Knowl Inf Syst (2011) 29:25–53
DOI 10.1007/s10115-010-0333-9

REGULAR PAPER

Mining fastest path from trajectories with multiple destinations in road networks

Eric Hsueh-Chan Lu · Wang-Chien Lee · Vincent S. Tseng

Received: 21 January 2010 / Revised: 4 May 2010 / Accepted: 17 July 2010 /
Published online: 8 August 2010

Abstract Nowadays, research on Intelligent Transportation System (ITS) has received many attentions due to its broad applications, such as path planning, which has become a common activity in our daily life. Besides, with the advances of Web 2.0 technologies, users are willing to share their trajectories, thus providing good resources for ITS applications. To the best of our knowledge, there is no study on the fastest path planning with multiple destinations in the literature. In this paper, we develop a novel framework, called *Trajectory-based Path Finding (TPF)*, which is built upon a novel algorithm named *Mining-based Algorithm for Travel time Evaluation (MATE)* for evaluating the travel time of a navigation path and a novel index structure named *Efficient Navigation Path Search Tree (ENS-Tree)* for efficiently retrieving the fastest path. With MATE and ENS-tree, an efficient fastest path finding algorithm for single destination is derived. To find the path for multiple destinations, we propose a novel strategy named *Cluster-Based Approximation Strategy (CBAS)*, to determine the fastest visiting order from specified multiple destinations. Through a comprehensive set of experiments, we evaluate the proposed techniques employed in the design of TPF and show that MATE, ENS-tree and CBAS produce excellent performance under various system conditions.

Keywords Data mining · Intelligent transportation system · Trajectory · Path planning

1 Introduction

In recent years, research on Intelligent Transportation System (ITS) has received a lot of attention due to its broad application base in our life. With the popularity of Global

E. H.-C. Lu · V. S. Tseng (✉)
Department of Computer Science and Information Engineering,
National Cheng Kung University, Tainan, Taiwan, ROC
e-mail: tsengsm@mail.ncku.edu.tw

W.-C. Lee
Department of Computer Science and Engineering, Pennsylvania State University,
University Park, PA 16802, USA

Positioning System (GPS)- based navigation devices, many applications have been developed on GPS-based mobile devices [14], including GPS navigation systems, real-time traffic information, park search systems, weather reports, and other location-based services [19]. While many interesting and useful GPS-related applications and services are expected to come to our life in the near future, path/route planning has become a common activity in daily use of the GPS devices and thus deserves more research efforts.

Owing to the rapid advances of the Web 2.0 technology, many GPS device users are willing to share their trajectories with others. For example, in [28,33,38,43,44], the authors discuss how to collect and analyze user trajectories from GPS devices in road networks. From the collected trajectories, advanced location-based services can be developed. For instance, user behavior mined from the dataset under different context and locations can be taken into account in development of context- and location-aware applications. Moreover, frequent trajectory patterns discovered from the collected trajectory database can benefit many ITS applications, e.g., traffic monitoring and control, traffic flow prediction, navigational path planning and routing, etc.

In the above applications, mobile user trajectories are usually constrained by the road network, which is composed of multiple edges (road segments) and nodes (intersections). When mobile users move, their locations, positioned by their GPS devices, are uploaded and stored in a centralized GPS trajectory database. In this database, a trajectory is composed of a series of (time, location) tuples which denote a particular user trajectory, e.g., (08:00, location_A), (08:10, location_B), . . ., and (17:00, location_Z), where location_A, location_B, and location_Z are coordinates (latitudes and longitudes) of those locations.

The GPS trajectory database is valuable to many ITS applications since the huge amount of trajectory data available from the users can best capture the traffic situations on the road segments. For example, R_{AB} is a road segment from intersection A to intersection B in the city center. We can observe that most of users take a long time to move from A to B between commuting hours (e.g., from 07:00AM to 08:00AM) by analyzing the GPS trajectory database. Hence, a navigation path that avoids R_{AB} may be faster than other routes that take R_{AB} during commuting hours. In this paper, we adopt data mining techniques to exploit valuable traffic information in large and complex trajectory database for path planning.

A number of studies have discussed the issues of navigation path planning in the road network environment. These planning strategies include the shortest travel distance path [15,29,35], the least free-flow time path [6], the popular path [10], and the fastest travel time path [4–6,11,18,25,34]. In these path planning studies, the planning are mostly based on various static cost estimates of road segment, e.g., distance or maximum velocity constraint, etc. However, the traffic condition varies at different time. Deriving a time-varied traffic model for conventional path planning techniques is very challenging and costly. Due to the availability of trajectories enable by Web 2.0 technology, a new idea to address the path planning problem is to employ trajectory mining techniques to capture variable traffic conditions in finding the fastest paths. In this paper, we present our approach for supporting path planning with multiple destinations. To the best of our knowledge, there is no prior study on the fastest path planning with multiple destinations based on trajectory mining techniques appeared in the literature.

In this paper, we develop a new system framework, called Trajectory-based Path Finding (TPF), based on a data mining approach for finding the fastest navigation path with multiple destinations. TPF is built upon a novel data mining algorithm, namely Mining-based Algorithm for Travel time Evaluation (MATE), which estimates the travel time of a navigation path and a novel index structure, called Efficient Navigation Path Search Tree (ENS-Tree), for efficient retrieval of the fastest navigation path. With MATE and ENS-tree, an efficient

fastest path finding algorithm for single destination is derived. To find the fastest path with multiple destinations in TPF, we propose a novel strategy, namely Cluster-Based Approximation Strategy (CBAS), to find the fastest visiting order for specified destinations.

Our contributions in this research study are five-fold:

1. We propose the TPF framework, a new approach for path planning. The problems and ideas in TPF have not been well explored in the research community.
2. We propose the MATE algorithm, a new technique for automatic capturing the traffic conditions in GPS trajectory database. MATE addresses the missing data problem by interpolation based on time segmentation.
3. We propose the ENS-Tree structure to significantly alleviate the computation cost and memory overhead in the retrieval of the fastest navigation path.
4. We propose the CBAS approach for efficient and effective discovery of the fastest visiting order for specified destinations.
5. Through a comprehensive empirical evaluation and sensitivity analysis, we show that MATE, ENS-tree and CBAS produce excellent performance under various system conditions.

The remaining of this paper is organized as follows. We briefly review the related work in Sect. 2. We first describe the problem in Sect. 3 and then present the proposed CBAS strategy in Sect. 4. We report the result of empirical performance evaluation in Sect. 5. Finally, we conclude the paper and discuss the future work in Sect. 6.

2 Related work

In this section, we review previous studies, which can be classified into four aspects: (1) the shortest path planning, (2) the fastest path planning, (3) traveling salesman problem, and (4) data mining.

Most prior studies on navigation path discovery address the shortest path related problems. Dijkstra proposed an algorithm [13] to find the shortest distance path between two nodes. The basic idea of Dijkstra's algorithm is to consider the corresponding cost of each neighboring node from the start node and choose the one with the least cost to expand. The process is repeated recursively until the destination node is reached. Hart et al. proposed the A* algorithm [22], which is a greedy best-first search algorithm. A* algorithm uses a heuristic function to guide the search from the start location toward the destination location. The heuristic function is divided into two functions: the path-cost function and the heuristic estimate function. The path-cost function is the real cost from start location to an intermediate location under examination, and the heuristic estimate function is the estimation cost from examined intermediate location to the destination location. While the efficiency of the search algorithm can be improved, the algorithm may fail to find the shortest path. Lim et al. proposed a link-based shortest path algorithm [30] to generate dissimilar paths for the travel information in real road network where exists turn prohibitions. The authors observed that heavy traffic may concentrate on some specific paths, which may lead to traffic oversaturation. Therefore, the authors used an overlap degree function to exclude the overlapping alternatives. However, due to the dynamic traffic of the road network, the shortest path is not necessarily the fastest one. In [6], Bekhor et al. evaluated a number of path generation strategies that include the least distance path, the least free-flow time path, and the least time path. The difference between various strategies is to plan navigation paths using different edge costs, e.g., the edge cost of the least distance path is the

distance; the edge cost of the least free-flow time path is the distance divided by the maximum velocity constraint; and the edge cost of the least time path is the estimated travel time.

The fastest path problem, an extension of the shortest path problem, changes the edge costs from the road distance to incorporate time-related factors, including the travel time, the maximal velocity constraint, and so on. In [4], Awasthi et al. proposed a rule-based method for evaluating the fastest paths on urban networks. This paper used the traffic log to build the statistical model for deciding the predictive fastest path. However, the model does not consider the starting time. In other words, the fastest paths at different starting time points may be different. In [25], Kanoulas et al. proposed the traffic speed pattern named CAtegorized PiecewisE COnstant speeD (CapeCod) pattern to represent the velocity of vehicle at different time points. This paper used A* algorithm to find the fastest path, which considers the different starting time points. In [18], Gonzalez et al. considered some environmental factors that may reflect the velocity influence of vehicle and use these factors to build a decision tree. The velocity of vehicle is evaluated based on the current environmental situations captured the decision tree. In [32], Vint et al. proposed the online learning solutions for freeway travel time prediction. In [24], Jula et al. proposed the method to predict travel times along the routes and estimate arrival times at the nodes of a stochastic and dynamic network in real time. In [34], Lu et al. proposed a mining-based algorithm PATE to predict the estimated travel time. This paper used the travel time evaluation table to find the shortest path within a user-specified travel time constraint. In addition, a prefix-tree-based structure, called NPST, was proposed to efficiently find the shortest navigation path. However, all of the researches were focused on the path-finding problem with single destination.

The Travelling Salesman Problem (TSP) [23] is one of the well-known problems in the field of the combinatorial optimization. In TSP, let $C = \{c_1, c_2, \ldots, c_N\}$ be a set of cities and $d(c_i, c_j)$ be the distance of distinct cities $\{c_i, c_j\}$. The goal is to find an ordering π of the cities, which minimizes the quantity $\Sigma_{i=1 \text{ to } N-1}\, d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(N)}, c_{\pi(1)})$. The complexity of TSP is $\boldsymbol{O}(N!)$, where N is the number of locations. Therefore, it is NP-hard [17] and therefore a brute force algorithm for finding optimal combinations has a worst-case running time that grows faster than any polynomial time complexity. Arora proposed polynomial time approximation schemes for Euclidean traveling salesman problem [3]. A number of researches look for heuristics that find near-optimal combinations with low time overhead. The heuristic approaches to the TSP can be divided into two classes. (1) Construction Heuristic Methods: Such heuristics build a solution by a growth process that terminates as soon as a feasible solution is constructed. In [37], Rosenkrantz et al. proposed that the most natural Greedy-based heuristic for TSP is the Nearest Neighbor algorithm (NN). In this algorithm, the traveler always goes the nearest unvisited location. (2) Improvement Heuristic Methods: Such algorithm is specified in terms of a class of operations that can be used to convert one combination into another. In [36], Pepper et al. proposed a heuristic-based algorithm to solve the TSP based on simulated annealing. In [39], Tsai et al. proposed a genetic and evolutionary algorithm to solve the TSP. However, the computation cost of these methods is too high to apply to the real- time navigation systems. Every TSP heuristic method can be evaluated in terms of two key parameters: its running time and the quality of the combinations which it produces [16].

In recent years, a number of studies use data mining techniques to discover useful patterns from transaction database [1,2] and WWW [8,9]. In [1], Agrawal et al. proposed the Apriori algorithm to efficiently discover the association patterns in the transaction database. In [2], Agrawal et al. proposed the AprioriAll algorithm to mine the sequential patterns in

the transaction database. In [8], Borges et al. proposed navigation pattern mining from web navigation database to find a series of consecutive patterns which satisfy the user-specified minimal support threshold. The algorithm is divided into three parts: (1) maximal forward references, (2) large reference sequences, and (3) maximal reference sequences. The navigation pattern is defined as the consecutive and acyclic pattern. In [10], Cheong et al. proposed techniques for popular path mining from navigation log to discover the consecutive patterns which satisfy a minimal frequency threshold. In the data clustering techniques, the most well-known clustering method is the k-Means algorithm [27], which is partition-based. Other partition-based methods contain k-medoids [27], PAM [27] etc. These methods partition the dataset into k clusters, based on similarities between data items, where k is a parameter specified by the user. In [12], Denton et al. proposed pattern-based clustering approach. In [7], Ben-Dor et al. proposed a novel and simple clustering heuristic, called the Cluster Affinity Search Technique (CAST). The CAST requires an affinity threshold t, where $0 < t < 1$. The algorithm guarantees that the average similarity in each generated cluster is higher than the threshold t. Tseng and Kao proposed a novel non-parameter clustering method, called the Smart Cluster Affinity Search Technique (Smart-CAST) [42]. The main ideas of the Smart-CAST are as follows. First, the method uses the CAST as the basic clustering method. Second, the method uses a quality validation method, Hubert's Γ (gamma) statistic [40], to find the best clustering result. Furthermore, the method reduces computations by eliminating unnecessary executions of clustering and by narrowing down the range of the parameter affinity threshold t. In time segmentation researches, Halvey proposed a notion of time segmentation [20] where time segments are predefined by the user. For example, all of the days are divided into workdays and holidays. However, the definition of time segments is difficult. In [21], Halvey proposed an average time segmentation method. The method needs the average value of series v and a user-specified range r. The time segmenting positions are obtained by $r + v$ and $r - v$. However, the user-specified range r is still difficult to define. In [31], Lin proposed the Symbolic Aggregate Approximation (SAX) for labeling the time series. SAX is the first symbolic representation for time series, which allows for dimensionality reduction and indexing with a lower-bounding distance measure.

3 Problem statement

In this section, we first define some terms used in discussion of our research work and then specify our research goal. Table 1 summarizes the notations used in the paper. Let $s = \langle (t_1, l_1), (t_2, l_2), \ldots, (t_m, l_m) \rangle$ be a *GPS trajectory sequence* of a mobile user with length equal to m, where (t_i, l_i) denotes that the mobile user is at the location l_i (latitude and longitude) at time point t_i and $t_i < t_{i+1}, \forall 1 \leq i \leq m$. The elements in a sequence are in ascending order of the time. By stripping off the timestamps, the corresponding *GPS trajectory path* of s is $p_{\text{tra}} = \langle l_1, l_2, \ldots, l_m \rangle$.

Definition 1 Let $p = \langle n_1, n_2, \ldots, n_u \rangle$ be a *navigation path* generated from $p_{\text{tra}} = \langle l_1, l_2, \ldots, l_m \rangle, u \leq m$. For each l in p_{tra}, l can be transferred to p if l is very close to any node (road intersection) in the road network, where n_i denotes the node and n_i and n_{i+1} are connected, $\forall 1 \leq i \leq u$.

Definition 2 A navigation path $p' = \langle n'_1, n'_2, \ldots, n'_v \rangle$ is a *sub-path* of another navigation path $p = \langle n_1, n_2, \ldots, n_u \rangle$, denoted as $p' \subset p$, if $v \leq u$ and there exists a strictly and

Table 1 Notation table

Notation	Description
t, ts, te	Time point
l	Location
s	GPS trajectory sequence
P_{tra}	GPS trajectory path
D	GPS trajectory database
n, d	Node, destination node (road intersection)
p	Navigation path
δ	Support threshold
tt	Estimative travel time
r	Travel time sequence
T	Travel time table
QD	A set of query destination nodes

consecutively increasing sequence $(k_1, k_2, \ldots, k_v)$, where $k_{i+1} = k_i + 1, \forall 1 \leq i \leq v-1$, such that for all $j = 1, 2, \ldots, v, n'_j = n_{k_j}$. Here, p is called the *super-path* of p'. For example, let $p_1 = \langle A, B, C, D\rangle$ and $p_2 = \langle B, C\rangle$ be two navigation paths, p_2 is a sub-path of p_1 (denoted as $p_2 \subset p_1$).

Definition 3 Given a *GPS trajectory database* $D = \{s_1, s_2, \ldots, s_z\}$ that contains z GPS trajectory sequences and the corresponding navigation paths of D is $P = \{p_1, p_2, \ldots, p_z\}$, the support (sup) of the GPS navigation path p $\sup(p)$ is defined as (1).

$$\sup(p) = \frac{|\{p_i \in P | p \subset p_i, 1 \leq i \leq z\}|}{z} \tag{1}$$

Definition 4 A navigation path $p = \langle n_1, n_2, \ldots, n_u\rangle$ is called a *frequent navigation path* if $\sup(p)$ is greater than or equal to a specified support threshold δ.

Definition 5 A navigation path $p = \langle n_1, n_2, \ldots, n_u\rangle$ is called a *popular navigation path* if p is a frequent navigation path and there is no frequent navigation path p' such that $p \subset p'$. For example, let $p_1 = \langle A, B, C, D\rangle$ and $p_2 = \langle B, C\rangle$ be two frequent navigation paths, p_1 is a popular navigation path but p_2 is not, since $p_2 \subset p_1$.

Definition 6 Let $[ts, te]$ be a *time segment* during the time period from time point ts to te.

Definition 7 Let $r = \langle n_s \rightarrow n_d, [ts_1, te_1] : tt_1, [ts_2, te_2] : tt_2, \ldots, [ts_w, te_w] : tt_w\rangle$ be a *travel time sequence* of a road segment with length equal to w, where $n_s \rightarrow n_d$ indicates the road segment from node n_s to n_d and $[ts_i, te_i] : tt_i$ indicates the estimative travel time during the starting time segment $[ts_i, te_i], \forall 1 \leq i \leq w$. Besides, $te_i = ts_{i+1} - 1, \forall 1 \leq i \leq w-1$. For example, let $r_1 = \langle A \rightarrow B, [0, 5] : 7, [6, 30] : 10, [31, 100] : 9\rangle$ be a travel time sequence of a road segment from node A to B. r_1 contains three time segments, i.e., [0, 5], [6, 30], and [31, 100], and the corresponding estimative travel time are 7, 10, and 9, respectively.

Definition 8 Let $T = \{r_1, r_2, \ldots, r_x\}$ be a *travel time table* of all road segments that contains x travel time sequences.

Definition 9 Let $\langle n'_s, n'_d, t'_s\rangle$ be a road segment, which is directly connected from node n'_s to n'_d and the starting time point at node n'_s is t'_s. The estimative travel time (ett) of $\langle n'_s, n'_d, t'_s\rangle$, to

be extracted from the travel time table T, is defined as (2), where r^* indicates the travel time sequence whose road segment is from node n'_s to n'_d. For example, let $r_1 = \langle A \rightarrow B, [0, 5] : 7, [6, 30] : 10, [31, 100] : 9\rangle$ be a travel time sequence. $\text{ett}(A, B, 25, T) = 10$, since the starting time point 25 is between the second time segment, i.e., [6, 30], the corresponding estimative travel time is 10.

$$\begin{aligned} &\text{ett}\left(n'_s, n'_d, t'_s, T\right) = r^* \cdot tt_i \\ &\quad \text{where } r^* \cdot n_s = n'_s, r^* \cdot n_d = n'_d, r^* \cdot ts_i \leq t'_s \leq r^* \cdot te_i, \text{ and } r^* \in T \end{aligned} \tag{2}$$

Definition 10 Let $p = \langle n_1, n_2, \ldots, n_u\rangle$ be a navigation path produced by navigation systems with length equal to u and the starting time point at node n_1 is t_s, the estimative travel time (ETT) of navigation path p from the travel time table T is defined a recursive function as (3).

$$\begin{aligned} &\text{ETT}\left(p, t_s, T\right) = \text{ETT}\left(p', t_s, T\right) + \text{ett}\left(n_{u-1}, n_u, \text{ETT}\left(p', t_s, T\right) + t_s, T\right) \\ &\quad \text{where } p' = p - \langle n_u\rangle \text{ and } \text{ETT}\left(\langle l_1\rangle, t_s, T\right) = 0 \end{aligned} \tag{3}$$

For example, let $p = \langle A, B, C\rangle$ be a navigation path and the starting time point at A is 25. $\text{ETT}(\langle A, B, C\rangle, 25, T) = \text{ETT}(\langle A, B\rangle, 25, T) + \text{ett}(B, C, \text{ETT}(\langle A, B\rangle, 25, T) + 25, T)$, in which $\text{ETT}(\langle A, B\rangle, 25, T) = \text{ETT}(\langle A\rangle, 25, T) + \text{ett}(A, B, \text{ETT}(\langle A\rangle, 25, T) + 25, T)$, where $\text{ETT}(\langle A\rangle, 25, T) = 0$, hence $\text{ETT}(\langle A, B\rangle, 25, T) = \text{ett}(A, B, 25, T)$. Finally, $\text{ETT}(\langle A, B, C\rangle, 25, T) = \text{ett}(A, B, 25, T) + \text{ett}(B, C, \text{ett}(A, B, 25, T) + 25, T)$.

Definition 11 A navigation path $p = \langle n_1, n_2, \ldots, n_u\rangle$ is called the *fastest navigation path* from node n_1 to n_u and the starting time point at node n_1 is t_s if there is no navigation path $p' = \langle n'_1, n'_2, \ldots, n'_v\rangle$ such that $\text{ETT}(p', t_s, T) < \text{ETT}(p, t_s, T)$, where $n'_1 = n_1, n'_v = n_u$, and T indicates the travel time table.

Definition 12 Let $E = \{e_1, e_2, \ldots, e_g\}$ be a set of elements with amount equal to g, the *permutation set* $P(E)$ of E is defined as (4). For example, let $E = \{A, B, C\}$ be a set of elements. $P(E) = \{\langle A, B, C\rangle, \langle A, C, B\rangle, \langle B, A, C\rangle, \langle B, C, A\rangle, \langle C, A, B\rangle, \langle C, B, A\rangle\}$.

$$\begin{aligned} P(E) = &\left\{\langle e'_1, e'_2, \ldots, e'_g\rangle | \exists f : e_k \leftrightarrow e'_h\right\} \\ &\text{where } f \text{ is a bijection function and } 1 \leq k \leq g, 1 \leq h \leq g \end{aligned} \tag{4}$$

Definition 13 Let $QD = \langle d_1, d_2, \ldots, d_q\rangle$ be a set of query destination nodes and the starting node is n_s. The starting time point at node n_s is t_s and the travel time table is T. The optimal permutation OP of QD is defined the as (5), which means that to find the permutation with minimal estimative travel time from the permutation set $P(QD)$.

$$OP\left(n_s, QD, t_s, T\right) = \arg_{p \in P(QD)} \min\left(\text{ETT}\left(n_s + p, t_s, T\right)\right) \tag{5}$$

With the above definitions, the research problem we are targeting in this paper is formulated as follows.

Problem Formulation. Given a GPS trajectory database D containing a large number of GPS trajectory sequences of mobile users, our goal is to develop a solution, which takes a set of query destinations QD as the input, to discover the fastest navigation path with the multiple destinations QD. In this research, we propose TPF framework to solve this problem.

4 Proposed method

In this section, we describe our system framework, namely, Trajectory-based Path Finding (TPF), for a location-based service that finds the fastest navigation path with multiple destinations. In this framework, three important research issues need to be addressed: (1) travel time evaluation for all road segments; (2) popular navigation path mining and maintenance; and (3) fastest navigation path planning. Correspondingly, we first propose a Mining-based Algorithm for Travel time Evaluation (MATE) to estimate the travel time of all the road segments in the network. Next, we mine the popular navigation paths with a sequential pattern mining algorithm and maintain the discovered popular paths in the proposed Efficient Navigation Path Search Tree (ENS-Tree). The ENS-Tree facilitates efficient retrieval of the possible popular navigation paths. With MATE and ENS-tree, an efficient fastest path finding algorithm for single destination is derived. Finally, we propose the Cluster-Based Approximation Strategy (CBAS) to find the fastest visiting order for specified multiple destinations.

4.1 The TPF system framework

The design of proposed system framework TPF for the targeted location-based service is illustrated in Fig. 1. The framework consists of three modules, including a road network database, a data mining mechanism, and a path planning engine. In this design, in addition to the road network database that maintains the detailed road connectivity and related information, our system has an "offline" mechanism for travel knowledge discovery and an "online" mechanism for path planning. In the offline mechanism, the main function is to precisely discover the traffic knowledge which is used to predict the future traffic trends. When mobile users move within the road network, the movement information which includes time and location will be detected by the GPS devices and be stored in the GPS trajectory database. There are a lot of valuable information in this database, such as travel time costs of road segments and movement behaviors of mobile users. Therefore, in the data mining mechanism, we design two mining techniques to discover the information, respectively. First, we propose MATE algorithm to discover the travel time table which is used to evaluate the

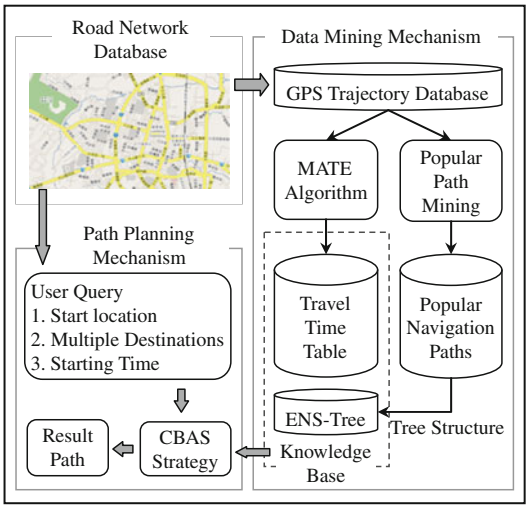

Fig. 1 System framework TPF

future traffic trends. Second, we use an existing sequential pattern mining method to mine the popular navigation paths as the candidate paths which are maintained by the proposed ENS-Tree structure. This design can help the system to improve the performance of fastest path planning. In the online planning engine, the main function is to efficiently find the fastest path with single or multiple destinations to users. To process these queries, we propose CBAS strategy which is based on divide-and-conquer. When a mobile user specifies single or multiple destinations to the system, the fastest navigation path contains these destinations will be returned to the mobile user by CBAS strategy according to the mobile user's current starting location and time point. As mentioned earlier, the proposed system framework aims to support path planning with multiple destinations efficiently and effectively.

4.2 GPS trajectory database

Before introducing the proposed methods, here we first discuss how a GPS trajectory can be obtained and the processing needed to transform a trajectory into a "navigation paths". A lot of GPS devices can be used to collect GPS data [33,43,44]. Using these devices, location points can be received every pre-setting timestamps. Figure 2a shows a GPS log that is a collection of location points $P = \{p_1, p_2, \ldots, p_n\}$. Each location point $p_i \in P$ contains latitude ($p_i.Lat$), longitude ($p_i.Lngt$), and timestamp ($p_i.T$). Note that the collected trajectories, without considering characteristics of physical roads, cannot be used directly for path planning and thus need to be transformed into navigation paths corresponding to roads. Our approach is to present the navigation paths in terms of road intersections, the coordinates of which can be easily obtained. Figure 2b shows a log of intersection nodes $N = \{n_1, n_2, \ldots, n_p\}$ for the road network shown in Fig. 2c. Each node $n_i \in N$ contains identification ($n_i.ID$), latitude ($n_i.Lat$), and longitude ($n_i.Lngt$). As shown in Fig. 2c, the GPS location points form a GPS trajectory sequence based on their time sequences. If the time interval between consecutive GPS location points exceeds a predefined threshold, this trajectory can be divided into several trajectories. A GPS trajectory can be transformed into a navigation path by location matching between this trajectory and the intersection nodes. When any location point in the trajectory is very close to any node in the network, the identification of this node is assigned to this location point. Therefore, all trajectories can be transformed to navigation paths after finishing the matching procedures. For example,

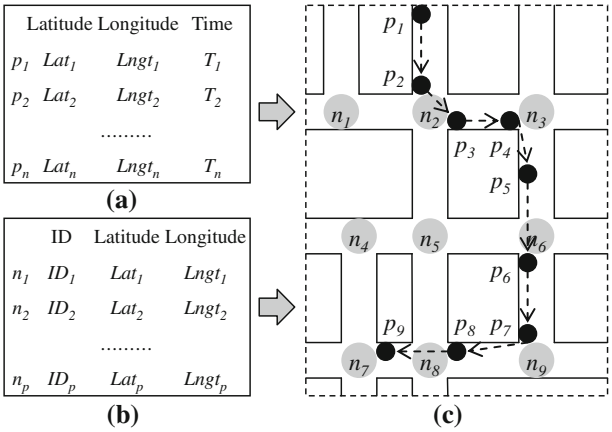

Fig. 2 The generation of GPS navigation database. **a** A GPS log; **b** a node log; **c** a GPS trajectory

Table 2 An example of navigation sequence database

Sequence ID	Navigation sequence
1	(8, *A*), (21, *B*), (30, *C*), (48, *N*), (68, *Z*), (90, *E*), (105, F)
2	(5, *J*), (15, *B*), (27, *C*), (60, *G*), (80, *H*), (85, *N*)
3	(9, *H*), (21, *N*), (29, *Z*), (37, *E*), (51, *F*), (62, *J*)
4	(1, *B*), (9, *C*), (18, *G*), (26, *K*), (45, *N*), (60, *Z*)

the GPS trajectory sequence in Fig. 2c is $\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}$. After location matching, the corresponding navigation path is $\{n_2, n_3, n_6, n_9, n_8, n_7\}$.

Table 2 shows an example of navigation sequence database which contains four records. In the database, a record represents a navigation path of a mobile user. For example, the sequence 1(8, *A*), (21, *B*), (23, *C*), (48, *N*), (68, *Z*), (90, *E*), (105, *F*) in Table 2 represents the navigation sequence which records the movement of the mobile user from node *A* to node *F* during time period 8–105. At the beginning, the mobile user was at node *A* at time point 8, and then he arrived at node *B* at time point 21. Therefore, we can obtain that the travel time from node *A* to node *B* is 13 by analyzing the navigation sequence database when the starting time point is 8.

4.3 MATE algorithm

With the availability of the navigation sequence database, the next task we have to tackle is the estimation of traffic cost for road segments in the road network. One simple approach to address this issue, called *Basic,* is to derive the average velocities for every road segment in the network for every time point. However, this solution may require excessive memory space because there exists a large number of possible time points. Noticing that the traffic conditions on road segments usually exhibit incremental changing patterns, it is a good idea to divide the time into time segments. To do so, a time segmentation strategy is to evenly divide the time line into a pre-determined number of time segments and estimate the travel time for each time segment. This approach, called *Even Segmentation (EvenSeg)*, needs to decide in advance the number of time segments which is difficult to decide. Therefore, in this paper, we proposed a dynamic time segmentation technique, called MATE, to estimate the travel time of road segments, which are stored as a travel time table for the network.

MATE algorithm consists of two main tasks: (1) Time segmentation for every road segment and (2) Discovery of travel time table. Even though the navigation sequence database may have several navigation sequences with the same starting node and destination node, their starting time points may be different. Take Table 2 as an example, the travel time costs of the road segment BC which are calculated based on Sequence 4, Sequence 2, and Sequence 1 are 8, 12, and 9 at starting time 1, 15, and 21, respectively. In other words, the travel time costs of the road segments vary at the different starting time points. The reason is that the traffic conditions on a road segment are usually different at different time points. In this subsection, we segment the starting time dimension into some time segments for each road segment. The idea behind our proposed time segmentation approach is to make the travel time costs are similar to each other in a time segment. The number of time segments and their ranges for road segments are automatically obtained by analyzing the navigation sequence database.

In the proposed time segmentation method, we first discuss the travel time distribution of a road segment. Figure 3 shows the average travel time cost of the road segment from node *A* to node *B* at the different starting time points. The horizontal axis represents the

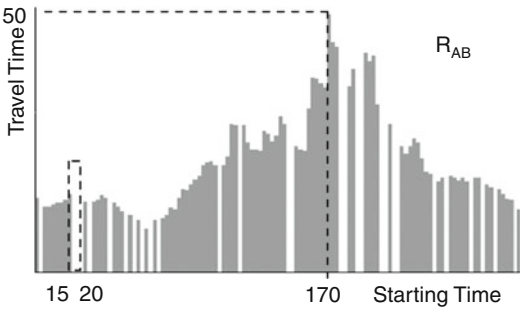

Fig. 3 The average travel time distribution of road segment AB

01 **Input:** The travel time cost $C = \{c_1, c_2, \ldots, c_M\}$ of a road and α
02 **Output:** The number of travel time levels N_{TL}
03 **GetNTL** (C, α)
04 $C' \leftarrow \{c \in C, c \neq \text{Empty}\}$
05 $\bar{c} \leftarrow \frac{1}{|C'|} \times \sum_{\forall c' \in C'} c'$
06 $N_{TL} \leftarrow \sqrt{\frac{1}{|C'|} \times \sum_{\forall c' \in C'} \left(c' - \bar{c}\right)^2} \times \alpha$
07 Return N_{TL}

Fig. 4 The GetNTL algorithm

starting time point from node A, and the vertical axis represents the average travel time cost of road segment AB. The dotted line represents the average travel time of road segment AB is 150 at starting time 170. The average travel time cost for road segment AB is analyzed from navigation sequence database by collecting and averaging the travel time costs of road segment AB with the same starting time. A point particularly worth noting is that the travel time cost is empty between starting time segment [15,20], indicating no historical data is recorded from node A to node B at these starting time points.

After obtaining all travel time distributions for all road segments, the next step is to decide the number of *travel time level*s N_{TL} which is used to classify the travel time. We argue that more travel time levels are needed to provide better precision when the travel time distribution shows a large variation. Therefore, the number of travel time levels N_{TL} is defined as (6), where C' indicates the set of non-empty travel time costs and N indicates the cardinality of set C', and α is a control parameter ranging from 0 to 1. The number of travel time levels increases when α increases. In our experiments, α is set as 0.3. Figure 4 shows the procedures for GetNTL (Get the Number of Time Levels) algorithm. Inputs include the travel time cost of a road segment and a control factor (line 01), where c_i indicates the travel time at starting time point $i, \forall 1 \leq i \leq M$. Output is the number of travel time levels (line 02). At first, non-empty travel time costs are collected into a set C' (line 04). The average travel time cost of C' is $\bar{c}$ (line 05). Finally, the number of travel time levels, N_{TL}, is returned. Figure 5 shows an example given that the number of travel time levels N_{TL} is obtained as 3.

$$N_{TL} = \sqrt{\frac{1}{N} \times \sum_{\forall c' \in C'} (c' - \bar{c})^2} \times \alpha \tag{6}$$

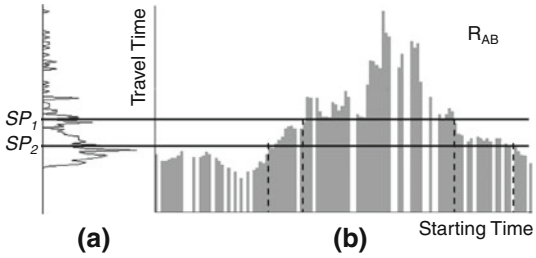

Fig. 5 **a** Time frequency distribution. **b** Result of time segmentation

```
01 Input: The travel time cost C = {c_1, c_2, ..., c_M} of a road and N_TL
02 Output: The segmenting positions SP
03 GetSP (C, N_TL)
04     SP ← ∅
05     N_V ← | {c ∈ C | c ≠ Empty} |
06     C' = Sort(C) /* Descendent sorting */
07     SP ← {c_i ∈ C' | i mod (N_V ÷ (N_TL+1)) = 0 and c_i ≠ Empty}
08     Return SP
```

Fig. 6 The GetSP algorithm

After calculating the number of travel time levels N_{TL}, the next step is to segment the starting time dimension into a number of starting time segments. Our idea is to make the segmentation based on cumulative frequencies of the travel time. Take Fig. 5 as an example, where N_{TL} is 3. Figure 5a shows the frequency distribution of travel time $f(t) = |\{r_i | r_i = t, \forall 1 \leqq i \leqq M\}|$ which is the number of travel time values are t, where r_i indicates the travel time at starting time point i and M indicates the total number of time points. We can obtain two segmenting positions $(SP)SP_1$ and SP_2 of travel time frequency distribution by equally dividing the total area A into three sub-areas, where $A = \Sigma f(t)$ for every t from minimal travel time to maximal travel time. Figure 6 shows the procedures for GetSP (Get Segmenting Position) algorithm. Given the travel time cost of a road segment and the number of travel time levels, the algorithm generates the set of segmenting positions. For example, two horizontal lines in Fig. 5 represent two segmenting positions of travel time distribution.

Next, the starting time segmentation is performed using the segmenting positions of travel time frequency distribution. Figure 7 shows the procedures for GetTS (Get Time Segment) algorithm. Input data includes the travel time cost of a road segment and a set of segmenting positions (line 01). Output data is the point set of time segments (line 02). At first, all non-empty travel time costs in C are classified to a group set G according to the segmenting points SP, respectively (line 04–line 14). When two neighbored elements in G are not the same, a time segment point is generated immediately (line 16–line 20). Finally, the GetTS algorithm returns all of the starting time segments (line 21–line 22). Take Fig. 5b as an example, the vertical dotted lines represent the segmenting positions of starting time dimension. When the segmenting positions of travel time frequency distribution cross through the travel time at any starting time point, a starting time segment is generated. There exists 4 cross points. We can obtain five starting time segments.

Finally, with the proper segmentation along the starting time, the travel time costs in the corresponding starting time segments can be obtained by average. Figure 8 shows the GetTTS (Get Travel Time Sequence) algorithm. Input data include the travel time cost of a road segment and the corresponding set of starting time segmenting points (line 01). Output data is the travel time sequence of the road segment (line 02). For each starting time segment of the road

```
01 Input: The travel time cost C = {c_1, c_2, ..., c_M} of a road and SP
02 Output: The points of time segments TSP
03 GetTS (C, SP)
04     C' ← {c_i ∈ C | c_i ≠ Empty}
05     L' ← {i | c_i ≠ Empty, ∀ c_i ∈ C} /* L' is the location of C' */
06     G ← ∅ /* G is the group of C' */
07     SP ← {0} ∪ SP
08     For i ← 1 to | C' |
09         For k ← | SP | to 1
10             If c'_i > SP_k
11                 G ← G ∪ {k}
12                 Break
13             End If
14         End For
15     End For
16     For i ← 1 to | G | - 1  /* | G | = | C' | */
17         If g_i ≠ g_{i+1}
18             TSP ← TSP ∪ L'_{i+1}
19         End If
20     End For
21     TSP ← {1} ∪ TSP ∪ {M}
22     Return TSP
```

Fig. 7 The GetTS algorithm

```
01 Input: The travel time cost C = {c_1, c_2, ..., c_M} of a road and the
   corresponding TSP
02 Output: The travel time sequence S of the road
03 GetTTS (C, TSP)
04     S ← <the start location of road → the destination of road>
05     For i ← 1 to | TSP | - 1
06         C' ← {c_j ∈ C | c_j ≠ Empty, TSP_i ≤ j ≤ TSP_{i+1}}
07         ett ← (Σ_{∀c'∈C'} c') / |C'|
08         S ← S contacts <[TSP_i, TSP_{i+1}]: ett> /* Sequence contact */
09     End For
10     Return S
```

Fig. 8 The GetTTS algorithm

segment (line 05 to line 09), the non-empty travel time costs are collected into a set C' (line 06). The estimated travel time in the time segment is calculated as the average travel time costs of C' (line 07). The travel time sequence of the road segment is obtained by integrating the starting location, the destination, and all of the estimated travel time costs (line 08). Figure 9 shows an example for the process of travel time evaluation on road segment AB. Time points $\{70, 95, 190, 225\}$ are time segmenting points, while 0 and 240 denote the starting and ending time of a day. The estimated travel time in each time segment is calculated by averaging all costs of travel time in the corresponding segment. The travel time sequence in this example is $\langle A \rightarrow B, [0, 70] : 45, [70, 95] : 56, [95, 190] : 84, [190, 225] : 54, [225, 240] : 48\rangle$. Finally, the travel time table is obtained by collecting all of the travel time sequence. The travel time table is used to evaluate the future travel time according to the starting time point.

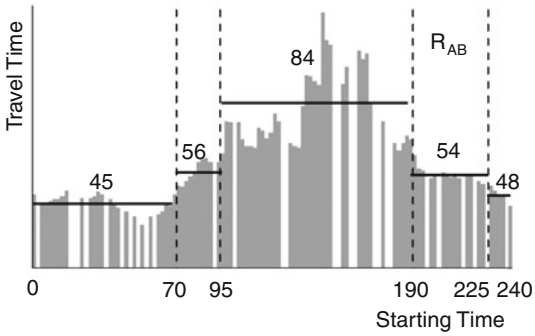

Fig. 9 An example of travel time evaluation process

Table 3 The travel time table

$A \rightarrow B$, [0–5]:7, [6–30]:10, [31–100]:9
$A \rightarrow D$, [0–35]:20, [36–60]:15, [61–100]:23
$B \rightarrow C$, [0–20]:15, [21–100]:18
$C \rightarrow A$, [0–10]:21, [11–40]:10, [41–100]:26
$C \rightarrow D$, [0–25]:30, [26–75]:40, [76–100]:25
⋮
$Y \rightarrow Z$, [0–7]:3, [8–51]:4, [52–100]:3

Table 3 shows an example of the travel time table. In this table, we observe that the starting time is divided into three time segments for the road segment AB and two time segments for the road segment BC. The travel time sequence, $A \rightarrow B$, [0–5]: 7, denotes that the travel time from node A to node B during starting time segment [0–5] is 7.

4.4 Mining popular navigation paths

After obtaining the travel time table, an estimated fastest path can be computed with conventional search algorithms such as breadth-first search (BFS). For example, by maintaining all possible paths between all pairs of nodes, a *Naïve* algorithm can find the fastest path by examining this candidate set. Although this Naïve approach may guarantee the fastest path always found, its search performance is very poor due to anticipated huge size of the candidate set. Therefore, we reduce the size of candidate set by only considering popular paths taken by many travelers. This strategy can efficiently reduce the memory cost and search time. On the contrary, the strategy does not guarantee a path satisfying the given starting node and the destination node to be found because the popular candidate set is only a part of candidate set. In the TPF, our remedy to this issue is to adopt BFS when path planning result is not returned for a given query.

Since the popular navigation path mining is an offline mechanism, the performance of mining algorithm is not critical. We adopt the AprioriAll algorithm, which was proposed by Agrawal et al. [2], to find the popular paths. Table 4 shows an example of the popular navigation paths which are discovered by the AprioriAll algorithm. Hence, we focus on reducing the memory cost for popular navigation path storage and retrieval. Here we propose a prefix-tree-based structure, named ENS-Tree, to store the popular navigation paths. There are two nice features provided by the proposed ENS-Tree structure: (1) compression of the duplicate navigation nodes and (2) efficient search of the navigation path. Figure 10 shows an example of the ENS-Tree structure which is constructed based on popular paths listed in

Table 4 The popular navigation paths

ID	The popular navigation path
1	ABCD
2	BCAD
3	BCE
4	CDE
5	CEF
6	DF

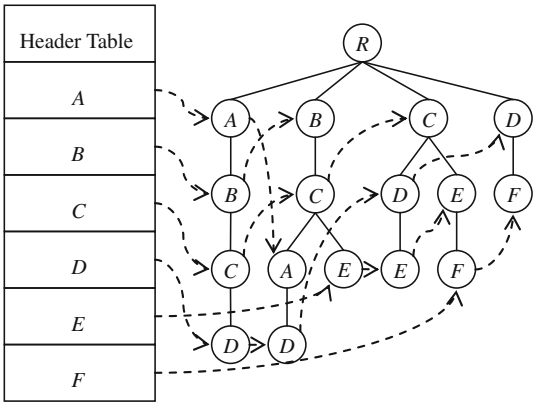

Fig. 10 An example of the ENS-Tree structure

Table 4. In the ENS-Tree, node R indicates the root. At first, the popular paths $ABCD$ and $BCAD$ are inserted under the root node. Moreover, the locations in these paths are linked by corresponding location nodes in the header table. When the path BCE is inserted into the tree, only the node E is generated under the path BC. Since the path BC has been stored under the root node, paths with a prefix BC can be compressed in the existing tree path. After inserting all the popular paths, the ENS-Tree is used to search for the fastest navigation path. Figure 11 shows the ENS-Tree construction algorithm.

4.5 Planning the fastest navigation path

To find the fastest path to the desired destinations, one or multiple destinations are specified in a search. For mobile user, we assume the start location and the starting time are automatically detected by the GPS devices. Accordingly, the candidate navigation paths that satisfy the starting location and the destination are first collected from the ENS-Tree. Then, the travel time costs of these paths are evaluated based on the travel time table and the starting time. Finally, the fastest one is returned to the users. In the following, we first discuss the path finding problem with single destination and then we present our strategy for finding path with multiple destinations.

Single Destination. The problem of single destination navigation path planning consists of two steps: (1) Candidate navigation paths retrieval from ENS-Tree and (2) Travel time estimation for candidate navigation paths. For the first step, the candidate paths satisfy starting and destination locations will be found from ENS-Tree. Figure 12 shows the PathFinding algorithm. At first, nodes in ENS-Tree whose label is the destination (called destination node) can be obtained by traversing the linked list in the header table corresponding to the targeted

```
01 Input: Popular navigation paths FP
02 Output: Efficient navigation path tree ENS-Tree
03 CreateTree (FP)
04     Create the root of ENS-Tree and root.child ← ∅
05     Header table HT ← {all of the unique symbols in FP}
06     For i ← 1 to FP.length
07         Temp ← root
08         For j ← 1 to FP_i .length
09             Create a new node N and N.id ← FP_i,j
10             If N.id ∈ Temp.child.id
11                 Temp ← c, ∃! c ∈ Temp.child | c.id = FP_i,j
12             Else
13                 Temp.child ← Temp.child ∪ N
14                 /*Get the corresponding label FP_i,j in header table*/
                   HTemp ← HT(FP_i,j)
15                 While HTemp.next ≠ null
16                     HTemp ← HTemp.next
17                 HTemp.next ← N
18                 Temp ← N
19             End If
20         End For
21     End For
22     Return ENS-Tree
```

Fig. 11 The ENS-Tree construction algorithm

```
01 Input: A start location S, a destination D, a starting time t_s,
          the ENS-Tree T, and the travel time evaluation table E.
02 Output: The fastest navigation path P
03 PathFinding (S, D, t_s, T, E)
04     CS ← ∅ /* The set of candidate navigation paths. */
05     /* Get the first corresponding node whose label is D in T. */
       nodeD ← T.HT(D).next /* T.HT indicates the header table of T. */
06     While nodeD.next ≠ null
07         cp ← nodeD.label /* Get the label of nodeD */
08         nodeS ← nodeD.parent
09         While nodeS ≠ root
10             cp ← nodeS.label contacts cp /*Contact the label of nodeS.*/
11             If nodeS.label = S and cp ∉ CS
12                 CS ← CS ∪ {cp} /* cp is one of the candidate paths. */
13             End If
14             nodeS ← nodeS.parent
15         End While
16         nodeD ← nodeD.next
17     End While
18     P ← ∅ /* The fastest path */
19     T_min ← ∞
20     For Each c ∈ CS
21         /* The function ETT is defined in the Definition 10 */
           If ETT(c, t_s) < T_min
22             P ← c
23             T_min ← ETT(P, t_s)
24         End If
25     End For Each
26     Return P
```

Fig. 12 The PathFinding algorithm

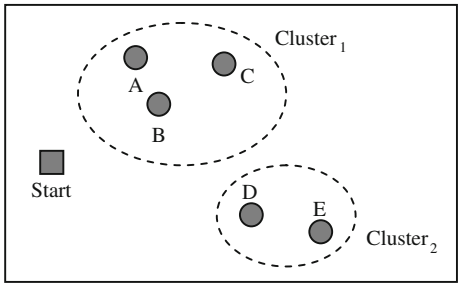

Fig. 13 A query scenario of multiple destinations

destination. Then, if a tree path from any destination node can be back-traced up to a node corresponding to the given starting location (called start node), the sub-path from the start node to the destination node are included in the candidate navigation path set (line 06–line 17). Note that, while the search process may traverse the ENS-tree from the start nodes down to the destination nodes, it has to search every sub-tree under every start node till all destination nodes are found. The search cost of the strategy is anticipated to incur a much higher cost than the back-tracking approach. Finally, the travel time costs for all candidate paths are evaluated using travel time table in accordance with the starting time (line 18–line 25). The fastest one is returned to the user (line 26).

For example, Table 3 shows the travel time table and Fig. 10 shows the ENS-Tree. A user requests the fastest path from B to D at starting time 15. At first, two paths BCD and $BCAD$ which satisfy the start and destination are obtained via the link D in header table. Then, we evaluate the travel time of these candidate paths. For the path BCD, the starting time at B is 15. The travel time from B to C at time point 15 is 15, thus, the arrived time at C is 30. The travel time from C to D at time point 30 is 40, thus, the arrived time at D is 70. That is to say the travel time of path BCD is 55 (subtract 15 from 70). Likewise, the travel time of path $BCAD$ is 40. Finally, the fastest path $BCAD$ is returned to the user, even though the distance of path BCD is shorter than the distance of path $BCAD$.

Multiple Destinations. In our daily life, planning of the fastest navigation path with multiple destinations is a desirable function. The problem of path finding with single destination can be generalized to multiple destinations. This is a combinatorial optimization problem, where the objective is to minimize the cost of combinatorial solution. After the order of destinations is decided, the cost can be evaluated by iterating the PathFinding algorithm on each combination. The most intuitive strategy, called *BruteForce*, is to evaluate all possible combinations of destinations and return the fastest one to users. Obviously, when the number of destinations increases, the computation cost of BruteForce increases exponentially. In an online navigation system, the response time of navigation path query is a critical requirement. The second strategy, called *Greedy*, is to select the nearest unvisited destination repeatedly. This approach is efficient in finding a destination order but only producing a local optimal solution. Therefore, the quality of this result may not be good.

In order to meet both the quality and efficiency requirements, we propose a novel algorithm, namely, *Cluster-Based Approximation Strategy (CBAS)*, to efficiently plan an approximately fastest path. CBAS takes the evaluated travel time between destination nodes into account. As Fig. 13 shows, there are five destinations A, B, C, D, and, E. We can cluster these destinations into two clusters in accordance with their evaluated travel time. When the user moves to destination A, CBAS only checks destination combinations in cluster$_1$, because destinations D and E are far away from destination A in terms of evaluated travel time. Obviously, visiting destination A followed by visiting destination D or E is not a high quality solution.

```
01 Input: A start location S, several destinations D = {d_1, d_2, …, d_n},
   a starting time t_s, the ENS-Tree T, and the travel time table E.
02 Output: The fastest navigation path P
03 CBAS (S, D, t_s, T, E)
04   Create a n by n similarity matrix M
05   For i ← 1 to n
06     For j ← 1 to n
07       M_ij ← 1 / ETT(PathFinding(d_i, d_j, t_s, T, E), t_s)
08     End For
09   End For
10   /* C = {c_1, c_2, …, c_m} be a partition of D */
     C ← Smart-CAST(M) /* Clustering the destinations */
11   P ← ∅ /* The fastest path */
12   While D ≠ ∅
13     d' ← arg_{d∈D}min(ETT(PathFinding(S, d, t_s, T, E), t_s))
       /* See Definition 10 */
14     P ← P ∪ PathFinding(S, d', t_s, T, E)
15     t_s ← t_s + ETT(PathFinding(S, d', t_s, T, E), t_s)
16     D ← D \ {d'}
17     S ← d'
18     c' ← The cluster c belongs to C and contains d'
19     O ← OP(S, c' \ {d'}, t_s, T) /* See Definition 13 */
20     For Each o ∈ O
21       P ← P ∪ PathFinding(S, o, t_s, T, E)
22       t_s ← t_s + ETT(PathFinding(S, o, t_s, T, E), t_s)
23       D ← D \ {o}
24       S ← o
25     End For Each
26   End While
27   Return P
```

Fig. 14 The CBAS algorithm

Therefore, we not only reduce computation costs but also achieve high quality in terms of destination order by integrating the strength of BruteForce strategy and Greedy strategy.

CBAS consists of three steps: (1) Destination clustering, (2) Local optimal search, and (3) Global path generation. Figure 14 shows the CBAS algorithm. When a mobile user wants to visit several destinations from the starting time point t_s, CBAS is employed to find the fastest path. In the first step, we use the non-parametric clustering algorithm Smart-CAST algorithm [42] to cluster the destinations into several destination clusters (line 04–line 10). Before executing the method Smart-CAST, we have to generate a similarity matrix M, based on the travel time costs. The entry M_{ij} in matrix M indicates the inverse of travel time cost which is evaluated by the single destination procedure between the destinations i and j in the road network, with the degrees in range of [0, 1] (line 04–line 09). After obtaining the similarity matrix M, the destination cluster can be built by the Smart-CAST method (line 10). In the second step, we first choose the destination node d', whose travel time is the minimum from the start location at the starting time point t_s, as the node to visit firstly (line 13–line 17). Then, the local optimal path for all destination nodes in the same cluster with the node d' would be generated by the BruteForce strategy (line 18–line 25). In the third step, the same procedure is repeated until all of the destinations have been visited (line 12–line 26). In this way, the fastest path solution is obtained. The corresponding fastest navigation path is returned to the mobile user (line 27). CBAS may suffer the problem of exponential time complexity when the cluster contains many destination nodes. To solve this problem, we can cluster again the cluster contains too many destination nodes based on the computation capability of mobile device.

Table 5 Major parameters of simulation model

Parameter	Description	Default Value
N_s	$N_s^* N_s$ road network	10 km
N_n	The number of nodes in the road network	100
N_e	The number of connected nodes for each node	3
λ	The load of traffic congestion	20
N_t	The length of time points	240
P_E	The event probability of user's behaviors	0.5
N_L	The size of GPS trajectory database	100 k
T_{avg}	Average travel time of GPS trajectory database	100
N_d	The number of destinations of a query	5
α	The control parameter	0.3

5 Experimental evaluation

We have conducted a series of experiments to evaluate the performance for the proposed MATE algorithm, ENS-Tree structure, and CBAS algorithm under various system conditions. Experiments can be divided into three parts, (1) precision of MATE algorithm, (2) performance of ENS-Tree structure, and (3) performance of CBAS strategy. All of the experiments were implemented in Java JDK 1.5 on an Intel Pentium 4 CPU 3.00 GHz machine with 1GB of memory running Microsoft Windows XP.

5.1 Simulation model

To evaluate the practicability of the TPF framework, we consider a real-time traffic information data in Kaohsiung city [26], which was collected by Kaohsiung government during the period of September 27, 2008 and May 27, 2009. This data contains a road map of Kaohsiung city and a traffic condition database. The area of Kaohsiung city is 153 km^2. The number of nodes (intersection) is 112 and the number of edges (roads) is 203. The Kaohsiung government sets up vehicle detectors on each road to detect how many cars passing by this road and to calculate their average speed on this road. The traffic conditions are reported to a central database server every 5 min. However, due to the lack of vehicle ID in this dataset, we simulate the vehicle trajectories based on the real traffic conditions collected in the dataset. Furthermore, to evaluate the impact of the TPF framework under various road network sizes and traffic conditions, we develop a simulation model based on [11] to generate the road network, the traffic condition, and the moving trajectory.

Table 5 lists the major parameters used in the simulation model with default settings. In the base experiment model, the network is modeled as a road network with size $N_s^* N_s$. There are N_n nodes in this network. The location including longitude and latitude of each node is determined based on uniform distribution within a given range N_s. All edges between two nodes are generated according to their distance. The shorter a distance between two nodes, the higher probability an edge is generated between them. For every node, the number of connected nodes is determined from a normal distribution with the mean equals to N_e. Edge attributes in the simulator include distance, maximum velocity constraint, least free-flow time, capacity, and scale. The time length of one day is divided into N_t timestamps, i.e., a timestamp is $1{,}440/N_t$ min.

The interarrival time of moving objects are used to control the traffic load on the network. The interarrival time represents the time between the arrival of a moving object and that of

the next moving object at a given starting node. The interarrival time is determined from an exponential distribution with mean equal to $1/\lambda$ which represents the severity of the network congestion [11]. The larger the mean of exponential distribution, the more congested the network. For one day, there are about 50,000 moving objects generated in the network under λ sets as 20. Each moving object is generated at a random start node and it may move by adhering to a certain moving strategy with probability P_E or randomly [41]. The default moving strategies include the shortest distance path strategy, the major road strategy, the least free-flow time strategy, and the direction strategy [34]. After finishing the simulation of traffic congestion, we can obtain the moving object distribution for each time slot in the network. As the number of moving objects in a road segment increases, the average speed of the moving objects decreases. Therefore, all the number of moving objects in a road segment from the moving object distribution can be transformed to the average speed according to the road capacity. Finally, the traffic congestions for everyday can be simulated.

To generate the navigation database, we simulate the user movement in the network. There are N_L users in this network. Each user is generated at a random start node and a random starting time point, and it may move by adhering to a certain moving strategy with probability P_E or randomly. The average travel time of each user is determined from a normal distribution with mean equal to T_{avg}. Each user randomly selects one or several destinations to travel. The number of destinations of a user's query is determined from a normal distribution with mean equal to N_d. Finally, we can obtain the navigation database contains N_L moving records for everyday.

In the following, we list the main measurements used in the experimental evaluation; (1) Mean Absolute Error (MAE) represents a quantity used to measure how close predictions are to the truths. (2) Travel Time represents the actual time cost of the recommended navigation path. (3) Travel Distance represents the distance cost of the recommended navigation path. (4) Memory represents the size of memory cost. (5) Latency represents the search time for finding and evaluating all the candidate paths which satisfy the start location and destination. (6) Execution Time represents the execution time cost for obtaining the combinatorial solution. For CBAS, the execution time includes the clustering of destinations.

For single destination experiments, we compare the proposed MATE method with another three path planning strategies: (1) the shortest distance path strategy (called *DIS*). The edge cost of this strategy is the road distance. (2) The least free-flow time path strategy (called *LFT*). The edge cost of this strategy is the road distance divided by the maximal velocity constraint. (3) The CAtegorized PiecewisE COnstant speeD (called *CapeCod*). The edge cost of this strategy is estimated by CapeCod pattern [25] which is discussed in Sect. 2. For multiple destination experiments, we compare the proposed CBAS with another two approaches: (1) *BruteForce*, which finds the best solution from all of the possible destination combinations. (2) *Greedy*, which always chooses the nearest location from the current location and repeats the same procedure until all of the destinations is visited. For the synthetic data generated by our data generator, 70% of the navigation data are used for training to obtain the travel time table and the popular navigation paths, and the rest 30% are for prediction.

5.2 Impact of the control parameter α

This experiment analyzes the MAE when the control parameter α of (6) is varied. As Fig. 15 shows, we observed that the MAE decreases by increasing α, because the number of time segments increases when α increases. The estimative travel time in each time segment is more precise if the time segment is smaller. Hence, the MAE is better when α increases. However, the memory storage cost of estimative travel time table is also increasing when the

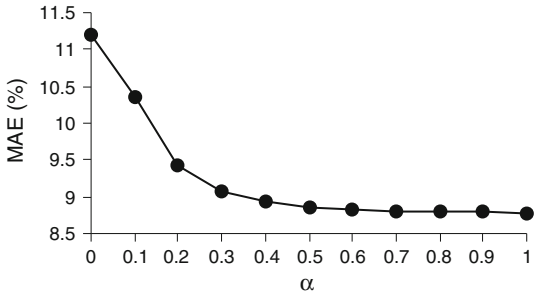

Fig. 15 MAE under various values of α

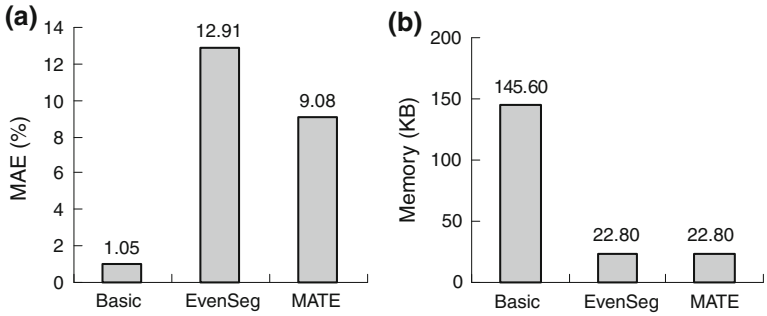

Fig. 16 MAE and memory storage under various strategies

number of time segments increases. A large α may bring the memory cost problem. In the following experiments, we choose 0.3 as the default value of α, since the MAEs are close to each other when α is larger than 0.3.

5.3 Comparison of various time segmentations

This experiment analyzes the MAE and memory storage under various time segmentation strategies. In this experiment, we first generate 25 various traffic logs and their corresponding navigation databases by the simulator. Then, we compare the error rate of estimated travel time costs and real traffic log under the three time segmentation strategies. In order to let the number of time segments of EvenSeg strategy be the same as MATE algorithm, we first execute MATE algorithm to dynamically obtain the number of time segments and then execute EvenSeg strategy using the same number. As Fig. 16a and b shows, we observed that (1) although the MAE of Basic strategy is the minimum, the memory cost is significant higher than segmentation-based strategies in terms of EvenSeg and MATE. This is because that the Basic strategy stores a large amount of estimated travel time information. Consequently, the estimated travel time is close to real travel time. However, the Basic strategy needs a large memory to store all the information that is a critical resource in mobile devices. (2) Under the same memory cost, the MAE of MATE outperforms that of EvenSeg, because MATE considers the variation of travel time. Therefore, MATE can obtain a more precise travel time table than the EvenSeg strategy. Overall, MATE algorithm was shown to have a better result.

5.4 Comparison of various traffic cost estimations

This experiment analyzes the travel time and travel distance under various traffic cost estimation strategies. Figure 17a and b represent the evaluated results under real dataset, i.e.,

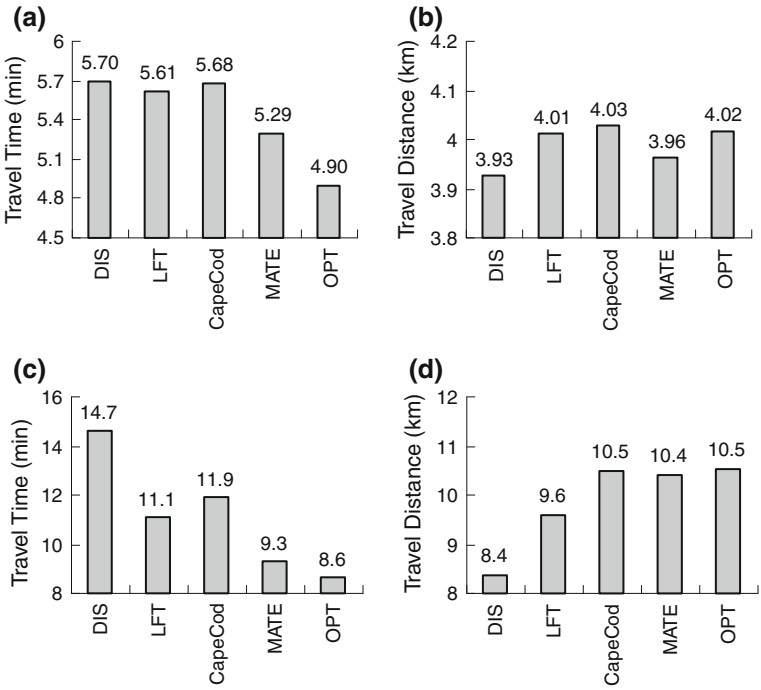

Fig. 17 Travel time and travel distance under various strategies

Kaohsiung city, and Fig. 17c and d represent the evaluated results under simulation dataset. In this experiment, we first randomly select 10,000 pairs of start location and destination. Then, we compare the actual travel time costs under four kinds of methods. The travel time of OPT represents the actual fastest travel time as the baseline solution. As the experimental result shows, we observed that (1) although the travel distance of DIS strategy is the minimum, the shortest path is not the fastest path. This is because of the traffic congestions in the network are always different. (2) The LFT strategy outperforms the DIS strategy, because the LFT strategy considers not only the distance information but also the maximal velocity constraints of road segments. The LFT strategy can obtain the fastest path, if there is no vehicle object in the network. However, in a real road network, the vehicle flows in the network are not always very low. (3) The travel time estimated by MATE is more precise than that by CapeCod since the planned travel time of MATE is better than that of CapeCod. This is because that CapeCod does not consider dynamic time segmentation, the estimated travel time in each time segment may not be correct, but MATE does. (4) The travel time of navigation path obtained by MATE algorithm has the best result because the future traffic trends are precisely predicted by MATE algorithm to search the estimative fastest path. Overall, the MATE algorithm was shown to have excellent result.

5.5 Comparison of various storage structures

This experiment analyzes the latency and memory storage under various storage structures for popular paths. In this experiment, we compare the proposed ENS-Tree structure to three kinds of structures, i.e., Naïve, Popular, and NPST, respectively. Naïve structure stores all

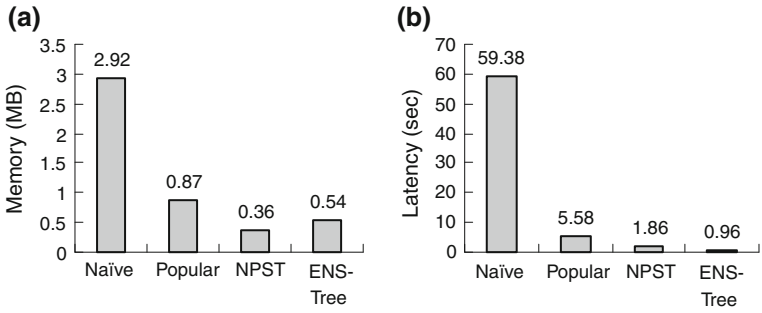

Fig. 18 Latency and memory storage under various strategies

possible navigation paths for all pairs of nodes using an array structure. Popular structure stores all popular navigation paths using an array structure. NPST structure stores all popular navigation paths using the Prefix-Tree structure was proposed in [34]. As Fig. 18a and b shows, we observed that (1) the latency and memory storage are significantly improved by the methods based on popular path mining, i.e., Popular, NPST, and ENS-Tree, because these strategies reduce the size of candidate navigation paths. (2) The latency and memory storage are improved by tree-based structures, i.e., NPST and ENS-Tree, because the paths whose prefix paths are the same in the tree structure can be compressed into a path. Therefore, not only the memory storage but also search efficiency can be improved. (3) Although the memory storage of ENS-Tree is slightly larger than that of NPST, the latency of ENS-Tree outperforms that of NPST. The reason is that ENS-Tree incurs an additional memory overhead for the header table and its link lists. The header table is a good tradeoff that helps the search procedure to improve the search efficiency. Overall, ENS-Tree has shown to have excellent search efficiency and low memory cost.

5.6 Comparison of various event probabilities P_E

This experiment analyzes the travel time and execution time when the event probability varies. In this experiment, the travel time of OPT represents the actual fastest travel time as the optimal baseline solution. As Fig. 19a,b shows, CBAS outperforms Greedy in terms of travel time. Although the travel time of CBAS can not achieve that of BruteForce, the execution time of CBAS is significantly faster than that of BruteForce. We observe that both of the travel time and execution time decrease with the increase in event probability. This is because, by increasing the event probability, the movement behaviors of mobile users in the road network are more regular. In other words, the traffic prediction is becoming more precise. Therefore, the travel time shows a decreasing trend.

5.7 Comparison of various traffic congestions λ

This experiment analyzes the travel time and execution time when the traffic congestion parameter lambda varies. In this experiment, the travel time of OPT represents the actual fastest travel time as the optimal baseline solution. As Fig. 20a and b shows, CBAS outperforms Greedy in terms of travel time. Although the travel time of CBAS can not achieve that of BruteForce, the execution time of CBAS is significantly faster than that of BruteForce. We observe that the travel time increases with the decrease in the traffic congestion λ. This is because that the traffic condition is crowded when the value of traffic congestion λ is low.

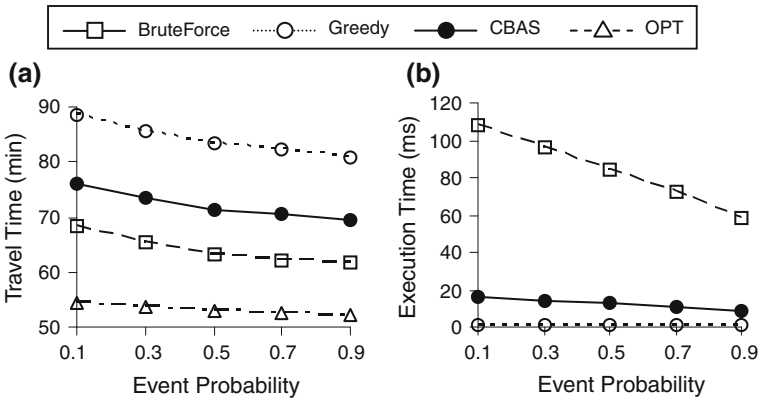

Fig. 19 Travel time and execution time with event probability varied

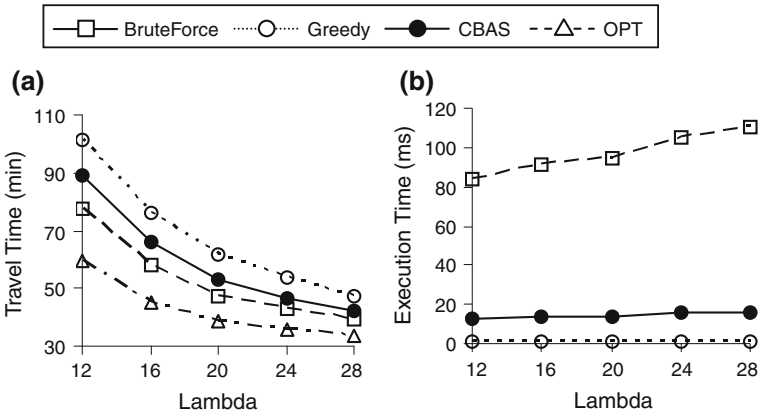

Fig. 20 Travel time and execution time with traffic congestion varied

Table 6 Settings of network scales

Network scale	1	2	3	4	5
Size (km)	6 × 6	8 × 8	10 × 10	12 × 12	14 × 14
# Nodes	60	80	100	120	140
# Edges	120	160	200	240	280

The average travel time increases when the traffic congestion increases. Therefore, the travel time shows a decreasing trend.

5.8 Comparison of various network scales

This experiment analyzes the travel time and execution time when the network scale varies. Table 6 shows the settings of network size, number of nodes, and number of edges in various road network scales. In this experiment, the travel time of OPT represents the actual fastest travel time as the optimal baseline solution. As Fig. 21a and b shows, CBAS outperforms Greedy in terms of travel time. Although the travel time of CBAS can not achieve that of

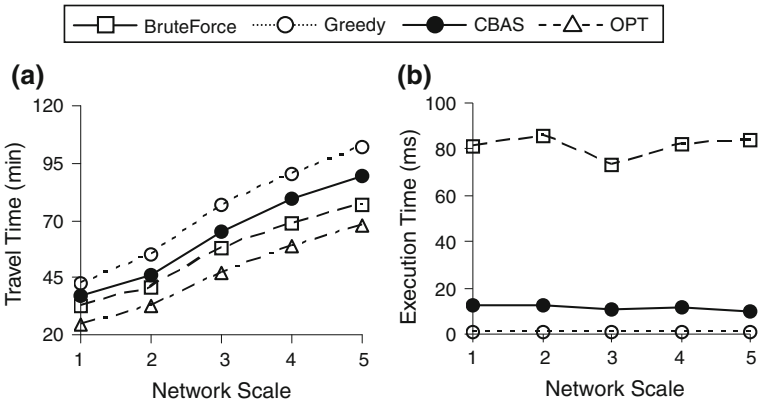

Fig. 21 Travel time and execution time with network scale varied

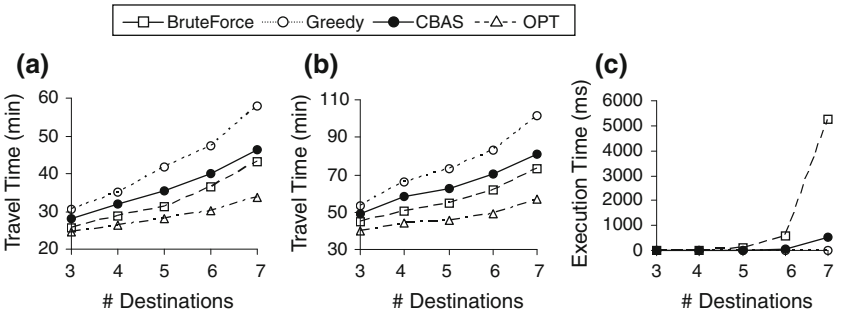

Fig. 22 Travel time and execution time with #destinations varied

BruteForce, the execution time of CBAS is significantly faster than that of BruteForce. We observe that the travel time increases by enlarging the network scale. The reason is that when the network scale enlarges, the number of nodes and the distance between two nodes also increase. Therefore, the average travel time is increasing with the increase in network scale.

5.9 Comparison of various number of destinations

This experiment analyzes the travel time and execution time when the number of destinations varies. Figure 22a represents the evaluated results under real dataset, i.e., Kaohsiung city, and Fig. 22b and c represent the evaluated results under simulation dataset. In this experiment, the travel time of OPT represents the actual fastest travel time as the optimal baseline solution. As the experimental result shows, CBAS outperforms Greedy in terms of travel time. Although the travel time of CBAS cannot achieve that of BruteForce, the execution time of CBAS is significantly faster than that of BruteForce. We observe that the travel time and execution time significantly increase as the number of destinations increase. The reason is that when the number of destinations increases, the complexity of combinatorial discovery also increase exponentially. Therefore, the average travel time increases along with the number of destinations. The execution time shows the same trend.

6 Conclusions and future work

In this paper, we have developed a new system framework, called Trajectory-based Path Finding (TPF), based on a data mining approach, for finding the fastest navigation path with multiple destinations. In TPF, a novel data mining algorithm, namely Mining-based Algorithm for Travel time Evaluation (MATE), is proposed for estimating the travel time of a navigation path. Besides, a novel index structure, called Efficient Navigation Path Search Tree (ENS-Tree), is proposed for efficient retrieval of the fastest navigation path. With MATE and ENS-tree, an efficient fastest path finding algorithm for single destination is derived. To find the fastest path for multiple destinations, we have proposed a novel method, known as Cluster-Based Approximation Strategy (CBAS), for efficiently and precisely discovering the fastest navigation path with multiple destinations. Although a number of studies exist in the literature to explore various approaches for navigation path planning, few consider the issue of path finding with multiple destinations using data mining techniques to analyze the trajectories. To the best knowledge of the authors, this is the first work on fastest path planning for multiple destinations based on trajectory mining.

To evaluate the performance of the proposed framework TPF, we collect a real dataset and design a simulation model to conduct a series of experiments, which can be classified as follows: (1) precision of MATE algorithm, (2) performance of ENS-Tree structure, and (3) performance of CBAS algorithm. For the experiments for MATE algorithm, the results show that the proposed MATE achieves high quality planning results in terms of travel time. For the experiments for ENS-Tree structure, we show that our proposed ENS-Tree structure is very efficient. It not only reduces the memory cost but also improves the search performance. For performance of multiple destinations planning, we observed that CBAS outperforms Greedy in terms of travel time. Although the travel time of CBAS does not match up with that of BruteForce, the execution time of CBAS is significantly faster than that of BruteForce. The experimental results demonstrate that our proposed methods are efficient and accurate.

As for the future work, we plan to collect real data for further analysis and testing of the proposed TPF. In addition, we plan to extend the ideas in TPF to other applications such as public vehicle scheduling, aiming to enhance the quality of new applications in road networks.

Acknowledgments This research was supported by National Science Council, Taiwan, R.O.C. under grant no. NSC 96-2221-E-006-143-MY3.

References

1. Agrawal R, Imielinski T, Swami A (1993) Mining association rule between sets of items in large databases. In: Proceedings of ACM SIGMOD conference on management of data, pp 207–216
2. Agrawal R, Srikant R (1995) Mining sequential patterns. In: Proceedings of international conference on data engineering, pp 3–14
3. Arora S (1998) Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J ACM 45:753–782
4. Awasthi A, Lechevallier Y, Parent M et al (2005) Rule based prediction of fastest paths on urban networks. In: Proceedings of 8th IEEE conference on intelligent transportation systems, pp 978–983
5. Barai SK (2003) Data mining applications in transportation engineering. Transport 18(5):216–223
6. Bekhor S, Ben-Akiva ME, Ramming MS (2006) Evaluation of choice set generation algorithms for route choice models. Ann Oper Res 144(1):235–247
7. Ben-Dor A, Yakhini Z (1999) Clustering gene expression patterns. J Comput Biol 6(3):281–297
8. Borges J, Levene M (2000) Data mining of user navigation patterns. Lecture Notes in Computer Science 1836, pp 92–112

9. Chen MS, Park JS, Yu PS (1998) Efficient data mining for path traversal patterns. IEEE Trans Knowl Data Eng 10(2):209–221
10. Cheong CH, Wong MH (2006) Mining popular paths in a transportation database system with privacy protection. In: Proceedings of 22nd international conference on data engineering workshops, p 122
11. Chon HD, Agrawal D, El Abbadi A (2003) FATES: Finding a time dependent shortest path. In: Proceedings of 4th international conference on mobile data management, pp 165–180
12. Denton AM, Besemann CA, Dorr DH (2009) Pattern-based time-series subsequence clustering using radial distribution functions. Knowl Inf Syst 18(1):1–27
13. Dijkstra EW (1959) A note on two problems in connection with graphs. Numer Math 1(1):269–271
14. El-Rabbany A (2006) Introduction to GPS: the global positioning system, 2nd edn. Artech House, Boston
15. Engineer F (2001) Fast shortest path algorithms for large road networks. In: Proceedings of 36th annual ORSNZ conference
16. Fritzsche PC, Rexachs D, Luque E (2007) A computational approach to TSP performance prediction using data mining. In: Proceedings of 21st international conference on advanced information networking and applications workshops, pp 252–259
17. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman, San Francisco
18. Gonzalez H, Han J, Li X et al (2007) Adaptive fastest path computation on a road network: a traffic mining approach. In: Proceedings of 33rd international conference on very large data bases, pp 794–805
19. Google Map, http://maps.google.com/
20. Halvey M, Keane T, Smyth B (2005) Predicting navigation patterns on the mobile-internet using time of the week. In: Proceedings of 14th international conference on World Wide Web, pp 958–959
21. Halvey M, Keane T, Smyth B (2006) Time based patterns in mobile-internet surfing. In: Proceedings of SIGCHI conference on human factors in computing system, pp 31–34
22. Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern 4(2):100–107
23. Johnson DS, McGeoch LA (1995) The traveling salesman problem: a case study in local optimization. In: Aarts EHL, Lenstra JKLocal search in combinatorial optimization. Wiley, New York
24. Jula H, Dessouky M, Ioannou PA (2008) Real-time estimation of travel times along the arcs and arrival times at the nodes of dynamic stochastic networks. IEEE Trans Intell Transp Syst 9(1)
25. Kanoulas E, Du Y, Xia T et al (2006) Finding fastest paths on a road network with speed patterns. In: Proceedings of 22nd international conference on data engineering, p 10
26. Kaohsiung City Real-Time Traffic Information, http://kctraffic.tbkc.gov.tw/link01.htm
27. Kaufman L, Rousseeuw PJ (1990) Finding groups in data: an introduction to cluster analysis. Wiley, New York
28. Khoshgozaran A, Khodaei A, Sharifzadeh M et al (2008) A hybrid aggregation and compression technique for road network databases. Knowl Inf Syst 17(3):265–286
29. Lassabe N, Berro A, Duthen Y (2007) Improvement of a shortest routes algorithm. In: Proceedings of 10th IEEE conference on intelligent transportation systems, pp 613–617
30. Lim Y, Kim H (2005) A shortest path algorithm for real road network based on path overlap. J East Asia Soc Transp Stud 6:1426–1438
31. Lin J, Keogh E, Lonardi S et al (2003) A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of 8th ACM SIGMOD workshop on research issues in data mining and knowledge discovery, pp 2–11
32. van Lint JWC (2008) Online learning solutions for freeway travel time prediction. IEEE Trans Intell Transp Syst 9(1):38–47
33. Lo C-H, Peng W-C, Chen C-W et al (2008) CarWeb: a traffic data collection platform. In: Proceedings of international conference on mobile data management, pp 221–222
34. Lu EH-C, Lin CC, Tseng VS (2008) Mining the shortest path within a travel time constraint in road network environments. In: Proceedings of IEEE conference on intelligent transportation systems, pp 593–598
35. Pallottino S, Scutella MG (1998) Shortest path algorithms in transportation models: classical and innovative aspects. In: Equilibrium and advanced transportation modelling, pp 245–281
36. Pepper J, Golden B, Wasil E (2002) Solving the traveling salesman problem with annealing-based heuristics: a computational study. IEEE Trans Man Cybern Syst A Syst Hum 32(1):72–77
37. Rosenkrantz DJ, Stearns RE, Lewis PMII (1977) An analysis of several heuristics for the traveling salesman problem. SIAM J Comput 6(3):563–581
38. Tsai C-Y, Chou S-Y, Lin S-W et al (2009) Location determination of mobile devices for an indoor WLAN application using a neural network. Knowl Inf Syst 20(1):81–93

39. Tsai H, Yang J, Kao C (2002) Solving traveling salesman problems by combining global and local search mechanisms. In: Proceedings of congress on evolutionary computation, pp 1290–1295
40. Tseng VS, Chen LJ (2002) An empirical study of the validity of gene expression clustering. In: Proceedings of international conference on mathematics and engineering techniques in medicine and biological sciences
41. Tseng VS, Lin KW (2006) Efficient mining and prediction of user behavior patterns in mobile web systems. Inf Softw Technol 48(6):357–369
42. Tseng VS, Kao C (2005) Efficiently mining gene expression data via a novel parameterless clustering method. IEEE/ACM Trans Comput Biol Bioinform 2(4):355–365
43. Ye Y, Zheng Y, Chen Y et al (2009) Mining individual life pattern based on location history. In: Proceedings of international conference on mobile data management systems, services and middleware, pp 1–10
44. Zheng Y, Zhang L, Xie X et al (2009) Mining interesting location and travel sequences from GPS trajectories. In: Proceedings of 18th international World Wide Web conference pp 791–800

Author Biographies

Eric Hsueh-Chan Lu received the B.S. degree in Computer Science and Information Engineering from National Taiwan University of Science and Technology (NTUST), Taiwan, ROC, in 2003, and the Ph.D. degree in Computer Science and Information Engineering from National Cheng Kung University (NCKU), Taiwan, ROC, in 2010. His research interests include data mining, mobile computing, object tracking, mobility behavior discovery with prediction, and intelligent transport systems.

Wang-Chien Lee received the B.S. degree from the National Chiao Tung University, Hsinchu, Taiwan, ROC, the M.S. degree from the Indiana University, Bloomington, and the Ph.D. degree from the Ohio State University, Columbus. He is an associate professor of computer science and engineering at Pennsylvania State University, University Park, where he also leads the Pervasive Data Access Research Group to perform cross-area research in database systems, pervasive/mobile computing, and networking. Most of his research results have been published in prestigious journals and conference proceedings in the fields of databases, mobile computing, and networking. He has served as a guest editor for several journal special issues on mobile database-related topics, including the IEEE Transactions on Computer, IEEE Personal Communications Magazine, ACM MONET, and ACM WINET. He was the founding program committee co-chair for the International Conference on Mobile Data Management. He is a member of the IEEE and the ACM.

Vincent S. Tseng is currently a professor at Department of Computer Science and Information Engineering at National Cheng Kung University (NCKU), Taiwan, ROC. Before this, he was a postdoctoral research fellow in Computer Science Division of University of California at Berkeley during January 1998 and July 1999. He has acted as the director for Institute of Medical Informatics of NCKU since August 2008. During February 2004 and July 2007, he had also served as the director for Informatics Center in NCKU Hospital. Dr. Tseng received his Ph.D. degree from National Chiao Tung University, Taiwan, ROC, in 1997. Dr. Tseng has a wide variety of research interests covering data mining, biomedical informatics, mobile computing and Web technologies. He is on the editorial board of International Journal of Data Mining and Bioinformatics and has also served as chairs/program committee for a number of premier international conferences related to data mining and databases.