes can be expressed as, $S = \{p_k\}$, where $k = 1$ to 13. Five routes can be $r_1 = \{p_1, p_2, p_3\}$, $r_2 = \{p_3, p_4, p_5, p_6\}$, $r_3 = \{p_6, p_7, p_8\}$, $r_4 = \{p_9, p_{10}, p_3\}$,

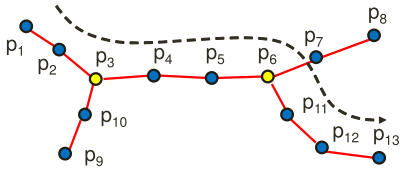


FIGURE 1. An example of path, where vehicles periodically update their points from p_1 to p_{13}. One vehicle moves from a source (e.g., p_1) to a destination (e.g., p_{13}) following a dashed arrow. Here, both p_3 and p_6 are articulation points.

and $r_5 = \{p_6, p_{11}, p_{12}, p_{13}\}$. In the graph, an articulation point can disconnect the graph into multiple disconnected routes, if each point is removed. A set of articulation points, S_a, is $S_a = \{p_3, p_6\}$. If the vehicle moves from p_1 to p_{13} following a path, $pt = \{p_1, p_2, p_3, p_4, p_5, p_6, p_{11}, p_{12}, p_{13}\}$, its trajectory is $tr = \{r_1, r_2, r_5\}$.

IV. MOBILE TRAJECTORY DATA REDUCTION

As the number of vehicles increases, more vehicles will frequently transmit their location update data to the server. Since each route has at least two points, a stream of location update data can immediately produce a huge number of routes and their corresponding trajectories. Thus, the cost of storing and updating routes and trajectories will quickly increase at the server. In this paper, we propose a mobile trajectory data reduction scheme, called T-Reduce, to reduce the cost of storing and maintaining the trajectories by minimizing the number of location update data transmitted from vehicles to the server and updating the trajectories stored at the server. The basic idea is that each vehicle pre-loads the most updated route information, checks whether it follows one of the existing routes, and transmits its current location to the server only when it is deviated enough from the route. Then the number of location update data transmitted from vehicles can be decreased, leading to overall trajectory data reduction at the server. The server also periodically updates the trajectories in an off-line manner based on the received location update data to further reduce them. The more trajectories are updated, the less number of location update data is transmitted.

The T-Reduce consists of two major operations: path refinement and route matching. The route matching operation is further divided into three sub-operations: route extraction, route recognition, and trajectory generation. The first operation, path refinement, is to pre-process and refined paths before performing the route matching operation. The path refinement is required because the raw location update data transmitted from vehicles includes multiple GPS errors and duplicated location update data to be corrected. The second operation, route matching, is to determine whether a new route belongs to any existing route stored in the route database. If it is the case, a trajectory is generated based on the new route *id* using the trajectory generation operation. Otherwise, we extract a new route using both route extraction and recognition operations and save them into the route database. An overview of the proposed approach is depicted in Fig. 2.

A. PATH REFINEMENT

We first analyze a set of paths built based on the location update data stored at the server. In this paper, we use a real trace of location update data collected from vehicles during a month (April 2016) and maintained by a logistics company[1] (Incheon, South Korea). Since the trace is not originally designed for the TMSs, we need to refine it by removing the points that do not show any path. The basic idea is to check whether any consecutive point is located within a given road topology considering a reasonable error bound. For example, two consecutive points are significantly far away from each other and even not located within a road as shown in Subfig. 3(a), resulting in an invalid path. However, a set of valid paths is observed in Subfig. 3(b), where the location update data follow the road topology. In our experiment, we can extract 249 valid paths out of original 553 paths by applying the path refinement operation.

Second, we also refine the valid paths by removing redundant or significantly deviated locations from the road. For example, an original path shown in Subfig. 4(a) is corrected by removing the points deviated significantly from the road in Subfig. 4(b). We further adjust the points close to the road and even virtually add more points along the path, if two consecutive points are far away from each other in Subfigs. 4(c) and (d), respectively.

[1] Due to the request of anonymity for privacy, we do not reveal the company name in this paper.

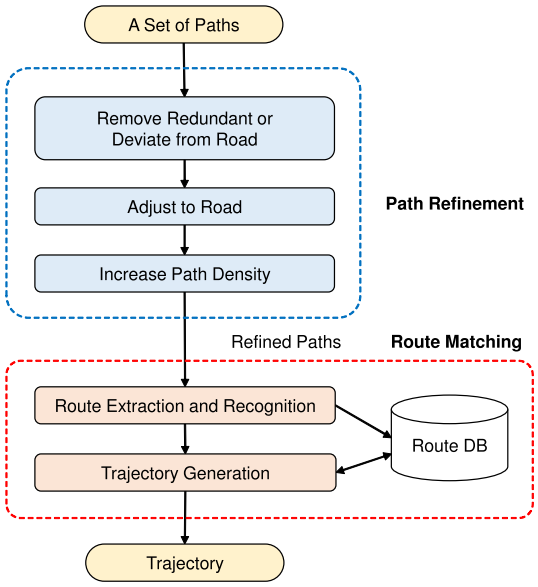


FIGURE 2. An overview of the proposed approach, consisting of two major operations: path refinement and route matching. The route matching operation is further divided into three sub-operations: route extraction, route recognition, and trajectory generation.

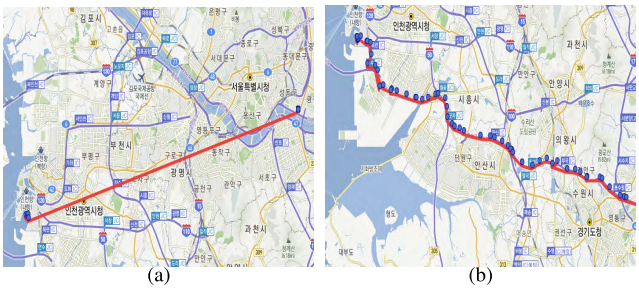


FIGURE 3. A snapshot of location update data collected at a server. Here, a blue dot represents a point where a vehicle updates and transmits its current location to the server. A thick red line represents a path where the vehicle moves. (a) Invalid Path. (b) Valid Path.

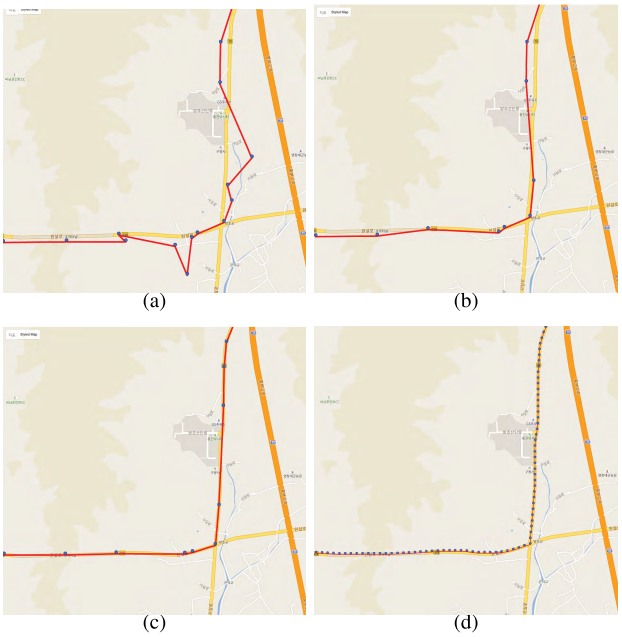


FIGURE 4. An original path (Subfig. 4(a)) is corrected by removing the location update data that are redundant or significantly deviated from the road (Subfig. 4(b)), adjusting the path close to the road (Subfig. 4(c)), and increasing the path density (Subfig. 4(d)). (a) Original path. (b) Refined path after removing redundant or significantly deviated from the road. (c) Refined path after adjusting the location update data to the road. (d) Refined path after increasing path density.

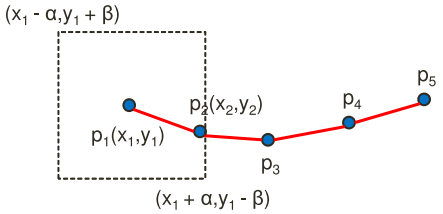


FIGURE 5. A two-dimensional range search, where each point is searched from its prior location. Here, five points, p_1 to p_5, are located in a given path.

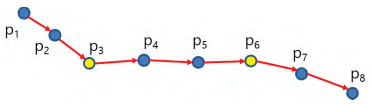


FIGURE 6. An example of path, where two articulation points marked as a yellow circle are included, p_3 and p_6.

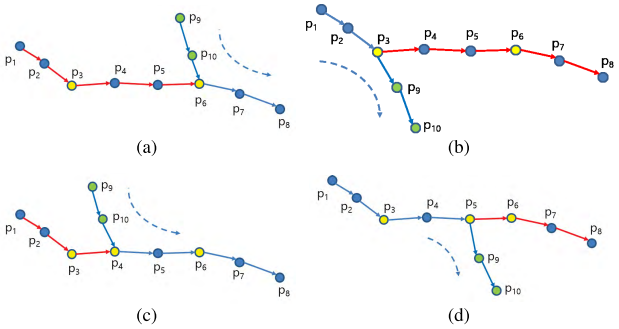


FIGURE 7. A set of cases, where vehicles enter into and exit from the route via an articulation or non-articulation point. Here, a vehicle moves following a dashed arrow. (a) Case 1. (b) Case 2. (c) Case 3. (d) Case 4.

B. ROUTE MATCHING

The route matching operation consists of three sub-operations: route extraction, route recognition, and trajectory generation.

1) ROUTE EXTRACTION

In this operation, a set of routes is extracted from the refined paths. Each route is a subset of paths and has at least more than two location update data. Since a route can be decomposed by an articulation point, any shared route can be divided into a set of smaller routes and thus, multiple number of new routes can be identified.

To realize this, we first check whether two consecutive points are closely located in a path and deploy a simple two-dimensional range search [20], [21] with system parameters, α and β. Note that both α and β may vary depending on the distance of two consecutive points in the path. For example, if the location update is conducted every minute, both α and β can be set to 1 (km) because the average moving distance of vehicles is approximately 886 (m) based on the analysis of location update data in this paper. In case of the location update every 30 seconds, both α and β can sufficiently be set to 500 (m).

Suppose there is a path including five location update data, p_1 to p_5, and both p_1 and p_2 are updated at (x_1, y_1) and (x_2, y_2), respectively in Fig. 5. Based on the point of p_1, the p_2 is searched whether it is located with a range, which is represented as a square area with two diagonal points located at $(x_3 - \alpha, y_3 + \beta)$ and $(x_3 + \alpha, y_3 - \beta)$, respectively. If the p_2 is found, it is considered to be located at the same path. Similarly, the points of p_3 to p_5 are searched based on their prior location. Thus, all five location update data are categorized into the same route, $r = \{p_k\}$, where $k = 1$ to 5.

Second, we divide a route shared by multiple paths into a set of smaller routes. The rationale behind this approach is to maintain each route as short as possible to efficiently manage the paths. For example, a path, pt_1, consists of three routes, $r_1 = \{p_1, p_2, p_3\}$, $r_2 = \{p_3, p_4, p_5, p_6\}$, and $r_3 = \{p_6, p_7, p_8\}$ in Fig. 6. There are two articulation points, $S_a = \{p_3, p_6\}$. Based on the pt_1, we can analyze four cases depending on where a vehicle enters into and exits from the path in Fig. 7. Subfig. 7(a) shows a case when the vehicle enters the path via the articulation point, p_6. In this case, a new path $pt_2 = \{p_9, p_{10}, p_6, p_7, p_8\}$ can be generated and a subset (i.e., path segment) of pt_1 where the vehicle moves can be a new route, $r_4 = \{p_9, p_{10}, p_6\}$. Subfig. 7(b) depicts a case when the vehicle exits the path via the articulation point, p_3. Similarly, a new path $pt_2 = \{p_1, p_2, p_3, p_9, p_{10}\}$ can be generated and a new route is identified, $r_4 = \{p_3, p_9, p_{10}\}$. In Subfig. 7(c), unlike prior two cases, the vehicle enters the path via a non-articulation point, p_4. Then we generate a new path

$pt_2 = \{p_9, p_{10}, p_4, p_5, p_6, p_7, p_8\}$ and split the existing route r_2 into two separate routes, $r_4 = \{p_3, p_4\}$ and $r_5 = \{p_4, p_5, p_6\}$. A new route $r_6 = \{p_9, p_{10}, p_4\}$ can also be generated. Here, p_4 becomes a new articulation point, $S_a = \{p_3, p_4, p_6\}$. The vehicle exits the path via a non-articulation point, p_5 in Subfig. 7(d). Similarly we generate a new path $pt_2 = \{p_1, p_2, p_3, p_4, p_5, p_9, p_{10}\}$ and split the existing route r_2 into two separate routes, $r_4 = \{p_3, p_4, p_5\}$ and $r_5 = \{p_5, p_6\}$. A new route $r_6 = \{p_5, p_9, p_{10}\}$ can also be generated. Here, p_5 becomes a new articulation point, $S_a = \{p_3, p_5, p_6\}$.

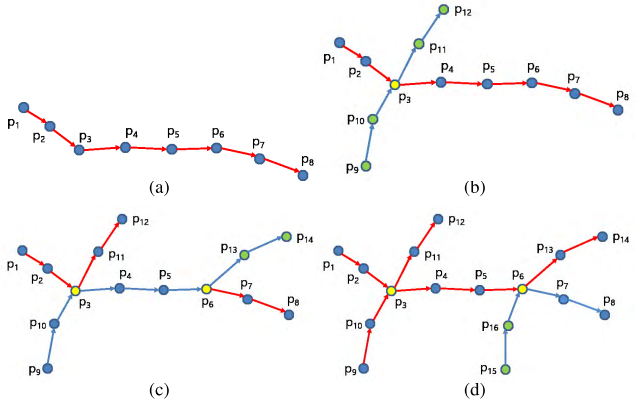


FIGURE 8. An example of identifying a new route, where an original single route can be extended to multiple routes depending on the mobility of vehicles. (a) An initial path. (b) Cases 3 and 2 (Subfigs. 7(c) and (b)) applied. (c) Case 4 (Subfig. 7(d)) applied. (d) Case 1 (Subfig. 7(a)) applied.

Third, we investigate how to identify a new route based on the aforementioned cases in Fig. 8. A simple path containing a single route is given in Subfig. 8(a), where $S = pt_1 = r_1 = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$ and $S_a = \emptyset$. In Subfig. 8(b), when a vehicle moves following a new path $pt_2 = \{p_9, p_{10}, p_3, p_{11}, p_{12}\}$, a new route $r_2 = \{p_9, p_{10}, p_3\}$ can be generated based on the case 3 (Subfig. 7(c)). Since the p_3 becomes an articulation point, $S_a = \{p_3\}$, we can split the existing route r_1 into two separate routes, $r_3 = \{p_1, p_2, p_3\}$ and $r_4 = \{p_3, p_4, p_5, p_6, p_7, p_8\}$, and delete the r_1. Then another new route $r_5 = \{p_3, p_{11}, p_{12}\}$ can be generated based on the case 2 (Subfig. 7(b)). As a result, four additional routes can be identified, r_2, r_3, r_4, and r_5. When the vehicle moves following a new path $pt_3 = \{p_9, p_{10}, p_3, p_4, p_5, p_6, p_{13}, p_{14}\}$ in Subfig. 8(c), the path already includes the existing route, r_2. According to the subset of path, $\{p_3, p_4, p_5, p_6, p_{13}, p_{14}\}$, we can generate a new route $r_6 = \{p_3, p_4, p_5, p_6\}$. Since the p_6 becomes an articulation point, $S_a = \{p_3, p_6\}$, we can split the exiting route r_4 into two separate routes, $r_6 = \{p_3, p_4, p_5, p_6\}$ and $r_7 = \{p_6, p_7, p_8\}$, and delete the r_4. Then another new route $r_8 = \{p_6, p_{13}, p_{14}\}$ can be generated based on the case 4 (Subfig. 7(d)). Thus, three more routes can be identified, r_6, r_7, and r_8. Subfig. 8(d) depicts when the vehicle moves following a new path $pt_4 = \{p_{15}, p_{16}, p_6, p_7, p_8\}$. In the path, a new route can be generated, $r_9 = \{p_{15}, p_{16}, p_6\}$, based on the case 1 (Subfig. 7(a)). Finally, the original path with a single route has been extended to eight routes, r_2 to r_9. Here, $S = \{p_k\}$, where k is 1 to 16, and $S_a = \{p_3, p_6\}$. The major operations of route extraction are summarized in Fig. 9.

Notations:
- p_s, p_e: A starting and ending point of a path.
- S, S_a: A set of points on routes and set of articulation points.
- r: A route.
- l: A last route id.
- $ptSgmt$: A path segment.
- isR: A boolean variable to store whether the previous point is on a route or not. An initial value is set to FALSE.

◇ Traverse $\forall\ pt.p_i$ from the starting point ($pt.p_s$) to the ending point ($pt.p_e$),
◇ When the current point is on a route ($pt.p_i \in S$) and is an articulation point ($pt.p_i \in S_a$) and the previous point is not on the route ($isR =$ FALSE),
  $isR \leftarrow$ TRUE;
  Append $pt.p_i$ to a new route r_{l+1};
  $l \leftarrow l+1$;
◇ When the current point is on a route ($pt.p_i \in S$) and is not an articulation point ($pt.p_i \notin S_a$) and is not a starting point ($pt.p_i \neq pt.p_s$) and the previous point is not on the route ($isR =$ FALSE),
  $isR \leftarrow$ TRUE;
  Append $pt.p_i$ to a new route r_{l+1};
  ptSgmt $\leftarrow$ getPathSegment($pt.p_i$);
  $id \leftarrow$ getRouteId($pt.p_i$);
  Identify [ptSgmt.p_s to ptSgmt.p_i] as a new route r_{l+2};
  Identify [ptSgmt.p_i to ptSgmt.p_e] as a new route r_{l+3};
  $l \leftarrow l+3$;
  Delete a route r_{id};
  $S_a \leftarrow S_a \cup \{pt.p_i\}$; /* add a new articulation point */
◇ When the current point is not on the route ($pt.p_i \notin S$) and the previous point is on the route ($isR =$ TRUE),
  $isR \leftarrow$ FALSE;
  if $pt.p_i \neq pt.p_s$ **and** $pt.p_{i-1} \in S_a$
    Append $pt.p_{i-1}$ to a new route r_{l+1};
◇ When the current point is not on the route ($pt.p_i \notin S$) and the previous point is not on the route ($isR =$ FALSE),
  $S_a \leftarrow S_a \cup \{pt.p_{i-1}\}$; /* add a new articulation point */
  ptSgmt $\leftarrow$ getPathSegment($pt.p_{i-1}$);
  $id \leftarrow$ getRouteId($pt.p_i$);
  Identify [ptSgmt.p_s to ptSgmt.p_{i-1}] as a new route r_{l+1};
  Identify [ptSgmt.p_{i-1} to ptSgmt.p_e] as a new route r_{l+2};
  $l \leftarrow l+2$;
  Delete a route r_{id};
  Append $pt.p_{i-1}$ to a new route r_{l+1};
◇ Otherwise,
  Append $pt.p_i$ to a new route r_{l+1};
  $S \leftarrow S \cup \{pt.p_i\}$; /* add a new point */

FIGURE 9. The pseudo code of route extraction.

2) ROUTE RECOGNITION

Due to an unexpected road accident and construction, a vehicle may not follow the route what it was originally planned according to the pre-loaded route. In this operation, each vehicle monitors whether it follows the current route and conducts the corresponding operation for location update. Thus, a new route can be generated and the original trajectory stored at the server can be changed.

First, there is a path containing a single route as shown in Subfig. 10(a), where $r = \{p_k\}$, where $k = 1$ to 8 and the coordinates of p_1 and p_2 are (x_1, y_1) and (x_2, y_2), respectively. Suppose a vehicle periodically updates its point at $p'_{k'}$, where $k' = 1$ to 5. Upon the vehicle updates its point at p'_1 (e.g., (x'_1, y'_1)), it checks whether it moves following any existing route, r. The vehicle creates a virtual square area with two diagonal points located at $(x'_1 - \alpha, y'_1 + \beta)$ and $(x'_1 + \alpha, y'_1 - \beta)$ with system parameters, α and β. Then the

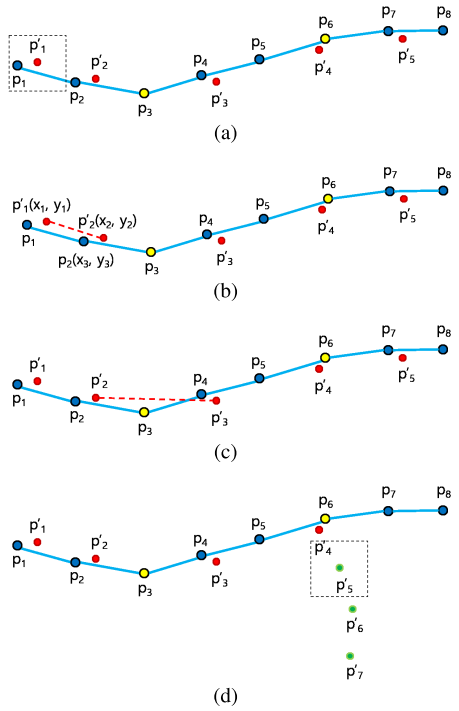


FIGURE 10. A path of vehicle follows the routes.

vehicle executes the two-dimensional range search to see if p_1 and p_2 are located within the area. If either p_1 or p_2 is not found within the area, the vehicle may not move following the r correctly and transmit its current location to the server for update, e.g., p'_1 ($< (x'_1, y'_1), t_{cur} >$). If a point like p'_1 or p'_2 is on a route and is not an articulation point, then the trajectory generation module is executed. Second, suppose the vehicle moves along a new route and updates its point at p'_k, where $k = 5$ to 7 in Subfig. 10(d). Since the pre-loaded route does not contain p'_6, the vehicle transmits its location to the server for update. In this case, the vehicle transmits the pre-loaded route id containing p'_1 to p'_4 and new points, including p'_5 to p'_7.

3) TRAJECTORY GENERATION

In this operation, the server first decomposes a path into a set of path segments based on the received new location update data transmitted from vehicles by using the route extraction sub-operation. Here, a subset of path is a path segment. If a path segment is not found in the route database, it is added to the route database with a new route id. The server also periodically updates the existing routes and extracts new routes in an off-line manner. The vehicle checks whether it surely follows the r by measuring the distance (d) between p'_1 and a line connecting p_1 and p_2 as shown in Subfig 10(b). The distance can be easily calculated by using $d = \frac{ax'_1 - y'_1 - ax_1 + y_1}{\sqrt{a^2+1}}$, where a is a slope, $\frac{y_2 - y_1}{x_2 - x_1}$. If the distance is $d < \epsilon$, p'_1 is considered to be located on the line and thus, the vehicle most likely follows the r_i. Otherwise, the vehicle transmits its current location to the server for update. Here, ϵ is a system parameter, which is set to 100 (m) in this paper. Similarly the distance between p_3 and a line connecting p'_2 and p'_3 is measured in Subfig. 10(c).

Notations:
- p_s, p_e, S, S_a: Defined before.
- p_r: A point on a route.
- R_a: A set of routes which are connected with an articulation point or a starting point of a route. An initial value is set to $\emptyset$.
- id_{prev}: A route id of a path's previous point. An initial value is set to 0.
- id: A route id.

◇ Traverse $\forall$ $pt.p_i$ from the starting point ($pt.p_s$) to the ending point ($pt.p_e$),
  $p_r \leftarrow$ getRoutePoint($pt.p_i$);
◇ When the current point is on a route ($p_r \in S$) and is not an articulation point ($p_r \notin S_a$),
  $id \leftarrow$ getRouteId(p_r);
  if $id \neq id_{prev}$
    Execute the trajectory generation module;
    $R_a \leftarrow$ getConnectedRouteIds($r_{id}.p_s$);
◇ When the current point is on a route ($p_r \in S$) and is an articulation point ($p_r \in S_a$),
  $R_a \leftarrow$ getConnectedRouteIds(p_r);
◇ Post processing step,
  if $p_r \notin S_a$
    $id_{prev} \leftarrow id$;
  $R_a \leftarrow \emptyset$;
◇ When the current point is not on a route ($pt.p_i \notin S$),
  Identify a new route by using the route extraction module;

FIGURE 11. The pseudo code of route recognition.

Second, we consider the number of points in a route (n_r) and the number of points following a path from the n_r (n_p) to determine whether the route can be added into a trajectory. If the ratio of n_p to n_r is greater than a threshold value (th_v), the route is added into the trajectory, where we set the th_v to 0.6 in this paper. This is because we observe that a route consisting of few points (i.e., four or five) often does not satisfy the ratio greater than the th_v in our experiments. Thus, we try not to set the th_v higher than 0.6 to add more routes in the trajectory.

Third, since a trajectory consists of multiple routes identified by the server, it can be partially updated with a newly found route that is matched to the path. Then the server can wirelessly transmit the most recently updated trajectory to the vehicles while they are on the roads to prevent them from frequently transmitting their locations for update. Since this transmission is often a one-time operation and the size of trajectory information is small, the communication overhead can be negligible. For example, a size of trajectory consisting of 69 routes is around 58.7 KByte. Note that each vehicle pre-loads a trajectory what it plans to follow before departing. The major operations of trajectory generation are summarized in Fig. 12.

C. ENHANCEMENT

In the location update data stored in the server, GPS coordinates of each location are in fact real values, such as (37.47045031793510, 126.61817906322101) or (37.46958210000000, 126.62285199999999). Two location update data transmitted from the same location may not have the exact same coordinates. In light of this, we deploy

Notations:
- r, id, p_s, p_e: Defined before.
- d: A distance between a route's point and a line connecting p_{prev} and p_{next}.
- n_r, n_p: A number of route's points and a number of route's points which follow a path, respectively.
- p_{prev}, p_{next}: A previsous point and a next point of the current point on a path, respectively.
- ε: A threshold value about the distance between a point and a line.
- th_v: A threshold value to determine whether a path segment is same as a route or not.
- tr: A trajectory.
- l: A last trajectory id.

◇ Append a route id whose ratio exceeds the threshold value to a trajectory,
  $n_r \leftarrow 0$; $n_p \leftarrow 0$;
  for $\forall\ r_{id}.p_k$ from $r_{id}.p_s$ to $r_{id}.p_e$ **do**
    $n_r \leftarrow n_r + 1$;
    $p_{prev} \leftarrow$ getPreviousPoint($r_{id}.p_k$);
    $p_{next} \leftarrow$ getNextPoint($r_{id}.p_k$);
    if $p_{prev} =$ NULL **or** $p_{next} =$ NULL
      $n_p \leftarrow n_p + 1$;
    else
      $d \leftarrow$ distance(p_{prev}, p_{next}, $r_{id}.p_k$);
      if $d < \varepsilon$
        $n_p \leftarrow n_p + 1$;
  if $\frac{n_p}{n_r} > th_v$
    Append a route r_i to a trajectory tr_l;
    $l \leftarrow l + 1$;

FIGURE 12. The pseudo code of trajectory generation.

a two-dimensional tree[2] (later in short, 2-d tree) [22], [23], widely used for range search in diverse geometric problems, to efficiently determine whether a location update datum is found in the existing route. The basic idea is to create a virtual square area with two diagonal points, which are system parameters (α and β), in the 2-d tree and determine whether a given location l (e.g., (x_l, y_l)) is located within the square area (see also Fig. 5). We first visit the tree from the root node containing a location r (e.g., (x_r, y_r)) and check whether the y_r-coordinate of root node is located between $y_l - \beta$ and $y_l + \beta$. Then we have three cases: (i) the y_r-coordinate is located between $y_l - \beta$ and $y_l + \beta$; (ii) the y_r-coordinate is less than $y_l - \beta$; and (iii) the y_r-coordinate is greater than $y_l + \beta$. In the case of (i), if the coordinates of root node are located within the square area, both left and right subtrees of root node are visited. In the case of (ii) (or (iii)), the left (or right) subtree of the root node is visited. Second, we visit the nodes in the next level of tree, check the x-coordinate of node is located between $x_l - \alpha$ and $x_l + \alpha$, and conduct the same operations corresponding to the aforementioned three cases, respectively. Whenever the level of tree is incremented, the current axis of coordinate to check is altered. The 2-d tree is traversed by incrementing the level of tree until the last leaf node is visited. If there is a node whose location l' (e.g., $(x_{l'}, y_{l'})$) is located within the square area, both l and l' are considered as the same locations. Third, we search the route database with the location l' to find any existing route that includes the l'.

[2]A 2-d tree is a binary search tree built with the locations of vehicles as a key, e.g., x- and y-coordinates, in this paper. An insert operation used in normal binary search trees can also be used to insert a point into the 2-d tree. The 2-d tree is traversed starting from the root node by comparing the y-coordinate first. If a point to be inserted has a smaller (or larger) y-coordinate than that of the root node, the left (or right) subtree is searched. In the subtree, the x-coordinate is compared with the node. This axes-alternating comparison and search by incrementing the level of tree is repeated until the position is found or the last leaf node is visited.

V. PERFORMANCE EVALUATION

We implement the proposed scheme and compare and analyze its performance with two competitive approaches in terms of four major performance metrics. We also discuss potential implementation issues.

A. IMPLEMENTATION TESTBED AND ISSUES

We implement the T-Reduce scheme and conduct extensive experiments with a Microsoft Windows 10 based computer equipped with 3.4 GHz Intel i7-2600 CPU and 16 GB main memory. The major operations of T-Reduce scheme including path refinement and route matching including three sub-operations are implemented with C++. The APIs of Daum Maps [24] and Matlab are also used to visualize trajectories on the map and compute trajectory reconstruction error, respectively. In this paper, we use a real trace data of vehicles collected from a logistics company (Incheon, South Korea). The trace data of vehicles are a set of GPS coordinates periodically measured and collected every minute. Since we consider a freeway network where there are not many exit and entrance points, this update period is enough to easily expect the routes of vehicles. For example, a vehicle runs to the east in a freeway where there is no exit and entrance points for the next 10 miles. In case of when an urban road network is considered, however, the update period should be shorter to improve the accuracy of route recognition because of intersections and traffic lights.

TABLE 1. Execution result of route recognition algorithm.

	May	June	July	Total
Number of paths (A)	357	403	383	1,143
Number of recognized routes (B)	4,444	4,710	5,003	14,157
Average number of recognized routes per path (B/A)	12.4	11.7	13.1	12.4

Initially, 1,143 paths were collected from vehicles for three months, May to July 2016, for experiments as shown in Table 1. After applying our route extraction sub-operation to the collected paths, we are able to identify many routes and the average number of recognized routes per path is 12.4. Here, an initial route database has been constructed based on the collected paths from vehicles at April 2016. A set of recognized routes is depicted on the map with articulation points in Fig. 13.

B. EXPERIMENT RESULTS AND PERFORMANCE COMPARISON

For performance evaluation and comparison, we first conduct extensive experiments and measure the performance in

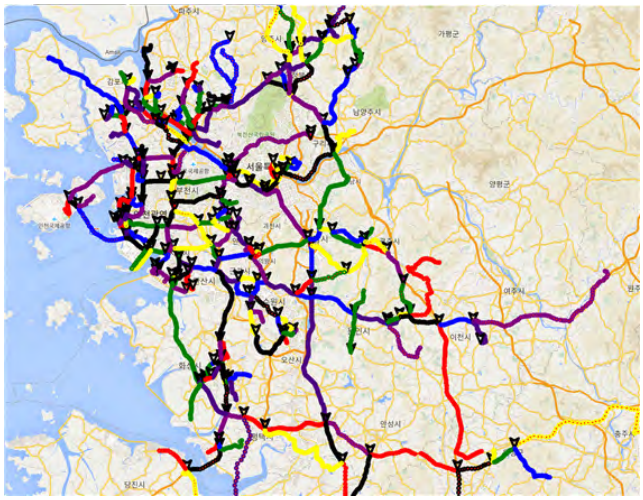


FIGURE 13. An initial set of routes stored in the route database is depicted using different colors on the map. Here, an articulation point is marked as a triangle.

terms of four major performance metrics: average trajectory error, efficiency of route recognition, size of accumulated trajectories, and cost efficiency. We compare the proposed scheme with the trajectory-based approach (TB) [15] and the corner-based approach (CB) [17]. In the TB, vehicle updates the location only if the current trajectory exceeds a predefined trajectory error bound. Here, the trajectory error is the difference between the actual trajectory and the trajectory reconstructed from location update data. However, the CB combines the feature location detection with the online trajectory-based sampling algorithm to minimize the average trajectory error with the minimal number of location updates.

1) AVERAGE TRAJECTORY ERROR

A trajectory error is to measure the distance between the original path and its reduced or recognized route as shown in Fig. 14. In Table 2, the trajectory errors of each approach

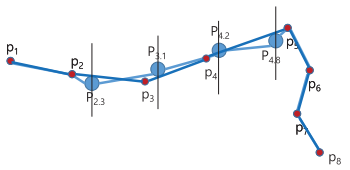


FIGURE 14. The trajectory errors between the original paths and their reconstructed routes.

TABLE 2. Comparisons of average trajectory errors.

Month	Average Trajectory Error (m)		
	TB	CB	T-Reduce
May	21.5786	22.3721	14.4399
June	25.6820	38.4679	17.6755
July	26.8382	40.0130	22.3721
Average	24.6996	33.6177	**18.1625**

are averaged and compared based on the collected paths during May to July 2016. Since both TB and CB schemes do not have reference routes (i.e., identified routes), the trajectory errors are the distance between input paths and their reduced routes. However, the T-Reduce scheme calculates the distance between input paths and their recognized routes. Our approach shows the lowest error and outperforms both TB and CB schemes for entire months. Note that since the GPS coordinates of each location is given in terms of latitude (ϕ) and longitude (λ), we need to convert them in the unit of meter (m).

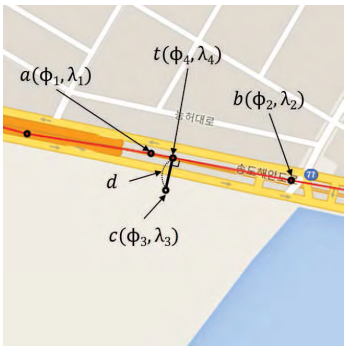


FIGURE 15. The distance between a line ($\overline{ab}$) and a location (c).

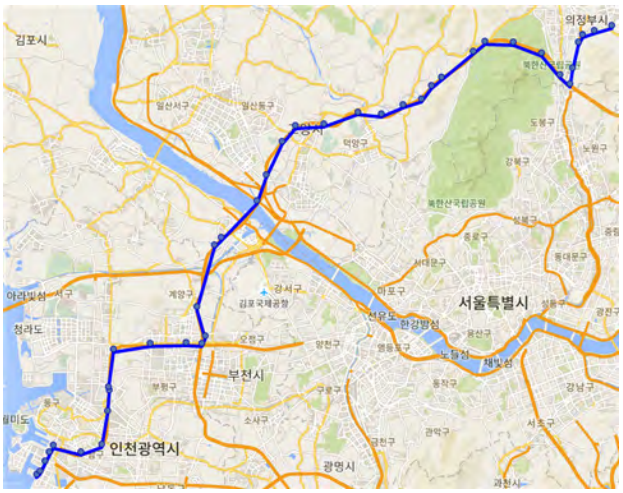
In Fig. 15, suppose there are a location c and a line $\overline{ab}$ that connects the a and b. Both a and b are the closest locations on the original input path. Then the distance between the c and $\overline{ab}$ can be calculated by using the Haversine equation.[3]

2) EFFICIENCY OF ROUTE RECOGNITION

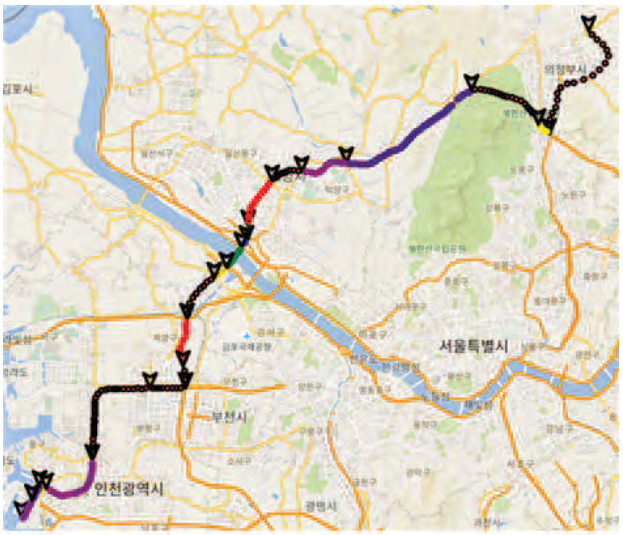
In Fig. 16, we show a sample path and its recognized routes. A vehicle identified as its *id* DT6111 has moved from the left lower corner to the right upper corner in the map as shown in Subfig. 16(a). The vehicle transmits its current location to the server every 60 seconds. Then the server recognizes a set of routes and articulation points in Subfig. 16(b). Note that all routes are already recognized routes stored in the route database and no newly recognized route is used. When multiple vehicles transmit their location update data, however, it is expected for the server to extensively search exiting routes to identify a new route.

Thus, the efficiency of route recognition operation is measured in terms of the number of comparisons between newly identified routes and existing routes. Since each route consists of at least more than two locations, the number of comparisons will quickly increase as the number of routes increases and each route becomes longer. In this paper, we deploy a 2-d tree based on the frequency of routes in order to reduce the number of comparisons. The rationale behind this approach

[3] We use four location data, a(ϕ_1, λ_1), b(ϕ_2, λ_2), c(ϕ_3, λ_3), and t(ϕ_4, λ_4). We first calculate both $\Delta\phi = \phi_4 - \phi_3$ and $\Delta\lambda = \lambda_4 - \lambda_3$. Then $a = \sin^2(\frac{\Delta\phi}{2}) + \cos(\phi_3) \cdot \cos(\phi_4) \cdot \sin^2(\frac{\Delta\lambda}{2})$, $c = 2 \cdot \tan^{-1}(\frac{\sqrt{a}}{\sqrt{1-a}})$, and $d = R \cdot c$. Here, R is the radius of the Earth and d is the distance between the location c and the line $\overline{ab}$.



(a)



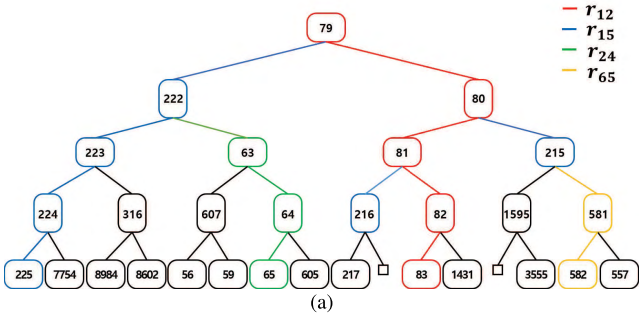
(b)

FIGURE 16. AA sample path and its recognized routes are presented using different colors. Here, a vehicle identified as its *id* DT6111 has transmitted a set of location update data. Here, an articulation point is marked as a triangle. (a) A sample path. (b) A set of recognized routes.

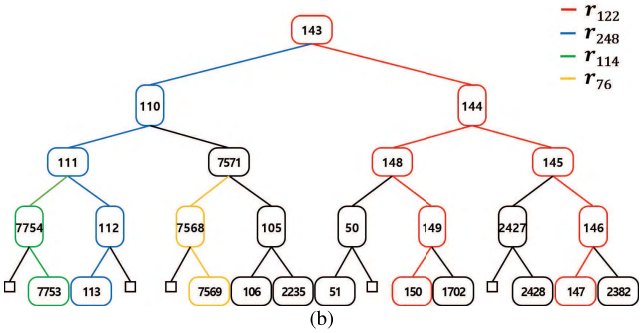
is that some routes are frequently used but some routes do not during the route recognition operation. Since the 2-d tree is a binary search tree, a node closely located to the root can be accessed quickly with the less number of comparisons. Thus, the locations included in most frequently used routes are intentionally located close to the root. In Table 3, most frequently used routes are summarized with their route IDs based on the paths collected at May 2016. For example, the route r_{122} is used 142 times for the route recognition operation.

TABLE 3. A rank of route frequency based on the paths collected in May, 2016.

Rank	Frequency	Route *id*	Rank	Frequency	Route *id*
1	142	r_{122}	6	90	r_{39}
2	126	r_{248}	7	71	r_{177}
3	122	r_{114}	8	65	r_{33}
4	101	r_{76}	9	64	r_{38}
5	96	r_{12}	10	58	r_{27}



(a)



(b)

FIGURE 17. The 2-d trees ordered either by route number or frequency, where a node closely located with the root is accessed quickly with the less number of comparisons. (a) Route number. (b) Route frequency.

Fig. 17 shows two 2-d trees constructed by route number and route frequency, respectively. In Subfig. 17(a), a 2-d tree is constructed with four routes, $r_{12} = \{p_{79}, p_{80}, p_{81}, p_{82}, p_{83}\}$, $r_{15} = \{p_{215}, p_{216}, p_{222}, p_{223}, p_{224}, p_{225}\}$, $r_{24} = \{p_{63}, p_{64}, p_{65}\}$, and $r_{65} = \{p_{581}, p_{582}, \}$ based on the route numbers. Here, four route numbers are not consecutively ordered because routes numbered between can further be divided and even deleted in the route recognition operation. In Subfig. 17(b), however, a 2-d tree is constructed with four routes, $r_{122} = \{p_{143}, p_{144}, p_{145}, p_{146}, p_{147}, p_{148}, p_{149}, p_{150}\}$, $r_{248} = \{p_{110}, p_{111}, p_{112}, p_{113}\}$, $r_{114} = \{p_{7754}, p_{7753}\}$, and $r_{76} = \{p_{7568}, p_{7569}\}$ based on their frequencies. Since locations (e.g., p_{143}, p_{144}, and p_{110}) included in highly used routes (e.g., r_{122} and r_{248}) are placed close to the root, less number of comparisons is required compared to that of locations placed based on the route numbers. We construct two 2-d trees based on the collected paths during May to July 2016, conduct the route recognition operation, and count the number of comparisons. We can significantly reduce the number of comparisons by more than 20% in average using the 2-d tree constructed by route frequency in Table 4.

3) SIZE OF ACCUMULATED TRAJECTORIES

Based on the collected paths during May to July 2016, the size of accumulated trajectories stored in the server are measured

TABLE 4. A number of comparisons between newly identified routes and existing routes using 2-d trees constructed by route number or frequency.

Month	Total Number of Comparisons		Reduction Ratio ($\frac{N_a - N_f}{N_a} \cdot 100$)		
	Ordered by Route Number (N_a)	Ordered by Frequency (N_f)	Maximum	Minimum	Average
May	2,933,076	2,261,286	51.0109	1.6793	**23.48410**
June	3,709,182	2,843,982	46.6899	0.1741	**23.86954**
July	1,900,225	1,492,163	44.2194	0.4027	**22.63416**

TABLE 5. Comparisons of accumulated trajectory size (Unit: Kbyte).

Month	OP	TB	CB	T-Reduce
May	2,652	1,083	1,005	315
June	3,057	1,216	1,135	317
July	2,278	876	799	318
Average	2,662	1,058	980	**317**

and compared in Table 5. Here, the OP represents original paths without applying any scheme and thus, it is used as the performance low-bound in this paper. The T-Reduce scheme can reduce the size of accumulated trajectories by 88%, 70%, and 68% compared to the OP, TB, and CB schemes, respectively. Although the T-Reduce scheme yields the smallest size, it requires an initial route information a priori, e.g., a route database. The size of initial route information is about 265 Kbyte including initial routes, locations, and articulation points. Since the proposed route recognition operation outputs only a route ID, it increases 275 bytes per route in average.

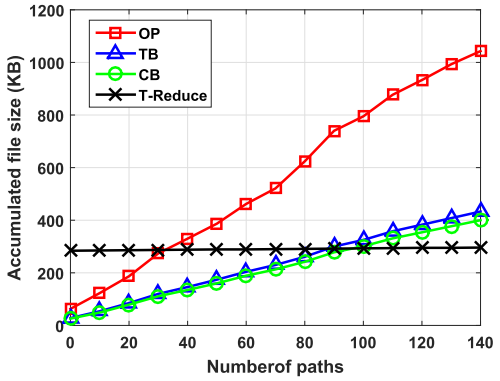


FIGURE 18. Comparisons of the accumulated trajectory size. Here, the OP represents original paths without applying any scheme.

Fig. 18 shows the size of accumulated trajectories stored in the server based on the paths collected in May 2016. The T-Reduce scheme shows higher size of accumulated trajectories than that of other schemes in the lower number of paths because of the initial route information. As the number of paths increases, however, our approach does not increase the size much compared to other rapidly increasing schemes. Thus, unlike our approach, the OP, TB, and CB show the lack of scalability.

4) COST EFFICIENCY

In this paper, the cost efficiency proposed in [17] is measured and expressed as, $\mu = \frac{E_{TB}}{E_{T-Reduce}} \cdot \frac{S_{TB}}{S_{T-Reduce}}$ or $\mu = \frac{E_{CB}}{E_{T-Reduce}} \cdot \frac{S_{CB}}{S_{T-Reduce}}$, where E and S are the average trajectory error and the size of accumulated trajectories, respectively. Table 6 summarizes the comparison results between the T-Reduce scheme with both TB and CB schemes in terms of the average trajectory error, size of trajectories, and cost efficiency. In Table 6, our approach shows 4.54 and 5.72 cost efficiencies compared to both TB and CB schemes, respectively.

TABLE 6. The summary of comparisons of cost efficiency.

	TB	CB	T-Reduce
Average Trajectory Error (E) (Unit: m)	24.6996	33.6177	18.1625
Average Size of Accumulated Trajectories (S) (Unit: Kbyte)	1,058	980	317
Cost Efficiency (μ)	**4.54**	**5.72**	

C. DISCUSSION

Both TB and CB schemes primarily focus on how to efficiently minimize the number of location update data transmitted from vehicles in order to reduce the size of trajectories stored in the server. In the TB scheme, each vehicle sends the current location whenever its actual trajectory exceeds a predefined trajectory error bound from the trajectory stored in the server. In the CB scheme, however, a set of corner locations on the road is identified in advance and each vehicle sends the current location whenever it passes the corner location. In our experiment, 48 Bytes is required to store a single GPS coordinates. The average number of GPS coordinates counted in a path is 154. We observe that both TB and CB schemes can reduce this number to 63 and 58 on average, respectively. The TB scheme requires an average of 3,033 Bytes per path but the CB scheme requires 2,815 Bytes. Each trajectory is stored as a set of location update data in the server and requires at least the minimum number of location data to keep its shape. Thus, both TB and CB schemes have an inherent constraint to reduce the size of trajectories but hope to receive the less number of location update data. Unlike prior approaches, the T-Reduce scheme does not minimize the number of location update data transmitted from vehicles but identifies a set of routes. Each trajectory is recognized in terms of route and its corresponding route *ids* are stored in the server rather than storing entire GPS coordinates involved. In our experiment, three Bytes are required to store a single route *id* and 16 route *ids* are found in a path. Then 48 Bytes are required to store the path, which is the same amount of space required to store a single GPS coordinates. Thus, the T-Reduce scheme can significantly reduce the trajectory data stored in the server compared to both TB

and CB schemes. In addition, we envision that the T-Reduce scheme can be deployed as a part of major application in Intelligent Transportation Systems (ITS), where trajectories of vehicles show an expectable pattern. For example, route information of short- or long-distance buses is already established and buses run only within the fixed routes. Rather than storing entire location update data periodically transmitted from buses in a traditional approach, the T-Reduce scheme can only store the identified route ids.

VI. CONCLUSION

In this paper, we investigated a trajectory data reduction mechanism to efficiently reduce the size of trajectory data stored in the server. In order to realize this, we proposed path refinement and route matching operations including three sub-operations, route extraction, route recognition, and trajectory generation. We setup a small-scale testbed with a real-world trace data collected from a logistic company and conducted extensive experiments for performance evaluation and comparison. The experiment results show that the proposed approach can reduce the average trajectory error and route information up to 26.4% and 88%, respectively compared to two existing schemes. Moreover, the proposed approach can achieve up to 5.72 times cost efficiency than that of the existing schemes.

In this paper, we first recognized the routes based on the paths collected during April to July 2016. To see the full potential of proposed approach, we plan to keep on collecting the paths from vehicles and recognizing the routes until no more new routes are found. Since the road infrastructure where vehicles move is very limited in South Korea, we expect that the route database will ultimately be saturated. Thus, the proposed route recognition operation can be expedited. Second, we focused on a freeway network where vehicles repeatedly use the same routes and their trajectories show an expectable pattern. We also plan to extend the T-Reduce scheme by considering an urban road network where vehicles turn frequently, and their routes become complex and shorter because of intersections and traffic signals. To efficiently recognize routes, three possible cases will further be investigated: (i) when a route is skipped between two location update points; (ii) when a location update point skips an articulation point; and (iii) a wrong route is recognized due to the inaccuracy of GPS.

REFERENCES

[1] *Global Express and Small Parcels*, Transport Intell., Bath, U.K., 2015.

[2] Korea Transportation Institute. (Sep. 2015). *Montly/Yearly Truck Registration Information*. [Online]. Available: http://www.ktdb.go.kr/

[3] Y. Zhang, J. Zhao, and G. Cao, "On scheduling vehicle-roadside data access," in *Proc. ACM VANET*, 2007, pp. 9–18.

[4] Gett. *A Ride Sharing App*. Accessed: Jul. 2018. [Online]. Available: https://gett.com/juno/

[5] Lyft. *A Ride Sharing App*. Accessed: Jul. 2018. [Online]. Available: http://www.lyft.com/app/

[6] Sidecar. *A Ride Sharing App*. Accessed: Jul. 2018. [Online]. Available: http://www.side.cr/

[7] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica*, vol. 10, no. 2, pp. 112–122, 1973.

[8] H. Qian and Y. Lu, "Simplifying GPS trajectory data with enhanced spatial-temporal constraints," *ISPRS Int. J. Geo-Inf.*, vol. 6, no. 11, pp. 329–353, 2017.

[9] H. Imai and M. Iri, "Polygonal approximations of a curve—Formulations and algorithms," in *Proc. Comput. Morphol.*, 1988, pp. 71–86.

[10] M. R. Fard, A. S. Mohaymany, and M. Shahri, "A new methodology for vehicle trajectory reconstruction based on wavelet analysis," *Transp. Res. C, Emerg. Technol.*, vol. 74, pp. 150–167, Jan. 2017.

[11] J. Zhou, H. Leong, Q. Lu, and K. C. K. Lee, "Optimizing update threshold for distance-based location tracking strategies in moving object environments," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Jun. 2007, pp. 18–21.

[12] E. Pitoura and G. Samaras, "Locating objects in mobile computing," *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 4, pp. 571–592, Jul. 2001.

[13] R. Lange, F. Dürr, and K. Rothermel, "Online trajectory data reduction using connection-preserving dead reckoning," in *Proc. Mobile Ubiquitous Syst., Comput., Netw., Services (Mobiquitous)*, 2008, p. 52.

[14] R. Lange, T. Farrell, F. Durr, and K. Rothermel, "Remote real-time trajectory simplification," in *Proc. Pervasive Comput. Commun.*, Mar. 2009, pp. 1–10.

[15] H. Park, Y.-J. Lee, J. Chae, and W. Choi, "Online approach for spatio-temporal trajectory data reduction for portable devices," *J. Comput. Sci. Technol.*, vol. 28, no. 4, pp. 597–604, 2013.

[16] J. Tang, S. Zhang, Y. Zou, and F. Liu, "An adaptive map-matching algorithm based on hierarchical fuzzy system from vehicular GPS data," *PLoS ONE*, vol. 12, no. 12, p. e0188796, Dec. 2017.

[17] J. Kim, I. Kim, N. Kwon, H. Park, and J. Chae, "A hybrid algorithm for online location update using feature point detection for portable devices," *KSII Trans. Internet Inf. Syst.*, vol. 9, no. 2, pp. 600–619, 2015.

[18] K. Zheng, Y. Zheng, X. Xie, and X. Zhou, "Reducing uncertainty of low-sampling-rate trajectories," in *Proc. Int. Conf. Data Eng. (ICDE)*, Apr. 2012, pp. 1144–1155.

[19] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proc. Alvey Vis. Conf.*, 1988, pp. 147–151.

[20] J. Matoušek, "Geometric range searching," *ACM Comput. Surv.*, vol. 26, no. 4, pp. 422–461, 1994.

[21] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, "Orthogonal range searching: Querying a database," in *Computational Geometry: Algorithms and Application*, 3rd ed. Oxford, U.K.: Springer, 2008, ch. 5, pp. 95–120.

[22] D. E. Willard, "New data structures for orthogonal range queries," *SIAM J. Comput.*, vol. 14, no. 1, pp. 232–253, 1985.

[23] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[24] *Daum Maps API*. Accessed: Jul. 2018. [Online]. Available: http://apis.map.daum.net/

NAMGU KWON received the B.S. degree in computer science and engineering from Incheon National University, South Korea, in 2013, where he is currently pursuing the M.S. degree with the Graduate School of Computer Science and Engineering. His research interests include Internet software and mobile computing.

JIBUM KIM received the B.S. and M.S. degrees in electrical engineering from Yonsei University, Seoul, South Korea, in 2003 and 2005, respectively, and the Ph.D. degree in computer science and engineering from The Pennsylvania State University, University Park, PA, USA, in 2012. In 2013, he joined the Los Alamos National Laboratory (Group T-5) as a Post-Doctoral Fellow. He is currently an Associate Professor with the Department of Computer Science and Engineering, Incheon National University, South Korea. His current research interests include computational science, artificial intelligence, and high-performance computing.

SUNHO LIM (SM'01) received the B.S. degree (*summa cum laude*) in computer science and the M.S. degree in computer engineering from Korea Aerospace University, Goyang, South Korea, in 1996 and 1998, respectively, and the Ph.D. degree in computer science and engineering from The Pennsylvania State University, University Park, PA, USA, in 2005. He was an Assistant Professor with the Department of Electrical Engineering and Computer Science, South Dakota State University, Brookings, SD, USA, from 2005 to 2009. He is currently an Assistant Professor with the Department of Computer Science, Texas Tech University (TTU), Lubbock, TX, USA. He leads the T^2WISTOR: TTU Wireless Mobile Networking Laboratory. His research interests include areas of cybersecurity, mobile data management and privacy, and wireless networks and mobile computing. He was a recipient of the Texas Tech Alumni Association New Faculty Award, Air Force Summer Faculty Research Fellowship, and Office of Navy Summer Faculty Research Fellowship. He is a Most Influential Faculty Member and Institute for Inclusive Excellence Fellow at TTU. He was a Guest Editor of Special Issue on Dependability and Security for Wireless Ad Hoc and Senor Networks and Their Applications in the *International Journal of Distributed Sensor Networks*. He served on the NSF proposal panel and program committees of many renowned conferences.

JINSEOK CHAE (M'92) received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, South Korea, in 1990, 1992, and 1998, respectively. In 2006, he joined the Department of Computer Science, California State University San Bernardino, California, USA, as a Visiting Scholar. From 2012 to 2014, he was the Dean of Admissions and Student Affairs with Incheon National University, South Korea. From 2015 to 2017, he was a Visiting Scholar with the Department of Computer Science, Texas Tech University, TX, USA. He was with the Engineering Laboratory, Seoul National University, and also with the Korea Research Information Center, South Korea. He is currently a Professor with the Department of Computer Science and Engineering, Incheon National University. His research interests include Internet software, Web technology, and mobile computing. He was an Editor-in-Chief of the *Journal of KIISE: Computing Practices and Letters* from 2011 to 2014.

HEEMIN PARK (M'04) received the B.S. and M.S. degrees in computer science from Sogang University, South Korea, in 1993 and 1995, respectively, and the Ph.D. degree in electrical engineering from the University of California, Los Angeles, in 2006. He was with Yonsei University, Sookmyung Women's University, and Samsung Electronics, South Korea. He is currently an Assistant Professor with the Department of Computer Software Engineering, Sangmyung University, Cheonan, South Korea. His research interests include Web-based information systems, networked and embedded computing systems, cyber physical systems, multimedia applications, and entertainment computing.