An Efficient Destination Prediction Approach Based on Future Trajectory Prediction and Transition Matrix Optimization

Zhou Yang, Heli Sun, Jianbin Huang, Zhongbin Sun, Hui Xiong, *Senior Member, IEEE*, Shaojie Qiao, Ziyu Guan, and Xiaolin Jia

Abstract—Destination prediction is an essential task in various mobile applications and up to now many methods have been proposed. However, existing methods usually suffer from the problems of heavy computational burden, data sparsity, and low coverage. Therefore, a novel approach named DestPD is proposed to tackle the aforementioned problems. Differing from an earlier approach that only considers the starting and current location of a partial trip, DestPD first determines the most likely future location and then predicts the destination. It comprises two phases, the offline training and the online prediction. During the offline training, transition probabilities between two locations are obtained via Markov transition matrix multiplication. In order to improve the efficiency of matrix multiplication, we propose two data constructs, Efficient Transition Probability (ETP) and Transition Probabilities with Detours (TPD). They are capable of pinpointing the minimum amount of needed computation. During the online prediction, we design Obligatory Update Point (OUP) and Transition Affected Area (TAA) to accelerate the frequent update of ETP and TPD for recomputing the transition probabilities. Moreover, a new future trajectory prediction approach is devised. It captures the most recent movement based on a query trajectory. It consists of two components: similarity finding through Best Path Notation (BPN) and best node selection. Our novel BPN similarity finding scheme keeps track of the nodes that induces inefficiency and then finds similarity fast based on these nodes. It is particularly suitable for trajectories with overlapping segments. Finally, the destination is predicted by combining transition probabilities and the most probable future location through Bayesian reasoning. The DestPD method is proved to achieve one order of cut in both time and space complexity. Furthermore, the experimental results on real-world and synthetic datasets have shown that DestPD consistently surpasses the state-of-the-art methods in terms of both efficiency (approximately over 100 times faster) and accuracy.

Index Terms—Destination prediction, Markov model, matrix multiplication, dynamic programming

1 Introduction

The recent decade has witnessed the rise of ubiquitous smart devices that boast a multitude of sensors. Among them location related sensors play an essential role in enabling solutions concerning urban computing [1]. Specifically, geolocation related applications have long been a vibrant branch of extensive studies on mobile computing since its inception at the turn of this century. The ubiquity of hand-held smart gadgets has profoundly transformed our life on the go. Such devices are usually accompanied by the built-in feature of positioning enabled by GPS which sometimes works jointly with WIFI and RFID to enhance precision. Regarding the investigation of human movement, researchers have reaped immense benefit by clustering popular locations and analyzing historical trajectories generated by these devices.

Destination prediction is a popular theme of mobile computing and it is broadly employed in a variety of fields, such as mobile advertising, offering cognitive assistance, fastest path computation, mobile recommender system, human mobility study, popular routes discovery, driving experience enhancement etc. [2], [3], [4], [5], [6], [7], [8], [9]. In these applications, based on the starting location and the current location of a partial trip already traveled by a user, destination prediction methods seek to find out the user's destination of the whole journey.

Most earlier work employ Markov model and Bayesian inference to predict the final destination [8], [10], [11], [12], [13], [14], [15]. Some of them [10], [11], [12], [16], [17] incorporate information other than just historical trajectory data, such as geographic cover, user preference, best route and weather condition. However, such information may be very difficult to obtain in practice. Furthermore, some methods usually tend to underutilize the available

- *Z. Yang and X. Jia are with the Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China. E-mail: yangzhouxa@stu.xjtu.edu.cn, xlinjia@mail.xjtu.edu.cn.*
- *H. Sun is with the Department of Computer Science and Technology, the Shenzhen Research School, and the Shaanxi Province Key Laboratory of Computer Networks, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China. E-mail: hlsun@mail.xjtu.edu.cn.*
- *J. Huang is with the School of Computer Science and Technology, Xidian Univerisity, Xi'an, Shaanxi 710071, China. E-mail: jbhuang@xidian.edu.cn.*
- *Z. Sun is with the Department of Computer Science and Technology, and the Shenzhen Research School, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China. E-mail: zhongbin725@mail.xjtu.edu.cn.*
- *H. Xiong is with the Management Science and Information Systems Department, Rutgers, the State University of New Jersey, New Brunswick, NJ 08901-8554. E-mail: hxiong@rutgers.edu.*
- *S. Qiao is with the Chengdu University of Information Technology, Chengdu 610225, China. E-mail: sjqiao@cuit.edu.cn.*
- *Z. Guan is with the School of Information and Technology, Northwest Univerisity, Xi'an, Shaanxi 710127, China. E-mail: ziyuguan@nwu.edu.cn.*

Manuscript received 6 Mar. 2018; revised 12 Nov. 2018; accepted 19 Nov. 2018. Date of publication 28 Nov. 2018; date of current version 8 Jan. 2020. (Corresponding author: Jianbin Huang.)
Recommended for acceptance by Z. Cai.
Digital Object Identifier no. 10.1109/TKDE.2018.2883938

historical data, such as SubSynEA [15]. SubSynEA takes only the starting location and the current location to predict the final destination. It decomposes and then synthesizes the historical trajectories using Bayesian inference. This method may obtain inaccurate prediction as it ignores the importance of the most recent movement. Moreover, some methods cannot deal with the data sparsity problem very well as some locations are never covered in the historical data, such as DESTPRE [18]. DESTPRE builds a sophisticated index and then uses partial matching to predict the destination and therefore it may fail to report a destination as some paths may not be present in the historical trajectory data.

To address the aforementioned problems, we propose DestPD which is composed of the offline training phase and the online prediction phase. We make the following contributions in this paper:

- We propose a novel approach DestPD for destination prediction. Instead of only considering the starting and current location of an ongoing trip, DestPD first predicts the most probable future location and then reports the destination based on that. It directly targets the route connecting two endpoints, which mitigates the data underutilization problem inherent in current methods.
- To improve the computational efficiency, we also design some data constructs:
  - Two data constructs ETP and TPD are proposed for training.
  - Since frequent update of the model is necessary to enhance prediction accuracy, we devise OUP and TAA that reduce its computational cost.
- We develop a new framework for future trajectory prediction:
  - A new method finding similarity of trajectory is proposed. The method is more efficient than earlier ones and aims at two trajectories that share overlapping sub paths. Overlap usually occurs in very similar trajectories and so we are more interested in the nuances between them.
  - To locate the nodes that are important to determine possible trajectory direction, we introduce an efficient best node selection technique based on dynamic programming. Compared with the naive approach of exponential time and space complexity, it requires only scanning all the nodes in one pass.
- The experimental results on real-world and synthetic datasets show that DestPD is more accurate than three state-of-the-art methods. Furthermore, DestPD achieves higher coverage and cuts one order of magnitude in both time and space complexity.

The rest of the paper is organized as follows. We introduce the related work in Section 2. Then Section 3 presents the proposed DestPD approach, in which training ETP and TPD, the frequent update technique and the future trajectory prediction framework are detailed in three different sections respectively. In Section 4, the experimental settings and analyses of the results are provided. Finally, Section 5 concludes this paper.

2 Related Work

Previous work pertaining to this subfield of urban computing revolves around prediction/recommendation concerning routes/POIs in bustling urban areas. Most of these work tend to discover patterns which they term as 'popular' for subsequent decision-making that seeks to optimize a certain goal, be it a place of interest or future movement [5], [11], [12], [16], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29]. They generally follow two lines of investigation:

Destination prediction is most directly related to our work. Xue et al. [15] predicted destinations through a Bayesian based probabilistic framework that decomposes a trajectory into smaller segments and then combines them together. Xu et al. [18] designed an index to efficiently retrieve the original historical trajectories for its succeeding destination prediction. Krumm and Horvitz [11] incorporated in their approach multiple features including driving efficiency, ground cover, trip times etc. and employed an open-world model to capture the probabilities of users leaving for places which have never been visited in the past. Ziebart et al. [12] employed a sophisticated context-aware behavior model PROCAB to infer intersection decisions, routes and destinations of drivers. Gao et al. [25]. demonstrated the breach of privacy induced by insurance schemes through predicting the destinations by exploiting only the speed data. The approaches and aims of our work differs from all the preceding research. Besides, we also concentrate on destination prediction using only the historical trajectories, which has not been well studied in the past.

Future movement prediction/recommendation is also drawing more attention: Jeung et al. [27] proposed two query processing techniques that can obtain future movement prediction through a novel access index. The knowledge of the possible end points of a journey also facilitates opportunistic routing. Horvitz et al. [17] recommended sensible diversions along one trip route to a primary destination. Monreale et al. [28] designed a T-pattern Tree which is learnt from trajectory patterns. Future trajectory prediction [29] had even been applied in the Decision Support System (DST) of Air Traffic Management (ATM). Do et al. [5] developed an ensemble method that builds a contextual variable into a probabilistic framework for human mobility prediction. Yin et al. [30] recommended future venues or events to users based on personal preferences and historical spatial-temporal information. Liu et al. [31] proposed recommendation of point of interest (POI) based on multi factors via a holistic probabilistic model. In [32] Liu et al. studied temporal interval assessment in POI recommendation. In [33] Liu et al. provided personalized travel package recommendation using a cocktail approach. Ying et al. [34] clustered users to predict future locations through mining semantic trajectories. Do et al. [35] drew on the rich contextual information from smart phone sensors to infer where users will go. Wang et al. [36] investigated a unified model that combines both regularity and conformity using heterogeneous mobility data. Cho et al. [37] explored the relationship between mobility and friendship. Theses studies also include [13], [38]. They are directed at the prediction of subsequent movement, next places or the future trajectory all of which do not address the problem of destination prediction as our work does.

A majority of the aforementioned research focuses on one or several geo locations that matter most, either the current position or some statistically significant places, and then perceive them as several discrete states of a Markov model or a HMM [5], [8], [11], [13], [15], [17].

TABLE 1
The Adopted Notations (in Alphabetical Order)

Symbol	Description
α	Decay factor
d_i	Subinterval boundary point (distance)
d_{ab}	The L1 distance between location a and location b
d_t	Distance traveled so far
d_p	The length of predicted path of the future trajectory prediction framework
D_p	The desirable length of predicted path obtained through logarithmic decay
D	Total trip distance
len_d	Length of detour
Loc_p	Most probable future location
M^l	l-step Markov transition matrix
$p_{i\to j}$	Total transition probability for a trip from location i to location j
P_d	Destination probability
$P(d\|s)$	Transition probability for a trip starting at location s and ending at location d
T_p	The trajectory traveled so far
T_s	Trajectories starting at s
$T_{s,d}$	Trajectories starting at s and ending at d

Our work differs from most previous work in the following three respects. (1) First our method prioritizes the most recent movements. Data underutilization is the problem that SubSynEA [15] may suffer, since its model essentially considers only the starting location and the current location. Our approach takes into account the most recent movements through the future trajectory prediction framework. To some extent it resembles partial matching, but it can avoid the drawback (i.e., low coverage) of such methods [18] by our following probabilistic prediction. (2) Second, our work relies solely on the historical trajectory data to accomplish the prediction task. This is different from most previous work [5], [11], [12], [17], [39] (i.e., no other information such as time or user profiles is included). This general setting allows us to analyze user movement when such knowledge is not available, which is often the case. (3) Third, our approach cuts the computational effort associated with the Markov model. Specifically, compared with SubSynEA [15], we achieve one order of reduction in both time and space complexity. This fall in computational cost and our flexible data constructs (i.e., ETP and TPD) permits us to perform the frequent update of our model that enhances the prediction accuracy.

3 Efficient Destination Prediction

In this section, we present our approach to destination prediction. Our approach comprises the offline training to obtain ETP and TPD (Section 3.2), the frequent update technique (Section 3.3) and the online future trajectory prediction framework (Section 3.4). To facilitate better understanding of our work, all the notations are listed in Table 1.

3.1 Problem Definition

The problem of *destination prediction* can be formulated as follows: given the starting location $\boldsymbol{s}$, current location $\boldsymbol{c}$ and trajectory $\boldsymbol{sc}$ connecting them, we aim to find the final destination of journey $\boldsymbol{d}$. Our investigation of this problem also involves an intermediate variable, $\boldsymbol{Loc_p}$, the most probable future location produced by our algorithm of future trajectory prediction.

Xue [15] first derived Equation (1) for destination prediction. Equation (1) combines all the transition probabilities for our ultimate destination prediction. It shows the relationship between the three variables that jointly determine the final destination probability. During the offline phase, $p_{Loc_p\to d}$ and $p_{s\to d}$ are TPDs yielded through an iterative approach (Section 3.2). Both $p_{Loc_p\to d}$ and $p_{s\to d}$ are computed by the offline training algorithm (Algorithm 1, Section 3.2). In the online stage, future trajectory prediction is performed to obtain the most probable future location Loc_p(Section 3.4)

$$P_d \propto \frac{p_{Loc_p\to d}P(d|s)}{p_{s\to d}}. \tag{1}$$

We employ Bayesian inference to perform our prediction task. And this method can also be essentially described by Equation (1). $p_{Loc_p\to d}$ denotes the total transition probability for a journey from location Loc_p, the most probable future location, to a presumed destination d. Likewise, $p_{s\to d}$ represents a trip which starts at s. Note that $p_{i\to j} = M_{ij}^{d_{ij}}$ is actually an element of a d_{ij}-step Markov transition matrix which can be obtained through multiplying the single-step matrix d_{ij} times. $P(d|s) = \frac{|T_{s,d}|}{|T_s|}$ reflects the proportion of trajectories that begin at the same origin s (denominator $|T_s|$) but end up at different locations d (numerator $|T_{s,d}|$).

3.2 Training ETP and TPD

After we have obtained the single-step transition matrix from the historical trajectory data (e.g., M^1), we should then proceed to calculate multi-step transition probabilities by matrix multiplication. However, Markov transition matrix multiplication remains to be the major hurdle of performance improvement for our offline training, particularly for very large matrix (i.e., high granularity). In our case, matrix multiplication can be a computationally formidable task particularly when the size and the number of transition steps entailed by the Markov transition matrix are large. According to [15], the offline training for SubSynE takes beyond one hour for a map of medium grid granularity setting on a commodity machine. Therefore, the improvement of offline training can be essential to destination prediction. We achieve this goal through the introduction of two data constructs, ETP and TPD.

3.2.1 Definition of ETP and TPD

First let us define some key concepts before we analyze the efficiency improvement.

Definition 1 (Relative Adjacent Pair—RAP). *Given two locations i and j, the relative adjacent pair, $A_{ij} = \{A_{ij}^1,\ A_{ij}^2\}$, comprises precisely two cells that are immediately adjacent to j regarding i in the L1-metric sense. These two adjoining cells are on the route that links i with j.*

A special case arises when two cells are in the same row/column. In this case the RAP comprises soley one element which is the adjoining cell of j regarding i.

Example 1. In Fig. 1, cell 6 ($A_{19}^1 = 6$) and cell 8 ($A_{19}^2 = 8$) together form the RAP of cell 9 regarding cell 1. In addition, cell 2 is the singleton RAP of cell 3 regarding cell 1 (i.e., $A_{13} = 2$, the aforementioned special case).

Definition 2 (ETP—Efficient Transition Probability). *Given two locations i and j, $ETP(i, j, l)$ is the probability of*

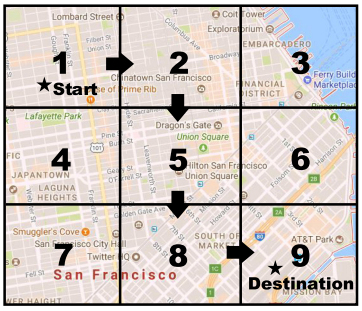


Fig. 1. An example that shows how ETP and TPD work.

the transition taking the most efficient route whose length corresponds to the L1 distance l.

The relationship between $(l-1)$-step transition and l-step transition can be found by

$$ETP(i,j,l) = \sum_{k=1}^{2} ETP(i, A_{ij}^k, l-1)P(j|A_{ij}^k), \tag{2}$$

where $P(j|A_{ij}^k) = \frac{|T_{A_{ij}^k,j}|}{|T_{A_{ij}^k}|}$, the single step transition probability, measures the frequency of transition from A_{ij}^k to j. Equation (2) consists of exactly the two components of Relative Adjacent Pair (RAP), i.e., A_{ij}^1 and A_{ij}^2 to recursively obtain the efficient transition probabilities.

The strength of this technique compared with sparse matrix multiplication is its ability to calculate the necessary transition probabilities only once and save them for later computation.

It is akin to the divide-and-conquer tactic in that every problem (reaching location j) can be worked out by dealing with its sub problems (reaching the RAP of location j).

Next we present TPD to deal with trajectories that may contain detours.

Definition 3 (TPD—Transition Probabilities with Detours). *Given two locations i and j, $TPD(i,j,len_d)$ gauges the probability of transition taking the route with a detour whose length is l_d.*

The definition of TPD resembles that of ETP when no detour is involved, i.e., $len_d = 0$. Next we can find that TPD can be obtained recursively by

$$TPD(i,j,len_d) = \begin{cases} ETP(i,j,l) & (len_d = 0) \\ \sum_{k=1}^{4} TPD(i,j_k,l_{ij_k})P(j|j_k) & (len_d > 0), \end{cases} \tag{3}$$

where j_k denotes the 4 cells that are immediately adjacent to j. This data construct has the capability to meet the needs of detours of different lengths without being prone to performance reduction as is matrix multiplication. Actually, the locations surrounding the starting point alternate between two states, either reachable or unreachable (see also Appendix A and Appendix B, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2018.2883938, for several illustrating examples).

3.2.2 Offline DestPD Training

In Algorithm 1, our offline training first obtains the ETP regarding two locations i and j whose corresponding L1 distance is d_{ij} (line 2-4). Then we can yield the TPD pertaining to a detour of len_d in addition to d_{ij} (line 6-10). ETP is first calculated in order to find TPD. Subsequently, we store the sum of TPDs in the corresponding total transition probability $p_{i\to j}$ (line 8). The increment of l' is two because of the unreachability inherent in two adjoining cells (line 9). We call this strategy 'Efficient First Detours Later' which enables us to obtain TPD in an efficient iterative manner (see also Appendix A, available in the online supplemental material, for a typical example of ETP and TPD).

Algorithm 1. Training ETP and TPD

Input: the single-step Markov transition matrix M^1 of the size $g^2 \times g^2$.
Output: the total transition probabilities for all of the origin-destination pairs.

1: **for** all the reachable origin-destination pair (i,j) **do**
2: **for** $l \leftarrow 1$ to $2g$ **do**
3: $p_{i\to j} \leftarrow ETP(i,j,l)$ (Equation (2))
4: **end for**
5: $l' \leftarrow 2$
6: **while** $l' < len_d$ **do**
7: ▷ *Compute every TPD whose length of detour is l for all the origin-destination pairs.*
8: $p_{i\to j} \leftarrow p_{i\to j} + TPD(i,j,d_{ij}+l')$ (Equation (3))
9: $l' \leftarrow l' + 2$
10: **end while**
11: **end for**
12: **return** The $p_{i\to j}$ of all the origin-destination pair (i,j).

The Upper Bound of Non-zero Entries. The reduction of computational complexity can be shown briefly through the analysis of upper bound of non-zero entries. The amount of ETP and TPD are always far below this bound.

It is apparent that during each step of transition one can travel to only half of the locations in the neighboring region of the starting point (see also Fig. 3 of Appendix B, available in the online supplemental material, for a concrete example). Here by claiming "half", we refer to the fact that at least 50 percent of the entries of a transition matrix are zero which can indeed be stated as a theorem below.

Theorem 1. *Given a g-by-g s-step Markov transition matrix m in the model, the amount of non-zero entries is $nz = \{m_{ij} | m_{ij} \neq 0\}$, then $\frac{|nz|}{|m|} \leq 0.5$ $(|m| = g^2)$.*

Proof. From any location in a map one can travel to the four directly adjacent cells. The ensuing move should land him on any of these four adjoining cells of his previous move. In order to get to his subsequent destination, he has to first leave his previous starting location, which essentially rules out his arrival on these places (starting points) in this turn of transition (self-transition excluded). Furthermore, since none of the immediately adjacent 4 cells is reachable after the previous step of transition, these starting cells cannot be got to from the other locations during this turn of transition as well (non self-transition also excluded now). The foregoing reasoning applies to every step of transition, precisely rendering at least half of the total locations (starting points) impossible to get to and the other half reachable (destinations). Hence the conclusion can be drawn that the non-zero elements

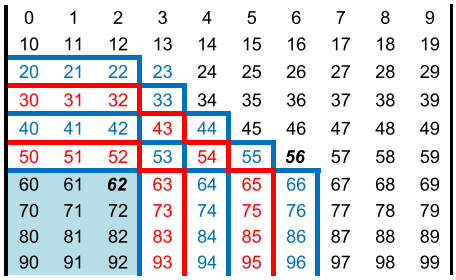


Fig. 2. In a map of granularity 10×10, the region of $TAA(56, 62, D, T)$, $D = \{0, 2, 4, 6, 8\}$, a longer detour corresponds to a more extended area.

constitute no more than 50 percent of a transition matrix. □

ETP and TPD are the cornerstone on which our approach is built. The reader is referred to Appendix A, available in the online supplemental material, for a more thorough discussion of their rationale, Appendix B, available in the online supplemental material, for an intuitive interpretation of efficiency improvement and Appendix C, available in the online supplemental material, for a rigorous analysis of complexity reduction of our approach.

3.3 Frequent Update of ETP and TPD

Traffic conditions undergo constant and instantaneous changes at every moment and it is rational that we capture its latest trend by frequently updating our model in the pursuit of better results. We notice that only a portion of all the transition probabilities varies during a short period of time, for instance at three-minute intervals.

Our approach focuses only on the portion whose traffic condition has altered in a very short period and hence is suitable to be frequently updated. Our treatment differs from its predecessor in that it breaks down the structure of Markov transition matrix and is directed at the items that are integral to the efficient transitions between cells in a map with or without detours. Consequently, we can adjust the proportion of updates for transition probabilities to meet the requirements of the constantly changing traffic condition while keeping the incidental cost regarding these variations as low as possible.

To factor into such changes of road traffic at a particular moment, previous approaches [8], [15] have to perform matrix multiplications for all of the cells residing in a map. However, according to our analysis, since changes occur at merely a part of the whole map, redundant computations are carried out by this solution. So we devise TAA and OUP to avoid this redundancy.

3.3.1 Definition of OUP and TAA

First let us present some definitions concerning OUP and TAA, two most essential components of frequent update of the model.

Definition 5 (OUP—Obligatory Update Point). *One needs to travel past the transit point k on the route from location i to location j. This transit point coincides with the update point that affects the region (TAA of Definition 6) which must be updated once the transition probabilities of this point have altered.*

In a very short time interval, the traffic condition of only a very small amount of locations have changed. However, the variation still influences substantially the area regarding this update point. Therefore it is sensible to first locate these points and then update the affected area accordingly. For example in Fig. 2, for $TAA(56, 62, D, T)$, location 62 is the OUP.

Definition 6 (TAA—Transition Affected Area). *Departing from location i and walking past the intermediate transit point j, one is likely to reach any of the cells residing in the transition affected area $TAA(i, j, D, T)$ after he takes his ensuing moves that may include a detour $d \in D$ adding up to the distance $t \in T$ of the whole trip.*

Fig. 2 shows an example of the region of $TAA(56, 62, D, T)$. It is evident that a longer detour always corresponds to a more extended area.

3.3.2 Online Frequent Update Algorithm

Our efficient destination prediction approach actually consists of two parts concerning ETP and TPD: in Algorithm 1 the first offline phase initializing the total transition probabilities for all of the origin-destination pairs, and in Algorithm 2 the second one which continuously enhances our prediction by frequently updating the model.

Algorithm 2. Frequent Update of ETP and TPD

Input: the set G_c of cells that have undergone changes causing the alteration of their transition probabilities.
Output: the updated total transition probabilities for all of the origin-destination pairs.

1: **for** every cell g_i in the map **do**
2: Find the nearest OUP with respect to g_i in G_c, denoted by OUP_{g_i}.
3: Find $TAA(g_i, OUP_{g_i}, D, T)$, denoted by $TAA(OUP_{g_i})$. (Algorithm 3)
4: **for** every cell g_j in $TAA(OUP_{g_i})$ **do**
5: ▷ *Similar to the offline training of its corresponding ETP and TPD.*
6: $p_{i \to j} \leftarrow ETP(g_i, g_j, l)$ (Equation (2))
7: $l \leftarrow 2$
8: **while** $l < len_d$ **do**
9: ▷ *Compute every TPD whose length of detour is l for all the origin-destination pairs.*
10: $p_{i \to j} \leftarrow p_{i \to j} + TPD(i, j, d_{ij} + l)$ (Equation (3))
11: $l \leftarrow l + 2$
12: **end while**
13: **end for**
14: **end for**
15: **return** All the updated $p_{i \to j}$ regarding cell g_i

In Algorithm 2, when we update the DestPD model, we first need to locate all the nearest OUPs regarding g_i (line 2). To guarantee all the necessary updates are performed, in total 4 OUPs in 4 directions are considered, namely west-north, east-north, east-south and west-south. We then compute the values of ETPs residing within a TAA (line 6). And then we move on to find their corresponding TPDs and keep track of the sum in $p_{i \to j}$ (line 10). The step size of this loop is still 2 in accordance with Theorem 1 (line 11).

In Algorithm 3, the TAA of interest can be found by first identifying the initial rectangular area with respect to cell i

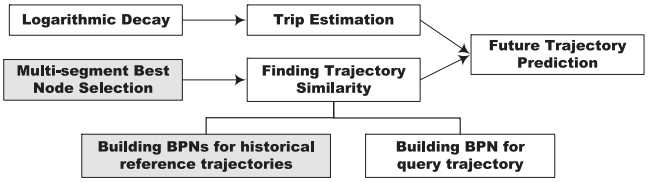


Fig. 3. A sketch of our framework for future trajectory prediction. The shaded boxes indicate offline processing.

and its OUP – j (line 2). This rectangle essentially comprises all the cells whose detours are 0 and can be obtained as follows: first we draw a vertical line and then a horizontal one across j; the whole map now is partitioned into 4 regions; then the rectangle that is in the diagonal direction of the region containing cell i is the desirable $rect(i, j)$. Once we have initialized this 0-detour TAA, we can then move on to find TAAs with longer detours by gradually extending their smaller counterparts through taking in their border neighbors (line 3-5). This is also shown in Fig. 2. The border color of the extended TAA region alternates between red and blue.

Algorithm 3. Find $TAA(g_i, OUP_{g_i}, D, T)$

Input: cell g_i, OUP_{g_i}, the set of detours D.
Output: the corresponding $TAA(g_i, OUP_{g_i}, D, T)$ regarding cell g_i, detours D and total travel distances T.
1: **for** every $o \in OUP_{g_i}$ **do**
2: $TAA(g_i, o, D, T) \leftarrow rect(i, j)$
3: **for** every $d \in D$ **do**
4: $TAA(g_i, o, d, T) \leftarrow TAA(g_i, o, d, T) \cup TAA(g_i, o, d-2, T).borderNeighbors(i, j)$
5: **end for**
6: **end for**
7: **return** $TAA(g_i, OUP_{g_i}, D, T)$

3.4 Future Trajectory Prediction for Loc_p

In this section, we discuss the incorporation of future trajectory prediction framework into our prediction model during the online prediction phase. We first determine the predicted length of ongoing trajectory (*Logarithmic Decay → Trip Estimation*) and then identify the location Loc_p that is most likely to be traveled in the future (*Multi-segement Best Node Selection → Find Trajectory Similarity → Future Trajectory Prediction for* Loc_p). After that our model produces the predicted results given the knowledge of this most probable future location. A sketch of our prediction framework is shown in Fig. 3.

Fig. 4 illustrates the relationship among all the concerning variables. First we employ the future path prediction framework to generate a path that connects the current location c to the most probable future location Loc_p. We specify the desirable predicted length D_p by applying a logarithmic decay to $E(D|d_t)$, the estimated total trip distance at the current timestamp. Then the end point of D_p (i.e., Loc_p) replaces the current location c since Loc_p is more likely to be closer to the final destination and thus gives better prediction results.

3.4.1 Trip Estimation

To decide on a proper value for the length of a predicted path, we first create a frequency diagram depicting the distribution of total trip distance of our historical data. Specifically, we let distance measurements fall into one of the subintervals separated by d_i. The expected value of total travel distance is calculated by

$$E(D) = \sum_i d_i P(d_i < D < d_{i+1}). \tag{4}$$

Then we iteratively estimate the total trip distance at a particular instant of time as

$$E(D|d_t) = \sum_i^{\infty} d_i \frac{P(d_i < D < d_i + 1)}{P(D \geqslant d_t)}, \tag{5}$$

where d_i should satisfy

$$\begin{cases} d_i < d_t, & i = sup\{i \geqslant 0 | d_t > d_i\} \\ d_i \geqslant d_t, & \text{otherwise.} \end{cases}$$

Equation (5) provides the expected value of total trip distance given the current trip that has been traveled so far. It offers us a ballpark figure of the journey distance at a specific time which can be used to determine the predicted length D_p.

Logarithmic Decay. As mentioned in the overview of this chapter, we seek to identify Loc_p and thus only a certain proportion of $E(D|d_t)$ (i.e., D_p) is taken to achieve this aim. Moreover, the rationale for the reduction of predicted path is that Loc_p should also approach the current location c as the trip gradually comes to its end. Hence we employ a logarithmic function to perform this task which is given as

$$D_p = E(D|d_t) \log_{\alpha} \frac{d_t}{E(D|d_t)}, \tag{6}$$

where the argument of the logarithm $\frac{d_t}{E(D|d_t)}$ quantifies the estimated trip completion percentage based on our preceding $E(D|d_t)$. The base α, the decay factor, indicates how fast the predicted percentage should decline. We repeatedly alter the decay factor α in our experiment and find that setting it to 0.004 works the best.

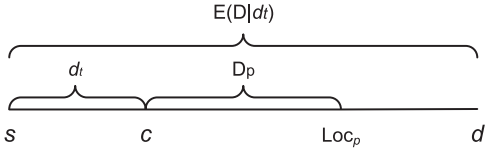


Fig. 4. The relationship among all the concerning variables.

3.4.2 Find Trajectory Similarity

We propose a novel algorithm to measure the similarity between the query trajectory and its reference trajectories. We observe that very often there is overlap between the query trajectory and its reference trajectories. Hence we come up with a technique that focuses on the sub paths that are actually different and ignores the overlap. First let us define some important concepts.

Definition 7 (Intermediate Node). *Let path* $\langle n_1, n_2, \ldots n_k \rangle$ *be the shortest one that connects* n_1 *with* n_k. *Suppose* n_{k+1} *is adjacent to* n_k *and path* $\langle n_1, n_2, \ldots n_k, n_{k+1} \rangle$ *is not the shortest one that links* n_1 *with* n_{k+1}. *We call* n_k *the* intermediate node.

Essentially, we compare both the query trajectory and the reference trajectory with the shortest path to accelerate

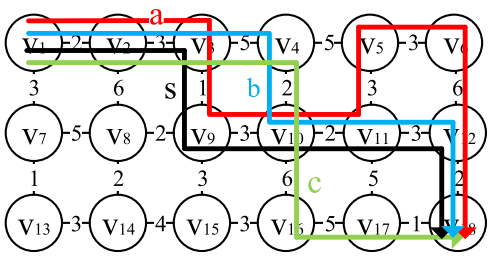


Fig. 5. An example illustrating time varying best path notation BPN.

similarity finding. The nodes that deviate from the shortest path are the ones that we are most concerned with. Because they are circumstantially linked to possible differences between the query trajectory and the reference trajectory. Specifically, it suffices to only record the first node that is not on the shortest path. In our case, it is the *intermediate node* n_k (Definition 7). Because this node can then be the springboard for us to identifying the actually different sub paths of the query trajectory and the reference trajectory (Algorithms 5 and 6).

Definition 8 (Time Varying Best Path Notation). *At time t, given two nodes v_i and v_j, we denote the best path connecting the two endpoints, v_i and v_j, as the one with the minimum travel cost, i.e., $BPN(t, v_i, v_j) = \langle v_i, v_j \rangle_t$. A path with a certain degree of inefficiency that connects v_i with v_j but also travels past the node v_k (i.e., v_k is not on the preceding shortest path) then can be defined as $BPN(t, v_i, v_j) = \langle v_i, v_k, v_j \rangle_t$. In the latter case, we need to ensure that path $\langle v_i, v_k \rangle$ and path $\langle v_k, v_j \rangle$ are two shortest paths regarding their respective endpoints. Moreover, v_k is exactly the* intermediate node *(Definition 7) of $\langle v_i, v_k, v_j \rangle_t$.*

Example 2. Let us illustrate this concept with an example (Table 2). There are 3 paths in Fig. 5 at time t, the shortest path $s = \langle v_1, v_2, v_3, v_9, v_{10}, v_{11}, v_{12}, v_{18} \rangle_t$, the query path $a = \langle v_1, v_2, v_3, v_{19}, v_{10}, v_{11}, v_5, v_6, v_{12}, v_{18} \rangle_t$, and the reference path $b = \langle v_1, v_2, v_3, v_4, v_{10}, v_{11}, v_{12}, v_{18} \rangle_t$. Using Time Varying Best Path Notation, the preceding 3 paths are respectively denoted by $s = \langle v_1, v_{18} \rangle_t$, $a = \langle v_1, v_6, v_{18} \rangle_t$, $b = \langle v_1, v_4, v_{18} \rangle_t$. Since only v_6 is the node that causes the inefficiency of query path a to reach v_{12}, we explicitly record this node in the Time Varying Best Path Notation. Additionally, for path a, this notation also ensures that both route $\langle v_1, v_6 \rangle$ and route $\langle v_6, v_{18} \rangle$ are the shortest regarding their respective end points, i.e., (v_1, v_6) and (v_6, v_{18}). Similarly, for reference path b only v_4 plays the part of lengthening the path that reaches v_{10}, hence we also record this node in its notation.

Time Varying Best Path Notation can be obtained by Algorithm 4. In Algorithm 4, first we compute the all-pair shortest path with a method such as Floyd-Warshall algorithm [40], and store it in $BP(t, v_i, v_j)$ for every node pair (line 1-3). Next, every BPN is initialized with only the first node of original trajectory (line 4). Then we simply include the nodes that induces the inefficiency of a route (line 6-11). We examine whether the last node of $BP(t, v_i, v_j)$ coincides with the node of original trajectory (line 7). If not, then we need to record the node in BPN (line 8), and subsequently our best-path check should focus on a new segment starting from node j (line 9).

TABLE 2
Time Varying Best Path Notation

Full Path	Time Varying Best Path Notation
$s = \langle v_1, v_2, v_3, v_{19}, v_{10}, v_{11}, v_{12}, v_{18} \rangle_t$	$\langle v_1, v_{18} \rangle_t$
$a = \langle v_1, v_2, v_3, v_{19}, v_{10}, v_{11}, v_5, v_6, v_{12}, v_{18} \rangle_t$	$\langle v_1, v_6, v_{18} \rangle_t$
$b = \langle v_1, v_2, v_3, v_4, v_{10}, v_{11}, v_{12}, v_{18} \rangle_t$	$\langle v_1, v_4, v_{18} \rangle_t$
$c = \langle v_1, v_2, v_3, v_4, v_{10}, v_{16}, v_{17}, v_{18} \rangle_t$	$\langle v_1, v_4, v_{16}, v_{18} \rangle_t$

Example 3. Consider the construction of Time Varying Best Path Notation of path a in Fig. 5. Initially, there is only one node $BPN(t, v_{i1}, v_{ik}) = \langle v_1 \rangle_t$. The first 7 nodes of path a happen to be on the shortest path that connects v_1 with v_6. Hence the first 6 iterations for path a do not append any node to its Time Varying Best Path Notation. Subsequently, path a reaches node v_{12}. Compared with the shortest path, this additional node v_6 gives rise to a longer trip that ends at node v_{12}. During this iteration, the mismatch between node v_{11} and node v_6 causes the condition of the if statement (line 7) to evaluate to false, and thus node v_6 is appended to the Time Varying Best Path Notation of path a (line 8). The starting location s assesses whether a path is the shortest relative to the position of s. Initially s points to v_1. After the inclusion of a new node v_6, we have to update s to the new location v_{11}, accordingly reflecting the addition of v_6.

In our following discussion, we omit the subscript t of all BPNs for sake of the succinctness of notation.

Algorithm 4. $ObtainBPN(Tr_t)$—Obtain the Time Varying Best Path Notation

Input: a trajectory $Tr_t = \langle v_{i1}, v_{i2}, \ldots v_{ik} \rangle_t$ of length k at time t.
Output: the Time Varying Best Path Notation of Tr_t $BPN(t, v_{i1}, v_{ik})$.

1: **for** all node pairs (v_i, v_j) in the map **do**
2: $BP(t, v_i, v_j) \leftarrow$ the path with the lowest travel cost at time t
3: **end for**
4: $BPN(t, v_{i1}, v_{ik}) \leftarrow \langle v_{i1} \rangle_t$
5: $s \leftarrow 1$
6: **for** $j \leftarrow 2$ to k-1 **do**
7: **if** $BP(t, v_{is}, v_{ij})$'s last node before $v_{ij} \neq v_{i(j-1)}$ **then**
8: $BPN(t, v_{i1}, v_{ik}) \leftarrow BPN(t, v_{i1}, v_{ik}) \cup v_{i(j-1)}$
9: $s \leftarrow j$
10: **end if**
11: **end for**
12: **return** $BPN(t, v_{i1}, v_{ik})$

In Algorithms 5 and 6, we efficiently find the similarity between the query path q and the reference path r. This similarity later serves as the support for future trajectory prediction. There are already several well established approaches dealing with this issue in the literature, such as Dynamic Time Warping based method [41], [42], Longest Common Subsequence based method [43] and Edit Distance based method [44], [45] etc. However, in our problem most paths overlap substantially and only a certain portion of the whole path differs largely from one another. This subtlety allows us to devise a technique that only concentrates on these differences and ignores the rest. Indeed our method can be mostly suitable for the problem of finding path similarity characterized by the aforementioned condition (i.e., a certain degree of overlap).

Algorithm 5. $FindSim(q, r)$—Efficient Identification of the Similarity of Paths

Input: the Time Varying Best Path Notation of the query path $q = BPN(t, v_{i1}, v_{i2}, \dots v_{ik})$ of length k, and that of the reference path $r = BPN(t, v_{j1}, v_{j2}, \dots v_{jk'})$ of length k'.
Output: the numeric value sim of similarity between the query path q and the reference path r.
1: **for** $m \leftarrow i1$ to ik **do**
2: $q', r' \leftarrow BuildSubPath(q, r, m)$ (Algorithm 6)
3: $sim \leftarrow sim + Similarity(q', r')$
4: **end for**
5: **for** $n \leftarrow j1$ to jk' **do**
6: $q', r' \leftarrow BuildSubPath(r, q, n)$ (Algorithm 6)
7: $sim \leftarrow sim + Similarity(q', r')$
8: **end for**
9: **return** sim

In Algorithm 5, we iterate through the Time Varying Best Path Notation of the query path q and that of the reference path r (line 1-4, line 5-8). During each iteration, two sub paths of comparison are built using Algorithm 6. Then we calculate the similarity between the two sub paths and add this value to the cumulative sum sim (line 3 and line 7) which is the final output of Algorithm 5 (line 9).

In Algorithm 6, the start index s, one of the three input parameters, indicates the position that we begin to discover the two sub paths. Since the last element visited by Algorithm 6 is the predecessor of v_s (line 17-21), we indeed find the two sub paths in a cyclic manner. First two empty sub paths q' and r' are initialized (line 2). The underlying data structure of q' and r' is double-ended queue. Double-ended queue supports the $inject()$ operation, inserting an element at back, and the $push()$ operation, putting an element at front. After this initialization, we check whether node v_s has been visited. If so we simply return two empty sub paths whose similarity is 0. That the $v_s.visited()$ evaluating to true (line 3) implies that v_s has been covered in earlier sub paths. Hence we do not need to consider v_s again. We then store v_s in variable v and the original full path of q and r in qf and rf respectively (line 6 and line 7). Next we actually build the two sub paths in the succeeding 3 $while$ loops. The first one puts all the subsequent nodes after v_s in the sub path of query path q' (line 8-11). The second one places all the nodes in the sub path of reference path r' (line 12-16). The third loop prepends the nodes before v_s to q' (line 17-21). The $intersects()$ function (line 8 and line 13) ensures that the current v always lies within the boundary of either path q (i.e., $!v.intersects(r)$) or path r (i.e., $!v.intersects(q)$). Before the inclusion of v into q' or r' and the transition to its respective successor or predecessor from its full path (line 10, line 15 and line 20), we $visit()$ this node (line 9, line 14 and line 19) to prevent future revisit (i.e., $v_s.visited()$ on line 3).

Example 4. Still consider the previous example in Fig. 5 Suppose a is the query path $q = \langle v_1, v_6, v_{18} \rangle$, b the reference path $r = \langle v_1, v_4, v_{18} \rangle$. First we traverse the Time Varying Best Path notation of path q and come across v_6. v_6 has not been visited before and thus we $inject()$ v_6 at the end of sub path q'. Next we come to v_{12}, the successor of v_6. v_{12} $intersects()$ path r and therefore we too $inject()$ v_{12} to sub path q' but now we break this loop. For now it is apparent that sub path $q' = \langle v_6, v_{12} \rangle$ is partially built through the forgoing first loop. We then move on to build sub path r'. Akin to the preceding process, we add v_{12} and v_{11} to sub path r', the only difference being that the $push()$ operation of the double-ended queue is employed since the nodes of r now is retrieved in reverse order. After this second loop terminates, we can see that the construction of sub path $r' = \langle v_{11}, v_{12} \rangle$ is finished. Next we proceed to the last loop that $push()$ es the first two nodes v_{11} and v_5 to sub path q', which completes the construction of sub path $q' = \langle v_{11}, v_5, v_6, v_{12} \rangle$. Finally we return the desired two sub paths, the query sub path $q' = \langle v_{11}, v_5, v_6, v_{12} \rangle$ and the reference sub path $r' = \langle v_{11}, v_{12} \rangle$.

Algorithm 6. $BuildSubPath(q, r, s)$—Build Sub Path of Comparison for v_s

Input: the query path q, the reference path r, and the start index s of query path q.
Output: two sub paths of comparison q' and r'.
1: ▷ *Double-ended queue q' and r'*
2: $q' \leftarrow \langle\rangle, r' \leftarrow \langle\rangle$
3: **if** $v_s.visited()$ **then**
4: **return** q', r'
5: **end if**
6: $v \leftarrow v_s$
7: $qf \leftarrow q.fullPath, rf \leftarrow r.fullPath$
8: **while** $!v.intersects(r)$ **do**
9: $v.visit()$
10: $q'.inject(v), v \leftarrow v.succesor$ from qf
11: **end while**
12: $q'.inject(v), v \leftarrow v.predecessor$ from rf
13: **while** $!v.intersects(q)$ **do**
14: $v.visit()$
15: $r'.push(v), v \leftarrow v.predecessor$ from rf
16: **end while**
17: $r'.push(v), v' \leftarrow v, v \leftarrow v_s.predecessor$ from qf
18: **while** $v \neq v'$ **do**
19: $v.visit()$
20: $q'.push(v), v \leftarrow v.predecessor$ from qf
21: **end while**
22: **return** q', r'

Complexity Analysis. Compared with the conventional methods tackling measuring similarity between two paths, our approach focuses on the sub paths that are actually different. Hence the running time grows with the length of the sub paths instead of the two full paths. We list the time complexity of both two approaches in Table 3.

Let m and n denote the length of query path q and that of reference path r. m' and n' represents the length of the sub paths that are actually different. Apparently for Euclidean Distance, our method handles far less path locations if the lengths of two dissimilar sub paths are short. As for Dynamic Time Warping, Edit Distance and Longest Common Subsequence, it is also easy to show that our method boasts better time complexity. Let the length of query path q's each sub path of comparison q'_i designate by m'_i, and similarly n'_i for that of the reference path r. k denotes the total amount of occurrences of two different sub paths. Then we can have

$$\begin{aligned}\sum_i m'_i n'_i &= m'_1 n'_1 + m'_2 n'_2 + \cdots + m'_k n'_k \\ &\leqslant (m'_1 + m'_2 + \cdots + m'_k)(n'_1 + n'_2 + \cdots + n'_k) \\ &= \sum_i m'_i \sum_i n'_i \\ &\leqslant mn.\end{aligned}$$

TABLE 3
Time Complexity Comparison between the Conventional Method and Our Method

Method Name	Conventional Method	Our Method
Euclidean Distance [46]	$O(max(m,n))$	$O(max(m',n'))$
Dynamic Time Warping based method [41], [42], Edit Distance based method [44], [45], Longest Common Subsequence based method [43]	$O(mn)$	$O(\sum_i m'_i n'_i)$

Hence our method outperforms the conventional methods in terms of efficiency particularly when the length of differing sub path is not too large. For the example in Fig. 5, given trajectory s as the shortest path that links v_1 with v_{18}, our approach can attain $\frac{10\times8}{1\times1+2\times1}=\frac{80}{3}\approx 26.7$ times speedup for query path a and reference path b, and $\frac{10\times8}{1\times1+4\times2}=\frac{80}{9}\approx 8.9$ times speedup for query path a and reference path c.

We design BPN instead of merely recording one node that is not on the shortest route also because of potential efficiency gain. BPN allows us to load all the reference trajectories from disk in the compressed form. A hash table is then maintained in memory whose key is an (origin, destination) pair and value is the shortest path connecting these two endpoints. After that we recover the original trajectory with this hash table. The preceding method has a smaller memory footprint compared with the full path alternative, leading to better performance.

We would like to make a few final remarks on our approach. The intuition behind our algorithm is that we employ the shortest path as a metric which compares both a query path and a reference path. After one path is encoded in the BPN form, our subsequent focus is on the elements that comprise the non-shortest routes, which effectively excludes the common shortest ones. Hence our method is mostly suitable for finding similarity that involves overlap. Building BPN for all the reference trajectories in the historical dataset can be done offline, while the BPN of a query trajectory is obtained ad hoc. In this way we accelerate online similarity finding.

3.4.3 Multi-Segment Best Node Selection

Though the preceding trajectory similarity measure performs well to support future trajectory prediction, we observe that actually several critical nodes of a trajectory play a more important part in determining possible trajectory direction. Particularly those popular places are more likely to reveal the underlying travel patterns. The technique can be perceived as a sort of regularization, a term borrowed from the machine learning literature. Taking this analogy one step further, we focus more on the major stops of a trip to prevent trajectory overfitting.

We compute the normalized summation of number of Point of Interest (POIs) and travel frequency to capture this trait. These critical points have greater influence than the rest indicating the next most probable future location. Hence we devise a mechanism to identify these nodes and attach heavier weights to these nodes.

First every node of a trajectory is associated with an initial weight obtained from the popularity index (the summation of the number of POIs and its travel frequency). Then we want every selected node to be as representative as possible for its corresponding location. Thus we select a fraction of the nodes separated by the ones that are not chosen for this purpose. This means that an interval exists between two selected nodes to make them as diverse as possible.

Moreover, we also observe that more recent movement has greater impact on future travel direction. Thus we reflect this phenomenon through dividing a trajectory into several segments. And then we set different values for the aforementioned interval parameter m for them. The less recent segment, the larger the interval.

Workflow. We design an efficient dynamic programming method in Algorithm 7 to deal with this problem.

Algorithm 7. $SelectNode(Tr, k, M)$—Multi-Segment Best Node Selection

Input: a trajectory Tr, the number of segment k, the interval parameter $m_i \in M$ for segment i.
Output: the trajectory with selected nodes Tr'.

for $node \in Tr$ **do**
  $node.w \leftarrow$ Normalized summation of #POIs and travel frequency
end for
for every seg_i of Tr **do**
  $seg_i.S \leftarrow \{\}, seg_i.wSum \leftarrow 0$
  $cn \leftarrow seg_i.firstNode$
  repeat
    $pn \leftarrow seg_i.S \cap cn.predSet(m_i)$
    if $pn = \emptyset$ **then**
      $seg_i.S' \leftarrow seg_i.S, seg_i.wSum' \leftarrow seg_i.wSum$
      $seg_i.S \leftarrow seg_i.S \cup \{cn\}$
      $seg_i.wSum \leftarrow seg_i.wSum + cn.w$
      $cn \leftarrow cn.successor()$
      continue
    end if
    $delta \leftarrow cn.w - pn.w$
    if $delta > 0$ **then**
      $T \leftarrow seg_i.S, T.wSum \leftarrow seg_i.wSum$
      $seg_i.S \leftarrow seg_i.S' \cup \{cn\}$
      $seg_i.wSum \leftarrow seg_i.wSum' + cn.w$
      $seg_i.S' \leftarrow T, seg_i.wSum' \leftarrow T.wSum$
    end if
    $cn \leftarrow cn.successor()$
  until cn **is** $seg_i.lastNode$
  for $node \in seg_i \wedge node \notin seg_i.S$ **do**
    $node.w \leftarrow \alpha \times node.w$ $(\alpha \in (0,1))$
  end for
end for
return Tr' with its newly annotated weight values

First we would like to examine this problem from an intuitive perspective and present the naive recursive strategy. The problem can be formally defined as: given a chain of nodes, we want to find a subset such that (1) any two consecutive nodes in this new chain are at least distance m apart from each other in the original chain; (2) this subset of nodes attains the greatest weight sum.

To tackle this problem, we only need to consider two scenarios for the new chain: (1) including the first node (i.e., current node cn) of the original chain, (2) excluding the first node but taking in one node that is distance $[1, m]$ away from the first node.

By recursively applying the forgoing method we eventually enumerate all possible valid combination of nodes for

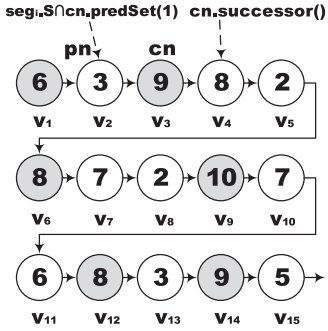


Fig. 6. An example illustrating best node selection algorithm. Interval parameter $m = 1$ for this segment. Gap $\in [1, 2]$. The shaded nodes are the selected ones that finally constitute $seg_1.S$.

our new chain. However, it is apparent that many sub chains generated by this recursive strategy will be identical. That is to say, many sub solutions actually coincide with one another. Hence we can employ dynamic programming to compute such sub chains only once and simply record the optimal sub chain. Listed in Algorithm 7 is the iterative technique that we take to avoid recursion of the naive approach.

Since seeking the best node set can be thought of as finding this set that includes the current node cn or not including this node, we use $seg_i.S$ to denote the set that has node cn, and similarly $seg_i.S'$ for the set without cn. By iteratively comparing the set with (i.e., $seg_i.S$) or without (i.e., $seg_i.S'$) newly appended node cn, we can always work out the current optimal solution.

In Algorithm 7, we first initialize the weight of every node with the normalized value of the sum of POI number and travel frequency (line 1-3). Then we generate the set of best node $seg_i.S$ for each segment of trajectory (line 4-28). At the beginning, $seg_i.S$ is an empty set and its sum of weight $seg_i.wSum$ 0 (line 5). cn represents the current node that we examine during this iteration, and we start from the first node $seg_i.firstNode$ of seg_i (line 6). pn denotes the possible preceding node of cn that has been selected in the previous iteration (line 8). We obtain pn by checking whether the intersection of $seg_i.S$ and the preceding m_i nodes $cn.predSet(m_i)$ is empty (line 9). If so we move on to the next iteration, visiting the successor $cn.scuccessor()$ of cn (line 10-14). Otherwise we then compute the difference $delta$ of weight of cn and pn (line 16). Only when the weight of current node $cn.w$ is greater than that of its previous selected node $pn.w$ do we update the set of best node $seg_i.S$. The update of $seg_i.S$ involves appending cn to $seg_i.S'$, the addition of $cn.w$ to weight sum $seg_i.wSum'$ and the respective assignment of these new values to $seg_i.S$ and $seg_i.wSum$ (line 19-21). At the end of update, we also record the old values (i.e., T and $T.wSum$) in $seg_i.S'$ and $seg_i.wSum'$ (7). $seg_i.S'$ and $seg_i.wSum'$ store the information of best node set and its weight sum without the inclusion of cn. After the best node set $seg_i.S$ is found, we reduce the weight of nodes that are not in this set through α (line 25-27).

In our experiment, all our historical trajectory data are preprocessed by Algorithm 7 with $SelectNode(Tr, 3, \{1, 2, 3\})$. That is to say we partition one trajectory into 3 equal segments. The gap value between two nodes in a segment is one in the set $\{1, 2, 3\}$.

We would like to further elaborate on the length of gap between two selected nodes. For a trajectory segment with interval parameter m, the length of gap should be in the range of $[m, 2m]$. Put differently, the gap is always no less than m (our requirement) but never exceeds twice the size of the minimal interval. Because once the gap reaches $2m + 1$, one can always add the middle node to the set of best node $seg_i.S$, increasing the total weight sum $seg.wSum$ and still satisfying the requirement of minimal interval.

TABLE 4
A Running Example of Algorithm 7 Starting from Node v_3

cn	pn	$seg_i.S$	$seg_i.wSum$	$seg_i.S'$	$seg_i.wSum'$
v_3	$\emptyset$	$\{v_1, v_3\}$	15	$\{v_1\}$	6
v_4	v_3	$\{v_1, v_3\}$	15	$\{v_1\}$	6
v_5	$\emptyset$	$\{v_1, v_3, v_5\}$	17	$\{v_1, v_3\}$	15
v_6	v_5	$\{v_1, v_3, v_6\}$	23	$\{v_1, v_3, v_5\}$	17

Time and Space Complexity. The most intuitive solution to this problem is a recursive search of all the possible combination of nodes. This recursion idea involves analyzing whether incorporating one node or excluding it but taking in its interval-m predecessor can attain the maximal weight sum. Assume a trajectory consists of n nodes. Since we have to check the foregoing two scenarios for every node along the trajectory during recursion, the final time and space complexity will be $O(2^n)$. Actually this simple recursive technique contains a substantial amount of overlapping sub problems. Hence we can employ a bottom-up approach that records the maximal weight sum of the current sub path. By virtue of memoization of dynamic programming, we only need to scan all the nodes in one pass, giving the time and space complexity $O(n)$.

Example 5. Let us consider the example in Fig. 6 that illustrates Algorithm 7 (also shown in Table 4). For brevity we focus on only one segment of a trajectory that shares the same interval value $m = 1$. Our Best Node Selection Algorithm will yield $seg.S = \{v_1, v_3, v_6, v_9, v_{12}, v_{14}\}$ and $seg.wSum = 50$ as the final result. Take node v_3 as a starting point of running example of Algorithm 7. Since p_n is empty, we first record the previous best set and its weight in $seg_i.S' = \{v_1\}$ and $seg_i.wSum' = 6$. Then the new node $cn = v_3$ is included in $seg_i.S = \{v_1, v_3\}$ along with its weight $cn.w = 9$ to total weight sum $seg_i.wSum = 6 + 9 = 15$. And this iteration ends, v_4 being the successor of cn. In the next iteration as $v_4.predSet(1) = \{v_3\}$, $seg_i.S = \{v_1, v_3\}$, $seg_i.S \cap v_4.predSet(1) = \{v_3\}$ and $delta = v_4.w - v_3.w < 0$, we simply skip v_4 and visit its successor v_5. In the iteration for v_5, because $seg_i.S = \{v_1, v_3\}$, $v_5.predSet(1) = \{v_4\}, pn = seg_i.S \cap v_5.predSet(1) = \emptyset$ we first update $seg_i.S' = \{v_1, v_9\}$ and $seg_i.wSum' = 15$, then include $cn = v_5$ in $seg_i.S = \{v_1, v_3, v_5\}$ and $seg_i.wSum = 15 + 2 = 7$. Next we come to the iteration for v_6. This time as $pn \neq \emptyset$ and $delta = v_6.w - v_5.w = 8 - 2 = 6 > 0$, we have to resort to seg'_S to build $seg_i.S \leftarrow seg'_S \cup \{cn\} = \{v1, v3\} \cup \{v6\} = \{v_1, v_3, v_6\}$ and $seg_i.wSum \leftarrow seg_i.wSum' + cn.w = 15 + 8 = 23$. Subsequently, we also have to keep track of the original $seg_i.S$, the best node set without v_6. Recall that we previously retain this information in T. Thus we simply assign the value of $T = \{v_1, v_3, v_5\}$ to $seg_i.S'$.

3.4.4 Future Trajectory Prediction

After the preceding processing is completed, we now present the final step that combines all the aforementioned

components in Algorithm 8. We identify the most probable future location Loc_p by first finding the similarity between the query trajectory T_p and its reference trajectories. The foregoing similarity value then serves as the support value for every possible future location. Finally the location with the highest support value is returned as the most probable future location Loc_p.

Algorithm 8. Future Trajectory Prediction

Input: the trajectory traveled by a user T_p.
Output: the most probable future location Loc_p.

```
1: Estimate total trip distance E(D|d_t). (Equation (5))
2: Obtain D_p by applying logarithmic decay. (Equation (6))
3: Ref ← Reference trajectories retrieved from the grid index
   in the BPN form obtained through ObtainBPN(Tr_t) of
   Algorithm 4
4: T_p ← ObtainBPN(T_p) (Algorithm 4)
5: d_p ← D_p − ε
6: while d_p ⩽ D_p + ε do
7:    for r ∈ Ref do
8:       r.sim ← FindSim(r, T_p) (Algorithm 5)
9:       r.locationAt(d_p).addSupport(r.sim)
10:   end for
11:   d_p.increment()
12: end while
13: return the most probable future location Loc_p that has the
    highest support value
```

In Algorithm 8, we first estimate total trip distance $E(D|d_t)$ through Equation (5). And then the length of future trajectory D_p is obtained with $E(D|d_t)$ and Equation (6). We then retrieve all the reference trajectories in the BPN form from the inverted grid index that is built offline. Our inverted index is a grid index that records the IDs of trajectories going through this grid. After that we encode query trajectory T_p in BPN too by Algorithm 4. Next we look for the most probable future location Loc_p in the distance range $d_p \in [D_p - \epsilon, D_p + \epsilon]$. ϵ is the range radius and we set it to 15 percent of D_p. During each iteration that increases d_p, we check all the reference trajectories and find their similarities with T_p and then update the support value of every respective future location. In the end we return the most probable future location Loc_p with the highest support.

4 Experimental Evaluation

This section is devoted to the assessment of performance of our algorithm compared with the three baselines, namely SubSynE, SubSynEA [15] and DESTPRE [18]. The basics of our experiment, comprising the introduction of dataset and the evaluation criteria, are described in Section 4.1. Then we present their running time and the effectiveness of responding to the queries posed by users in Sections 4.2 and 4.3. Specifically, the study on run-time efficiency concerns both the time for model training and query answering.

4.1 Experimental Settings

All the experiments are conducted on a desktop computer with 32 GB of memory and a quad core 2.7 GHz CPU.

Datasets. We test all the algorithms on a real-world dataset that is openly available and on a synthetic one.

Real-World Data:

(1) Shanghai: This dataset [47] was collected from the city of Shanghai, China. It contains approximately 4,000 taxis cruising in the urban area over one month.
(2) Chengdu: This dataset contains the trajectories of over 14,000 taxis during August 2014 in the city of Chengdu, China. The total amount of GPS records exceeds 1.4 billion.
(3) San Francisco: This dataset [4] is a collection of GPS records of the real-time whereabouts of 514 taxis running in San Francisco Bay Area during a time span of 30 days.

The 3 datasets are used to evaluate both the prediction accuracy and training efficiency.

Synthetic Data. We generate one synthetic data set in the form of single-step Markov matrix filled with transition probabilities. This dataset is solely for the assessment of training efficiency. Its size corresponds to the respective granularity of a real-world dataset.

Evaluation Criteria. We evaluate all three methods in terms of efficiency, prediction accuracy and coverage. Efficiency refers to the training time of SubSynE and our approach. Since the training stage for DESTPRE involves building a hash and a tree index which takes considerably longer time, we do not include DESTPRE in this comparison. Moreover, all three methods spend acceptable time for modern commodity machines during the prediction phase. The average deviation distance (measured in KM) of the top-3 prediction results from the true destination is used to assess prediction accuracy. Coverage takes the value between 0 and 1. It indicates the percentage of reported prediction results with regard to all the user requests.

For DESTPRE, we set the hyperparamenters recommended by its authors. Its three hyperparamenters are given as follows: matching threshold $\epsilon = 500$ m, similarity threshold $\theta = 0.15$, radius of start region $r = 500$ m. For SubSyn series algorithms, we simulate the real-world scenarios where the match ratio is $\tau = 0.57$, according to the authors of SubSyn. Furthermore, the number of predicted destinations k is set to 3.

4.2 Evaluation of Online Prediction

Our solution needs the incorporation of future trajectory prediction to locate the most likely future position Loc_p and thus takes extra time. This trade-off is favorable since it additionally incurs a fraction of a second (around an extra 65 ms in most cases) but vastly improves the accuracy. The succeeding step of destination prediction can be performed very fast (less than 1 ms) as it simply retrieves the probabilities for comparison. The dominant factor of these algorithms is the training time which sets our algorithm apart from its competitors. Therefore, we do not compare the run time of the online phase of all the algorithms.

Accuracy Evaluation. Two specific locations in the course of a trip—the 30 and 70 percent completion point—are used as the criteria. Because they indicate how well an algorithm will fare soon after a traveler begins his trip or soon before the trip ends. The grid granularity is also of our concern since it correlates strongly with the effectiveness of our approach. Besides we also alter the completion percentage of trip and the match ratio of identical trajectories (shown in Fig. 7). Here the identical trajectories refer to those in the testing dataset that are perceived as the same with the ones in the training dataset. Judged against the yardstick of this

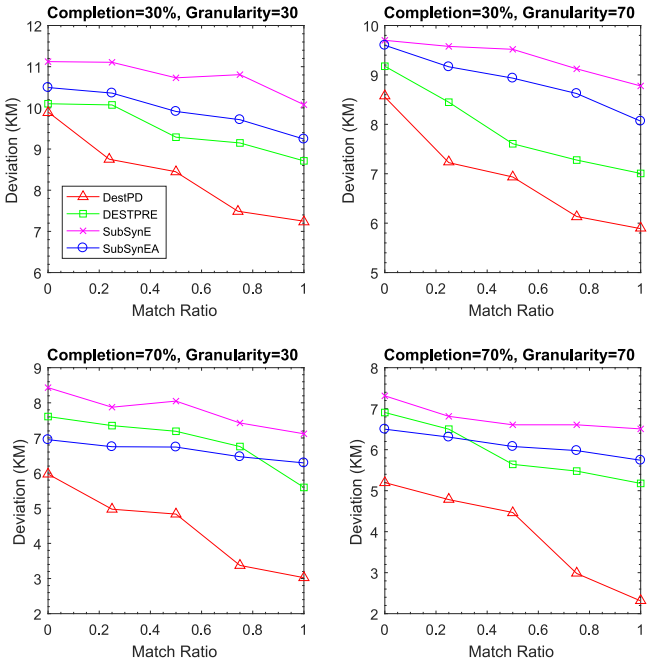


Fig. 7. Prediction accuracy varies with different match ratios.

ratio, all the algorithms can be examined from the perspective that reflects their ability to deal with the recurring historical data (i.e., exact matches, the ratio approaches 1) as well as generalize to completely new scenarios (i.e., routes that emerge for the first time, the ratio approaches 0).

30 - 70 percent Completion Points. We compute the mean of the deviation distances of the top three destinations returned by the algorithms. Our solution outperforms SubSynE and SubSynEA in terms of prediction accuracy quantified by this measure (Fig. 8). The granularity of a map plays an essential role in improving the accuracy though this effect gradually dwindles as the map becomes more fine-grained. Moreover, SubSynEA (second-order model) produces much better prediction results than SynSynE (first-order model) in the more coarse-grained settings. This distinction slowly fades away as the granularity increases, which is in agreement with our earlier analysis that the second-order model is prone to degradation since the rising amount of cells in a map obscures the distinction of two geographically isolated locations.

Different Completion Points. The ability to pinpoint the cause of variation (OUP and TAA) and to only re-compute the affected probabilities makes our approach efficient and accurate. This synergy gives rise to the definite edge of our approach over its competitors in terms of prediction accuracy. The advantage is particularly evident during the course starting from the 25 percent completion point and ending at the 85 percent completion point, the primary stage for location prediction (Fig. 8). The potential opportunities for various tasks, such as POI recommendation and advertising, abound especially in this course.

The second-order Markov model underlying SubSynEA enhances the prediction accuracy at the expense of a substantial increase in transition states which are offset to some extent by sparse matrix multiplication. Moreover, SubSynEA excludes the consideration for detour distances favoring the simplification of the model. Rather than employing a high-order Markov model, we stick with the first-order

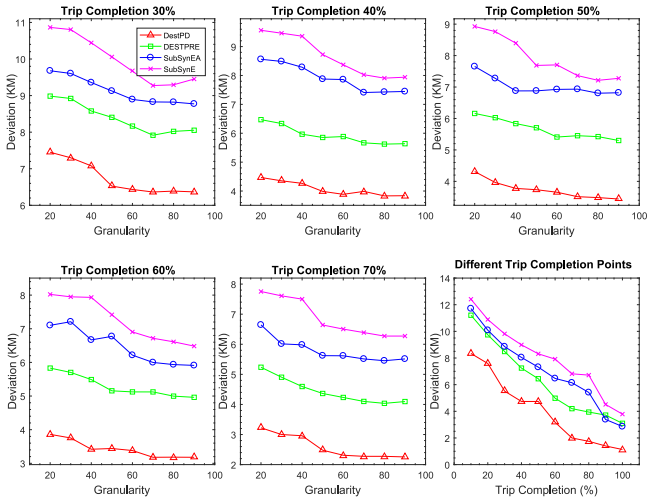


Fig. 8. Average deviation from destination at the 30-70 percent completion point and at the different trip completion points.

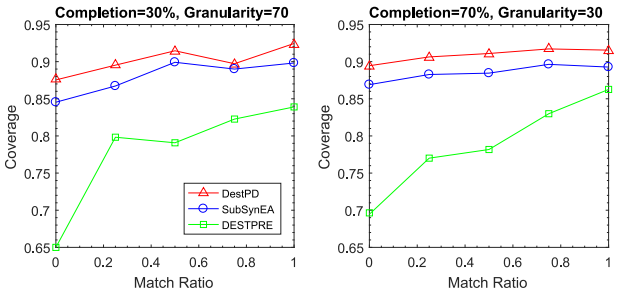


Fig. 9. Coverage varies with different match ratios.

model and apply the future path prediction algorithm first to discover the most probable future location. We find that our solution outperforms its competitors since the route picked by a user should be best described by the model focusing on the trajectory itself as opposed to several discrete Markov states.

The lack of a proper way handling the user-chosen route undermines the chances of right prediction of SubSynE. SubSynEA attempts to remedy this problem by only additionally considering the nearest historical location. However, a similar issue will arise regarding this technique that the itinerary traveled so far is still partially represented by merely three locations. The effectiveness of this strategy gradually diminishes as the granularity of map becomes less coarse. The distinction of the two states associated with the neighboring region of current location gets increasingly blurred, implicating that SubSynEA has the inherent propensity to fall back into SubSynE in fine-grained settings. This is in line with its prediction accuracy under such conditions in our experiment.

Coverage Analysis. Coverage measures how well an algorithm can respond to user queries in previously unknown cases. Shown in Fig. 9 is the coverage comparison between all the algorithms. It is apparent that both SubSynEA and our algorithm can achieve higher coverage than DESTPRE. Particularly when the matching ratio is low, DESTPRE can only respond to a very limited amount of user queries. Even when the matching ratio approaches 1, DESTPRE still attains lower coverage compared with SubSynEA and our algorithm. This can be ascribed to the mechanism employed

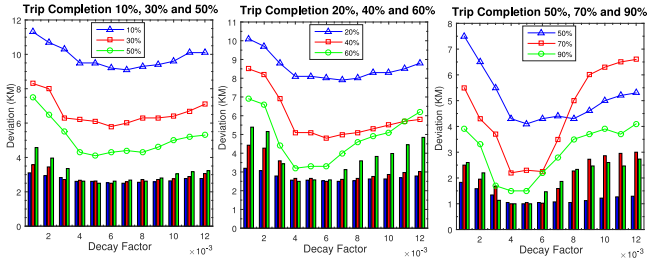


Fig. 10. The effect of decay factor.

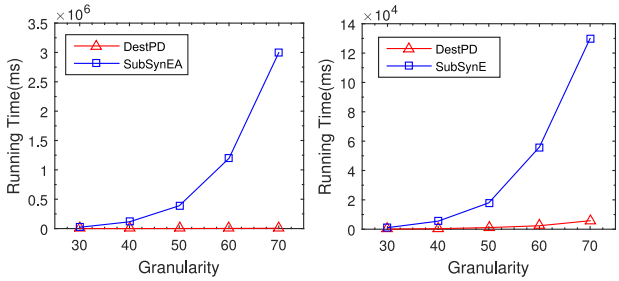


Fig. 11. Training time of DestPD, SubSynEA, and SubSynE.

by DESTPRE which predicts destinations only based on partial trajectory matching. Our algorithm also leverages this strength via a future trajectory prediction framework. But our method also generalizes very well to completely new scenarios by treating each segment of a trajectory separately through Bayesian inference. In reality, the combination of different routes can lead to a rapid surge of the number of possible trajectories. Therefore, it can be anticipated that our algorithm (based on statistical reasoning) will perform much better than DESTPRE in terms of coverage.

The Effect of Decay Factor. We discussed in the earlier section that the decay factor determines the speed of decline of the predicted percentage. Fig. 10 plots the deviation against the decay factor. The bars at the bottom of this chart, with three bars regarding their respective completion point in a group, indicate the resulted difference between one decay factor and the one that yields the least deviation.

Our experiment shows that, at the earlier part of a trip, a larger decay factor is preferred. As the trip gradually draws to its end, a smaller α makes a more favorable choice. We set this parameter to 0.004 to strike the right balance.

4.3 Evaluation of Offline Training

Efficiency of Training Algorithm. As is shown in Fig. 11, it is apparent that our approach is superior to SubSynE and SubSynEA as the granularity rises. This is particularly true once the map becomes more fine-grained. Our approach is 16.1 and 348.5 times faster than SubSynE and SubSynEA respectively when we set the granularity to 50. The zero entries most of the time constitute far beyond 50 percent of the elements of transition matrix for SubSynE and SubSynEA, and impose a burden on the overall efficiency. For every transition step, these zeros just repeatedly yield more zero entries that do not contribute to the computation of non-zero probabilities. The inadvertent inclusion of detour distances, dictated by matrix multiplication, significantly reduces the performance of SubSyn series algorithms.

5 Conclusion

In this paper we propose an efficient scheme for destination prediction that mainly involves the offline optimization of Markov transition matrix multiplication through ETP and TPD, the feasible online frequent update method for our model and the future trajectory prediction framework. The underlying rationale of ETP and TPD is thoroughly investigated in light of matrix multiplication operation. The reduction in time and space complexity is rigorously proved. In our future trajectory prediction framework, we design an efficient strategy to find similarity between two trajectories through BPN. Our new method is particularly suitable for two trajectories that have overlap. We also devise a dynamic programming technique to select a subset of nodes that are considered important.

Our method is efficient and effective. Experiments are carried out on real-world and synthetic datasets, which indicate that our method attains higher coverage, runs an order of magnitude faster and gains an increase of around 23 percent in accuracy, compared with the state-of-the-art approaches. Moreover, we believe that our novel BPN form of trajectories for finding similarity and node selection technique can be very useful for other work in the related domains.

Acknowledgments

Zhou Yang and Heli Sun are co-first authors. The authors would like to thank Dr. Yu Zheng for his thoughtful comments on this paper. The work was supported in part by the National Science Foundation of China grants 61672417, 61472299, 61876138, 61702405, 61772091 and 61802035, the Fundamental Research Funds for the Central Universities of China, Natural Science Basic Research Plan in Shaanxi Province of China grants 2017JM6045, and Science and Technology Foundation of Shenzhen City grants JCYJ20170816100845994. Any opinions, findings, and conclusions expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Y. Zheng, C. Mascolo, and C. T. Silva, "Guest editorial: Urban computing," *IEEE Trans. Big Data*, vol. 3, no. 2, pp. 124–125, Jun. 2017.

[2] D. J. Patterson, L. Liao, K. Gajos, M. Collier, N. Livic, K. Olson, S. Wang, D. Fox, and H. Kautz, "Opportunity knocks: A system to provide cognitive assistance with transportation services," in *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, 2004, pp. 433–450.

[3] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. P. Sondag, "Adaptive fastest path computation on a road network: A traffic mining approach," in *Proc. Int. Conf. Very Large Data Bases*, 2007, pp. 794–805.

[4] Y. Ge, H. Xiong, A. Tuzhilin, K. Xiao, M. Gruteser, and M. Pazzani, "An energy-efficient mobile recommender system," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 899–908.

[5] T. M. T. Do and D. Gatica-Perez, "Contextual conditional models for smartphone-based human mobility prediction," in *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, 2012, pp. 163–172.

[6] J. Yuan, Y. Zheng, and X. Xie, "Discovering regions of different functions in a city using human mobility and POIs," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 186–194.

[7] L.-Y. Wei, Y. Zheng, and W.-C. Peng, "Constructing popular routes from uncertain trajectories," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 195–203.

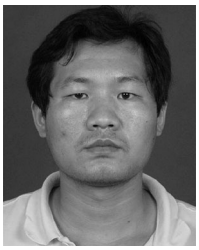
[8] A. Y. Xue, R. Zhang, Y. Zheng, X. Xie, J. Huang, and Z. Xu, "Destination prediction by sub-trajectory synthesis and privacy protection against such prediction," in *Proc. IEEE Int. Conf. Data Eng.*, 2013, pp. 254–265.

[9] J. Zheng and L. M. Ni, "Modeling heterogeneous routing decisions in trajectories for driving experience learning," in *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, 2014, pp. 951–961.
[10] J. Krumm, "Real time destination prediction based on efficient routes," No. 2006-01-0811, SAE Technical Paper, 2006.
[11] J. Krumm and E. Horvitz, "Predestination: Inferring destinations from partial trajectories," in *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, 2006, pp. 243–260.
[12] B. D. Ziebart, A. L. Maas, A. K. Dey, and J. A. Bagnell, "Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior," in *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, 2008, pp. 322–331.
[13] W. Mathew, R. Raposo, and B. Martins, "Predicting future locations with hidden Markov models," in *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, 2012, pp. 911–918.
[14] A. Y. Xue, R. Zhang, Y. Zheng, X. Xie, J. Yu, and Y. Tang, "DesTeller: A system for destination prediction based on trajectories with privacy protection," *Proc. VLDB Endowment*, vol. 6, no. 12, pp. 1198–1201, 2013.
[15] A. Y. Xue, J. Qi, X. Xie, R. Zhang, J. Huang, and Y. Li, "Solving the data sparsity problem in destination prediction," *VLDB J.*, vol. 24, no. 2, pp. 219–243, 2015.
[16] J. Krumm, R. Gruen, and D. Delling, "From destination prediction to route prediction," *J. Location Based Services*, vol. 7, no. 2, pp. 98–120, 2013.
[17] E. Horvitz and J. Krumm, "Some help on the way: Opportunistic routing under uncertainty," in *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, 2012, pp. 371–380.
[18] M. Xu, D. Wang, and J. Li, "DESTPRE: A data-driven approach to destination prediction for taxi rides," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2016, pp. 729–739.
[19] J. A. Alvarez-Garcia, J. A. Ortega, L. Gonzalez-Abril, and F. Velasco, "Trip destination prediction based on past GPS log using a hidden Markov model," in *Proc. Annu. Eur. Conf. Algorithms*, 2010, pp. 8166–8171.
[20] J. Alvarez-Lozano, J. A. García-Macías, and E. Chávez, "Learning and user adaptation in location forecasting," in *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, 2013, pp. 461–470.
[21] D. Ashbrook and T. Starner, "Using GPS to learn significant locations and predict movement across multiple users," *Pers. Ubiquitous Comput.*, vol. 7, no. 5, pp. 275–286, 2003.
[22] C. Zaiben, S. H. Tao, and Z. Xiaofang, "Discovering popular routes from trajectories," in *Proc. IEEE Int. Conf. Data Eng.*, 2011, pp. 900–911.
[23] C. Cheng, R. Jain, and E. van den Berg, "Location prediction algorithms for mobile wireless systems," in *Wireless Internet Handbook*. Boca Raton, FL, USA: CRC Press, 2003, pp. 245–263.
[24] J. Froehlich and J. Krumm, "Route prediction from trip observations," No. 2008-01-0201, SAE Technical Paper, 2008.
[25] X. Gao, B. Firner, S. Sugrim, V. Kaiser-Pendergrast, Y. Yang, and J. Lindqvist, "Elastic pathing: Your speed is enough to track you," in *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, 2014, pp. 975–986.
[26] G. Gidófalvi and F. Dong, "When and where next: Individual mobility prediction," in *Proc. ACM SIGSPATIAL Int. Conf. Advances Geographic Inf. Syst.*, 2012, pp. 57–64.
[27] H. Jeung, Q. Liu, H. T. Shen, and X. Zhou, "A hybrid prediction model for moving objects," in *Proc. IEEE Int. Conf. Data Eng.*, 2008, pp. 70–79.
[28] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti, "WhereNext: A location predictor on trajectory pattern mining," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 637–646.
[29] H. S. Samet Ayhan, "Aircraft trajectory prediction made easy with predictive analytics," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 21–30.
[30] H. Yin, Y. Sun, B. Cui, Z. Hu, and L. Chen, "LCARS: A location-content-aware recommender system," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 221–229.
[31] B. Liu, H. Xiong, S. Papadimitriou, Y. Fu, and Z. Yao, "A general geographical probabilistic factor model for point of interest recommendation," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 5, pp. 1167–1179, May 2015.
[32] Y. Liu, C. Liu, B. Liu, M. Qu, and H. Xiong, "Unified point-of-interest recommendation with temporal interval assessment," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1015–1024.
[33] Q. Liu, Y. Ge, Z. Li, E. Chen, and H. Xiong, "Personalized travel package recommendation," in *Proc. IEEE 11th Int. Conf. Data Mining*, 2011, pp. 407–416.
[34] J. J.-C. Ying, W.-C. Lee, T.-C. Weng, and V. S. Tseng, "Semantic trajectory mining for location prediction," in *Proc. 19th ACM SIGSPATIAL Int. Conf. Advances Geographic Inf. Syst.*, 2011, pp. 34–43.
[35] T. M. T. Do and D. Gatica-Perez, "Where and what: Using smartphones to predict next locations and applications in daily life," *Pervasive Mobile Comput.*, vol. 12, pp. 79–91, 2014.
[36] Y. Wang, N. J. Yuan, D. Lian, L. Xu, X. Xie, E. Chen, and Y. Rui, "Regularity and conformity: Location prediction using heterogeneous mobility data," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 1275–1284.
[37] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 1082–1090.
[38] W. Luo, H. Tan, L. Chen, and L. M. Ni, "Finding time period-based most frequent path in big trajectory data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 713–724.
[39] H. Su, K. Zheng, J. Huang, H. Jeung, L. Chen, and X. Zhou, "CrowdPlanner: A crowd-based route recommendation system," in *Proc. IEEE Int. Conf. Data Eng.*, 2014, pp. 1144–1155.
[40] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, 1962, Art. no. 345.
[41] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *Proc. 3rd Int. Conf. Knowl. Discovery Data Mining*, 1994, pp. 359–370.
[42] Y. Sakurai, M. Yoshikawa, and C. Faloutsos, "FTW: Fast similarity search under the time warping distance," in *Proc. 24th ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2005, pp. 326–337.
[43] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *Proc. 18th Int. Conf. Data Eng.*, 2002, pp. 673–684.
[44] L. Chen and R. Ng, "On the marriage of Lp-norms and edit distance," in *Proc. 13th Int. Conf. Very Large Data Bases*, 2004, pp. 792–803.
[45] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2005, pp. 491–502.
[46] K. Toohey and M. Duckham, "Trajectory similarity measures," *SIGSPATIAL Special*, vol. 7, no. 1, pp. 43–50, 2015.
[47] G. Dai, J. Huang, S. M. Wambura, and H. Sun, "A balanced assignment mechanism for online taxi recommendation," in *Proc. IEEE Int. Conf. Mobile Data Manage.*, 2017, pp. 102–111.

Zhou Yang received the bachelor's degree in computer science from Xidian University, in 2014. He is working toward the PhD degree in the School of Electronic and Information Engineering, Xi'an Jiaotong University of China. His research interests include data mining, knowledge discovery, and smart city.

Heli Sun received the PhD degree in computer science from Xian Jiaotong University, in 2011. She is an associate professor with the Department of Computer Science and Technology, Xian Jiaotong University of China. She was a visiting scholar with the University of California, Los Angeles, from 2017 to 2018. Her research interests include graph mining, social network analysis, and smart city.

Jianbin Huang received the PhD degree in pattern recognition and intelligent systems from the Xidian University, in 2007. He is a professor with the School of Computer Science and Technology, Xidian University of China. He was a visiting scholar with the University of California, Santa Barbara, from 2017 to 2018. His research interests include data mining, big data analysis, and smart city.



Zhongbin Sun received the BS and PhD degrees in computer science from Xian Jiaotong University, Xian, China, in 2010 and 2017, respectively. He is currently an assistant professor with the Department of Computer Science and Technology, Xian Jiaotong University. His research focuses on software engineering and data mining.

Hui Xiong received the BE degree from the University of Science and Technology of China, the MS degree from the National University of Singapore, and the PhD degree from the University of Minnesota. He is currently a professor and vice chair of the Management Science and Information Systems Department, and the director of the Rutgers Center for Information Assurance, Rutgers, the State University of New Jersey. His general area of research is data and knowledge engineering, with a focus on developing effective and efficient data analysis techniques for emerging data intensive applications. He has published prolifically in refereed journals and conference proceedings. He is a senior member of the ACM and IEEE.

Shaojie Qiao received the BS and PhD degrees from Sichuan University, Chengdu, China, in 2004 and 2009, respectively. From 2007 to 2008, he worked as a visiting scholar with the School of Computing, National University of Singapore. He is currently a professor with the Chengdu University of Information Technology, Chengdu, China. His research interests include complex networks and trajectory data mining.

Ziyu Guan received the BS and PhD degrees in computer science from Zhejiang University, China, in 2004 and 2010, respectively. He worked as a research scientist with the University of California, Santa Barbara, from 2010 to 2012. He is currently a professor with the School of Information and Technology, Northwest University, China. His research interests include attributed graph mining and search, machine learning, expertise modeling and retrieval, and recommender systems.

Xiaolin Jia received the PhD degree in computer science from Xian Jiaotong University, Xian, China, in 2006. She is a senior engineer with the Department of Computer Science and Technology, Xian Jiaotong University. She was a visiting scholar with the Pennsylvania State University from 2010 to 2011. Her research interests include data mining and information retrieval.