DESTPRE : A Data-Driven Approach to Destination Prediction for Taxi Rides

Mengwen Xu **Dong Wang** **Jian Li**

Institute for Interdisciplinary Information Sciences (IIIS), Tsinghua University*
Beijing, China
{xmw12@mails, wang-dong12@mails, lijian83@mail}.tsinghua.edu.cn

ABSTRACT

With the wide use of mobile devices, predicting the destination of moving vehicles has become an increasingly important problem for location based recommendation systems and destination-based advertising. Most existing approaches are based on various Markov chain models, in which the historical trajectories are used to train the model and the *top-k most probable* destinations are returned. We identify certain limitations of the previous approaches. Instead, we propose a new data-driven framework, called DESTPRE, which is not based on a probabilistic model, but directly operates on the trajectories and makes the prediction. We make use of only historic trajectories, without individual identity information. Our design of DESTPRE, although simple, is a result of several useful observations from the real trajectory data. DESTPRE involves an index based on Bucket PR Quadtree and Minwise hashing, for efficiently retrieving similar trajectories, and a clustering on destinations for predictions. By incorporating some additional ideas, we show that the prediction accuracy can be further improved. We have conducted extensive experiments on real Beijing Taxi dataset. The experimental results demonstrate the effectiveness of DESTPRE.

ACM Classification Keywords

H.2.8. Database Applications: Spatial databases and GIS

Author Keywords

Destination prediction; Quadtree; Minhash; Historical trajectories.

INTRODUCTION

A rapid growing number of mobile devices users across the world has been witnessed. Modern mobile devices, such as smart phones or navigation systems, usually have build-in GPS receivers that can locate the users with high accuracy.

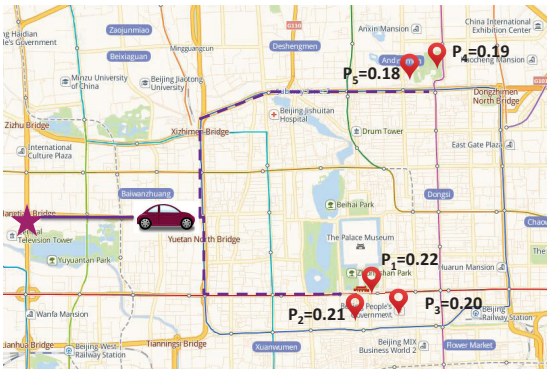


Figure 1. Destination Prediction Example. A vehicle's passed trip is in solid lines. Possible future routes (inferred from historical trajectories) are in dotted lines. One of these marked places may be the use's destination.

Such devices have produced a huge amount of location data, which can be utilized in a variety of location-based services (LBS) such as route finding, shopping or restaurant recommendation and location based social networks.

As the usage of LBS becomes part of many people's daily life, there has been an increasing demand for an accurate method of predicting a driver's destination as a trip progresses. Several applications such as location-based advertising can benefit from such a destination prediction method. For example, when a user is taking a taxi, we can collect the location information from her/his mobile phone or taxi's GPS device. Then, LBS provider predicts the most possible destinations and sends her/him advertisements about restaurants or hotels located in the neighborhood of the destinations, or to recommend sightseeing places [19]. Destination prediction can also potentially help developing route recommendation, navigation system and crowd anomalies detection. For example, a driver may want to know about traffic condition and make the right decision. The congested route can be avoided if the driver knows what most people's destinations are. In a navigation system, a prediction of a person's destination can help decide if the person is deviating from an intended route [10]. As a potential application, by predicting where many people go, the administrator may be able to predict the explosion of the size of the crowd in some places, and take corresponding preventive actions. Figure 1 illustrates an example of the destination prediction problem. Destination prediction is to predict the destination of a trip given a partial passed trajectory.

*The research is supported in part by the National Basic Research Program of China grants 2015CB358700, 2011CBA00300, 2011CBA00301, and the National NSFC grants 61033001, 61361136003.

UbiComp '16, September 12-16, 2016, Heidelberg, Germany
 ISBN 978-1-4503-4461-6/16/09$15.00
DOI: http://dx.doi.org/10.1145/2971648.2971664

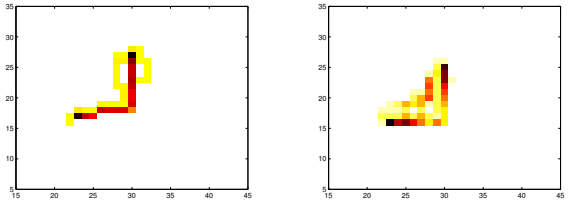
Existing Approaches

Due to the importance of the destination predication problem in the aforementioned applications, it has been studied extensively, and several methods have been proposed. We briefly review some of them here. Most existing approaches are based on various Markov chain (or hidden Markov chain) models [19, 7, 14, 1, 12]. One typical approach is to partition the region uniformly into the grid cells, or roads into segments, and use the cells or segments as the states of the Markov process [19, 1, 12]. The historical trajectories are used to train the transition probabilities of the Markov chain[1]. The Markov chain approaches are arguably natural, and have been empirically shown to be reasonably effective for the destination prediction problem. Adopting the (1st order) Markov chain model implies that the following implicit assumption is made: A vehicle travels in a memory-less random walk fashion. However, this obviously contradicts our intuition that a real trajectory is not totally random. To rectify this, Ashbrook et al. [19] proposed to use a modified probabilistic model, where they only retain those random walks that are not much longer than the shortest path. This clever trick clearly better captures a trajectory than a pure random walk. However, as we observed, the modified model still deviates from real trajectories. Figure 2(b) shows the heat map of the modified random walk trajectories using the transition probabilities computed as in [19]. The black cells are the start and destination cells. The darker the color is, the higher probability a trip contains the cell. Figure 2(a) shows the heat map of the real trajectories from the same start and destination cells. We can see clearly that they do not even look similar. Hence, we can conclude that the modified model [19] does not capture the real trajectories well.

Several previous work [10, 19, 22, 21, 1] used probabilistic inference to compute and return the top-k probabilities. When predicting the destination of an ongoing trip, the conditional probabilities of arriving at certain places are computed and the ones with the highest probabilities (i.e., the top-k most probable places) are returned as the prediction result. This method does not take destinations' geographic locations into consideration. It is quite likely that the probability for each place to be the destination is small and close to each other. The returned top-k places may be geographically very close to each other, but some other the places not so close with similar probabilities are ignored.

EXAMPLE 1. *Consider the example in Figure 1. There are 5 probable destinations denoted by position marks. The probabilities are respectively* $p_1 : 0.22, p_2 : 0.21, p_3 : 0.20, p_4 : 0.19, p_5 : 0.18$. *$P_1$, P_2 and P_3 are graphically close to each other, and P_4 and P_5 are graphically close to each other. If we were asked to return the top-2 places,* (P_1, P_4) *would be a better answer than* $(P1, P2)$ *(astute readers may have already realized that the prediction result can benefit from such geographic diversity).*

[1] In the Markov chain model (e.g., [12, 19, 1]), only the most recent state is useful because of the Markov property (i.e., the past states and future states are independent conditioning on the current state).



(a) (b)

Figure 2. (a) is the heatmap of real life trajectories. (b) is the heatmap of the modified random walk model with the same start and end point.

We stress that all information we use for the prediction in the paper is the historical trajectories and the partial trajectory of the present query. Some previous work [10, 4, 20] incorporated other informations such like ground cover, road conditions, and the identities. In particular, having identity information for private vehicles can be very useful for prediction since they tend to visit a few locations (home, office, school, etc.) very frequently, which can be identified easily from historical information of the same person. However, in this work, we investigate the predictive capability of trajectories, without using the identity information [2] So, the method in the paper is more suitable for taxi rides, or other applications where the identity information is not available.

Our Contributions

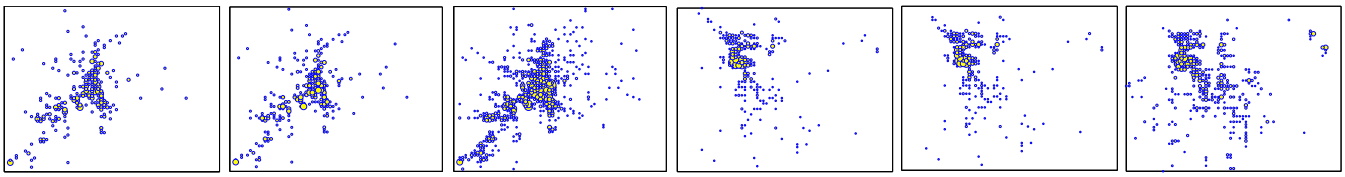
In this paper, we propose a new data-driven non-probabilistic approach to the destination predication problem. Our contributions and the organization of the paper are summarized as follows.

1. We first explain our design rationale and some observations that motivate our approach.
2. Then, we propose an indexing method based on BPR Quadtree and Minhash index, which can be used to retrieve similar trajectories efficiently and speed up the query time. We also define a new similarity measure more suitable for our problem, based on the Longest Common Subsequence similarity, to measure the similarity of (partial) trajectories.
3. We develop two destination prediction algorithms. The first algorithm makes predictions by only considering the frequency of the destinations of similar trajectories. The other algorithm performs a clustering on destinations of the similar trajectories in different diameter groups and makes the prediction based on the resulting clusters. We also provide an extension algorithm which can further enhance the accuracy of the prediction.
4. We conduct extensive experiments using real taxi trajectory dataset to evaluate the effectiveness and efficiency of our framework. The results show that our method performs more accurately and efficiently than previous methods.

OTHER RELATED WORK

Destination prediction problem can be classified into two categories, individual oriented and general oriented. Individual oriented offers personalized destination prediction while

[2] In our Beijing Taxi data, the passengers' identities are not available.



(a) (b) (c) (d) (e) (f)

Figure 3. Illustration of Observation 1. (a) shows the distribution of the destination locations with a specific prefix. (b) shows another one with a similar prefix. (c) shows that of all the prefixes similar to the first one. (d), (e), (f) show another instance. The size of each point indicates its frequency.

general oriented considers a query trajectory from an unknown. Several previous work on the destination predication problem made use of individual's identities. They trained the model by an individual's historical trajectories and predicted the destination for the same individual. Markov Chain model(including HMM) has been widely applied in individual-based destination prediction and route planning[1, 2, 13, 16, 11]. [17] proposed a method that uses similarity measure to identify the historical trajectory and predict the travel time. [5] used a pattern tree built from the historical movement patterns. [10] and [8] used Bayesian model which utilizes the travel time, travel efficiency and land category to inference the destination.

Several other work, including ours, do not assume the knowledge of the identity of the driver. However, many of them employed external information to enhance the prediction accuracy and even discarded the geographic information. Semantic information (e.g. restaurant, home etc.) was used in [4, 20] for mining the trajectory and predicting the destinations. The above studies mainly benefit from a specific setting of external information. However, our work assumes only the knowledge of historical trajectory information. We note that [19] made use the same information as we did and their approach was explained in the introduction.

DESIGN RATIONALE AND MOTIVATING OBSERVATIONS

Before we get into the details of DESTPRE, we first make some useful observations that motivate our approach. We hope these observations can provide useful insight in further study of the destination prediction problem and related problems.

As we mentioned in the introduction, we take a non-probabilistic approach. Nevertheless, we think that the real trajectories can still be captured by, or approximated by, a certain probabilistic process (which we refer to as the *underlying probabilistic process/model*). If the underlying process [3] (its structure and parameters) can be well learnt, the prediction problem would be a simple inference problem on this model. However, the underlying model may depend on many (hidden or explicit) parameters, [4] and is too complex to be described in full details. To improve over the existing approaches, one obvious thought would be to use a bigger model to better approximate the underlying probabilistic model. For example, one such proposal would be to use a higher-order Markov model, or to create a huge graphical model incorporating many more seemingly relevant parameters. But one can easily imagine that learning such a model is very challenging and expensive. Also it may require a huge amount of training data to fit the model. Therefore, instead of creating a better probabilistic approximation of the model, our approach is entirely data-driven (there is no probabilistic model). There are only a few hyper-parameters to choose. In the experimental section, we show how the results vary with different hyper-parameters and provide some guidance for choosing them. The rationale of our approach is based on the following two observations.

Observation 1. *If the prefixes of the trajectories are similar, the distributions of the geographic locations of their destinations also tend to be similar.*

Suppose we have observed the prefix $\mathsf{pref}(T)$ of a trajectory T. The underlying probabilistic model induces a conditional distribution $\Pr[\mathsf{dest}(T) \mid \mathsf{pref}(T)]$ where $\mathsf{dest}(T)$ is the location of the destination of T. So, to put Observation 1 in another word, if $\mathsf{pref}(T_1)$ is similar to $\mathsf{pref}(T_2)$ (we will formally define the similarity metric later), two distributions $\Pr[\mathsf{dest}(T_1) \mid \mathsf{pref}(T_1)]$ and $\Pr[\mathsf{dest}(T_2) \mid \mathsf{pref}(T_2)]$ are similar to each other.

See Figure 3 for an example extracted from the real dataset (the details of the dataset can be found in the experimental evaluation). We chose two similar prefixes. Figure 3(a) and 3(b) show respectively the distributions of the destinations of all trajectories with the two prefixes. We can see that they do look very similar. Figure 3(c) shows the destination distribution of the all trajectories with similar prefixes to the first one. The distribution again look very similar to the first two, except that it is denser. Figure 3(d), 3(e), and 3(f) show another instance.

Observation 2 *The destinations of trajectories with similar prefixes and diameters are clustered.*

While Observation 1 is certainly a very useful observation, we also observed that such distributions (i.e., $\Pr[\mathsf{dest}(T) \mid \mathsf{pref}(T)]$) often tend to be very scattered (see e.g., Figure 4(a)), which makes the prediction problem hard. Now, Observation 2 comes to rescue by saying that if we further restrict to the trajectories with similar diameters [5], the resulting distribution (i.e., $\Pr[\mathsf{dest}(T) \mid \mathsf{pref}(T), \mathsf{dia}(T) \approx \ell]$) is likely

[3] All previous Markov models can be thought as the approximations of this process.

[4] Our work concentrates on using only the trajectory information. There are other potentially relevant parameters (e.g., time, season, special events.) for the prediction problem.

[5] The diameter of a trajectory is the great circle distance between the start point and the destination point.

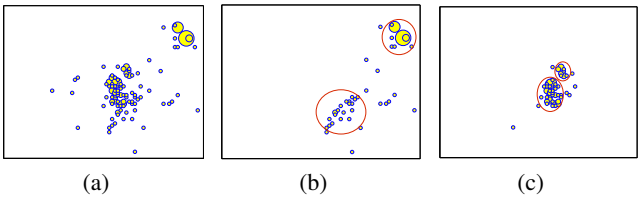


Figure 4. Illustration of Observation 2. (a) shows the destinations of similar prefixes. (b) and (c) show the destinations of similar trajectories in two different diameters groups.

to be clustered (see e.g., Figure 4(b) and Figure 4(c)). Intuitively, clustered distributions are much easier, as the cluster centers are naturally good predictors. Therefore, we can use the diameter of the ongoing trip to divide the historical trajectories in different groups.

When a vehicle is moving, its current trajectory can be obtained. If we can find the historical trajectories that are similar to the current trajectory, it is helpful to determine the remaining trip and the destination. Thus, destination prediction problem can be reduced to finding the trajectories that are similar to the given cell trajectory and extracting the reasonable destinations from them.

AN OVERVIEW OF DESTPRE

Now, we provide an overview of DESTPRE (see Figure 5). We first build the index at the offline stage. Then, DESTPRE can answer the query (a partial trajectory) from the client. More specifically, when a vehicle is moving, its GPS device sends the partial trajectory to the server, requesting for the predicted destinations. For the server, it needs to build the index at the offline stage and process the online query at the online stage. For the offline stage, first, we construct a BPR Quadtree and a uniform grid to split the map into cells. Then, we construct the index which is a collection of Minhash indices so as to find the possible similar trajectories quickly. For the online part, the client sends a partial trajectory to the server. When the server receives the partial trajectory, it searches the candidate trajectories from the index and filters out the similar trajectories using our similarity metric. At last, the server classifies the similar trajectories into different groups according to their diameters, and cluster the destinations of the similar trajectories by their geographic locations. The centers of all the clusters are returned as the predicted destinations to the client.

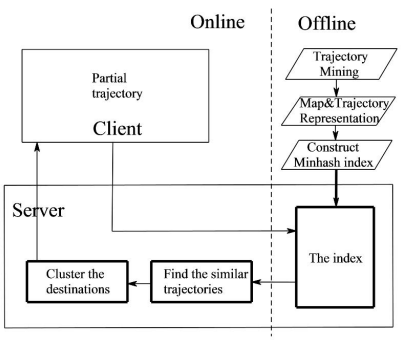


Figure 5. Framework Overview

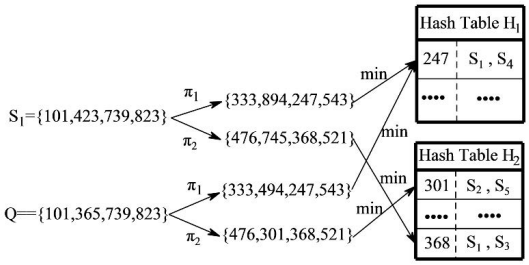


Figure 6. An example of the Minhash index.

PRELIMINARIES

We first provide some preliminary knowledge.

Bucket PR Quadtree

One important ingredient of DESTPRE is Bucket Point-Region Quadtree. See e.g., [15] for a detailed exposition of BPR Quadtree. BPR Quadtree is a variation of PR Quadtree, which is formed by recursively splitting the underlying space into four equal area blocks when the number of data points in each block reaches the bucket capacity τ. Each node of the BPR Quadtree corresponds to a block in the space. When the number of data points in a block reaches the bucket capacity τ, the leaf corresponding to the block becomes an internal node. All the points in it are rearranged into four children nodes.

BPR Quadtree can adapt to various densities of different regions. Higher density regions have finer cell subdivisions while rougher subdivisions are sufficient for lower density regions. This feature is particularly useful for our destination prediction purpose. More concretely, we can have a higher resolution for high density regions since there are more historical information about those regions and thus we can further refine them. In fact, we use a somewhat lower resolution for low density regions for two reasons: (1) there are fewer supporting historical data points, (2) the computation cost can be reduced significantly while the average accuracy is not affected much.

Minhash (Minwise hashing) Index

Minhash (or Minwise hashing)[3] is a class of Locality-sensitive hashing (LSH), which is used to quickly identify similar sets. Given a set $S \subseteq \Omega$, where $\Omega = \{1, 2, ..., D\}$, the Minhash family applies a random permutation $\pi : \Omega \to \Omega$ on S and stores only the smallest value after the permutation mapping. Formally, a Minhash function is defined as: $h_\pi(S) = \min_{s \in S}\{\pi(s)\}$, where π is a random permutation. For two subsets A and B of Ω, the probability of collision is the Jaccard similarity [3] $\Pr_\pi(h_\pi(A) = h_\pi(B)) = \frac{|A \cap B|}{|A \cup B|}$.

To quickly find similar sets from large data, we preprocess the sets by building an index. Specifically, we apply the Minhash function to the LSH preprocessing in [6]. The constructed index is called the Minhash index. It uses h different Minhash functions, where h is a fixed integer parameter. Each set S is represented by the h hash codes generated by the h Minhash functions, i.e., $h_{\pi_1}(S), \ldots, h_{\pi_\mathsf{h}}(S)$. For each Minhash function h_{π_i}, we create a table H_i composed of D buckets, each

corresponding to a hash code. In fact, each table H_i is implemented by a hash table. For set S, we store the pointer of S in the bucket $h_{\pi_i}(S)$ of H_i, for each $i \in [\mathsf{h}]$. We now describe the way to find the sets similar to a query set Q as follows. We compute the h hash codes $h_{\pi_1}(Q), \ldots, h_{\pi_\mathsf{h}}(Q)$. For all $i \in [\mathsf{h}]$, the sets placed in bucket $v_i = h_{\pi_i}(Q)$ of H_i are retrieved. These sets are candidates, and we compute their actual similarity with Q and return the true top ones.

EXAMPLE 2. *An example of constructing the Minhash index is shown in Figure 6. Specifically, given a set* $S_1 = \{101, 423, 739, 823\} \subseteq \Omega = \{1, 2, ..., 1000\}$ *and two Minhash functions* $h_1(X) = \min_{x \in X} \pi_1(x)$ *and* $h_2 = \min_{x \in X} \pi_2(x)$, *where* π_1 *and* π_2 *are two random permutations of* Ω. *After applying* π_1, *the elements of* S_1 *become* $\{333, 894, 247, 543\}$. *Therefore,* $h_1(S_1) = \min\{333, 894, 247, 543\} = 247$. *Similarly,* $h_2(S_1) = 368$. *Bucket 247 of hash table* H_1 *and Bucket 368 of hash table* H_2 *hold the pointer of* S_1. *For a set* $Q = \{101, 365, 739, 823\}$, $h_1(Q) = 247$, $h_2(Q) = 301$. *Then* S_1, S_4 *on Bucket 247 in* H_1 *and* S_2, S_5 *in Bucket 301 of* H_2 *are returned as the candidate sets. The Jaccard similarity of* S_1 *and* S_2 *is 0.75.*

TRAJECTORY REPRESENTATIONS AND SIMILARITY

Map and Trajectory Representations

The whole map is divided into two-dimensional cells according to the destination density. In other words, we utilize destination points of the historical trajectories to construct the BPR Quadtree and form the cells by inserting the destination points into the BPR Quadtree. Each leaf in the BPR Quadtree corresponds to a cell in the map. According to the property of the BPR Quadtree, if a certain region is a popular destination, the BPR Quadtree can provide a finer partition, which allows the algorithm to make more accurate estimation around the area. In fact, we do not want all the cells to be very small, which may lead to large trajectory representation, longer running time and larger memory usage. Thus, when constructing the BPR Quadtree, if the size of a cell is too small ($\leq 100m$), we do not split it.

We use $\mathscr{C}$ to denote the set of cells in the BPR Quadtree (called *quadtree cells*) and $\mathscr{G}$ to denote the set of cells in the uniform grid (called *grid cells*). We use the center of a (quadtree or grid) cell to represent all GPS points in the cell.

DEFINITION 1 (ORIGINAL TRAJECTORY). *An original trajectory* $T_v = (p_1, \cdots, p_n)$ *of a vehicle is a sequence of GPS points collected from one trip. Each point* p_i *consists of its latitude, longitude and a timestamp as* $(p_i.lat, p_i.lon, p_i.t)$.

DEFINITION 2 (QUADTREE CELL TRAJECTORY). *Given* $\mathscr{C}$ *and an original trajectory* T_v, *a quadtree cell trajectory* T_c *is a sequence of quadtree cells by mapping GPS points of* T_v *into quadtree cells in* $\mathscr{C}$. $T_c = (c_1, c_2, \cdots, c_m)$ *where each* $c_i \in \mathscr{C}$ *and* $c_j \neq c_{j+1}$, $\forall j \in [m-1]$.

We can also divide the map in uniform grid cells and use them to represent the trajectories. The uniform cell representation admits a rigorous analysis for the Minhash index (see Theorem 1).

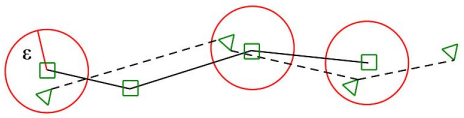


Figure 7. Maximum matching of two trajectories. The squares with dashed line is one trajectory, while triangles with solid line is another. The matching threshold is ε. $\mathsf{Maxm} = 3$ **in the example.**

DEFINITION 3 (UNIFORM CELL TRAJECTORY). *Given* $\mathscr{G}$ *and an original trajectory* T_v, *a uniform cell trajectory* T_u *is a sequence of cells by mapping GPS points of* T_v *into grid cells in* $\mathscr{G}$. $T_m = (g_1, g_2, \cdots g_m)$ *where each* $g_i \in \mathscr{G}$ *and* $g_j \neq g_{j+1}$, $\forall j \in [m-1]$.

The cell length of a cell trajectory $T_c = (c_1, c_2, \cdots, c_m)$ is the number of cells in it, i.e. the cell length $\mathsf{clen}(T_c)$ of T_c is m. The diameter of a cell trajectory is defined as the distance between the first cell and last cell. Let $\mathscr{T}$ be the set of historical cell trajectories. $\mathscr{T}$ is partitioned by their starting cells. We denote the set of trajectories starting from cell c by $\mathscr{T}_c$.

Trajectory similarity

Now we provide the similarity measurement that we use in this paper.

DEFINITION 4 (INCREASING MATCHING). *Let* $\mathrm{dist}(a, b)$ *be the great-circle distance (distance for short)* [6] *between two points a and b. Let* ε *be a positive constant, called* the matching threshold. *For two points* p_a *and* p_b, *if* $\mathrm{dist}(p_a, p_b) < \varepsilon$, *then the pair* (p_a, p_b) *can be matched together. Giving two sequences* $A = (a_1, a_2, \cdots, a_m)$ *and* $B = (b_1, b_2, \cdots, b_n)$, *we say a sequence with k pairs* $\{(a_{i_1}, b_{j_1}), (a_{i_2}, b_{j_2}), \cdots, (a_{i_k}, b_{j_k})\}$ *is an increasing matching, if* $i_1 < i_2 < \cdots < i_k$, $j_1 < j_2 < \cdots < j_k$ *and each* $(a_{i_d}, b_{j_d})\,(1 \leq d \leq k)$ *is a matching pair.*

An increasing matching M with the maximum length is called a maximum matching. The max-matching number is denoted by Maxm. See Figure 7 for an example.

We use the standard dynamic programming to compute the maximum matching between two trajectories (the dynamic programming is similar to the one for the longest common subsequence problem[18]). Now, we briefly explain the dynamic programming. Each cell in a cell trajectory can either be matched with a cell in another trajectory or remains unmatched. Let $A = (a_1, a_2, \cdots, a_n)$ and $B = (b_1, b_2, \cdots, b_m)$ be the given two cell trajectories. Note that when measuring the distance between the two cells, we use the center points of the two cells. Given ε as the matching threshold, we define the cardinality of the maximum matching between A and B as $\mathsf{Maxm}_\varepsilon(A, B)$. The dynamic programming recursion for computing the maximum matching of two cell trajectories is

$$\mathsf{Maxm}_\varepsilon(A,B) = \begin{cases} 0 \quad \text{if A or B is empty;} \\ 1 + \mathsf{Maxm}_\varepsilon((a_2,\ldots,a_n),(b_2,\ldots,b_m)), \\ \qquad\qquad \text{if } \mathrm{dist}(a_1,b_1) \leq \varepsilon; \\ \max \begin{cases} \mathsf{Maxm}_\varepsilon(A,(b_2,\ldots,b_m)); \\ \mathsf{Maxm}_\varepsilon((a_2,\ldots,a_n),B); \end{cases} \text{otherwise.} \end{cases}$$

[6] This is the shortest distance between two points on the surface of a sphere, measured along the surface of the sphere.

For our prediction purpose, we propose a new similarity measure between the partial cell trajectory and the historical cell trajectory, *Alternative Longest Common Subsequence* (ALCSS), as follows.

DEFINITION 5 (ALCSS DISSIMILARITY). *Given two cell trajectories T_a, T_b and the matching threshold ε, the ALCSS dissimilarity from T_a to T_b is defined as*

$$\mathsf{dsim}_\varepsilon(T_a,T_b) = 1 - \frac{\mathsf{Maxm}_\varepsilon(T_a,T_b)}{\mathsf{clen}(T_a)} \tag{1}$$

We note that ALCSS is a modified version of the Longest Common Subsequence similarity used in some previous work [17, 18]. One important difference is that dsim is asymmetric in a sense that $\mathsf{dsim}_\varepsilon(T_a,T_b)$ may not be the same as $\mathsf{dsim}_\varepsilon(T_b,T_a)$. The reason is that we want the $\mathsf{dsim}_\varepsilon(T_a,T_b)$ to be small even if T_a only matches well with a prefix of T_b.

DEFINITION 6 (SIMILAR TRAJECTORY). *A cell trajectory T_a is similar to another cell trajectory T_b with the matching threshold ε and the similarity threshold θ, if $\mathsf{dsim}_\varepsilon(T_a,T_b) \le \theta$.*

INDEX CONSTRUCTION AND RETRIEVAL

In this section, we investigate how to construct the index based on the Minhash index and retrieve similar trajectories.

Index Construction

Our index is a collection of Minhash indices. Each cell c in the map (a leaf in BPR Quadtree) contains a Minhash index to store the historical trajectories whose first cell is c. By regarding each trajectory as a set, finding similar trajectories corresponds to finding similar sets using Minhash index. For each cell c in the map, a Minhash index which uses h different hash functions is constructed using the method shown in the preliminary. When doing insertion, a set S_T is constructed according to the cell trajectory T. Then, for each $i \in [\mathsf{h}]$, we place the pointer of T into the bucket labeled $h_{\pi_i}(S_T)$ in hash table H_i.

As to insert the cell trajectory into Minhash index, we first show how to construct a set $S_T \subseteq \Omega$ called corresponding set (for the Minhash index) of a cell trajectory T. Then, we provide the algorithms for constructing the index.

Note that we should consider not only the cells in the cell trajectory, but also the ones that can potentially match them. Consider the quadtree trajectory $T_c = (c_1, c_2, \cdots, c_m)$. For each quadtree cell c_i in T_c, consider the disk D_i whose center is the center of c_i and radius is ε(usually ε is smaller than the side length of the grid cell). Let G_i be those grid cells that intersect D_i. Note that G_i covers all quadtree cells that can potentially match c_i. The whole cell trajectory is transformed into the set $S = G_1 \cup G_2 \cup \cdots \cup G_m$. An example of constructing the corresponding set for a trajectory is shown in Figure 8. Given a partial quadtree cell trajectory, the centers of the quadtree cells are a, b, c. They are mapped into points a', b', c'. Then the corresponding set of this partial trajectory is the set of grid cells in the upper layer. Now, we illustrate the construction of the index. We build a Minhash index $\mathscr{I}_c$ for each leaf $c \in \mathscr{C}$. I_c is used to index the historical cell trajectories in $\mathscr{T}_c$ (those starting from quadtree cell c). All the historical cell trajectories are inserted in the index while traversing the BPR Quadtree.

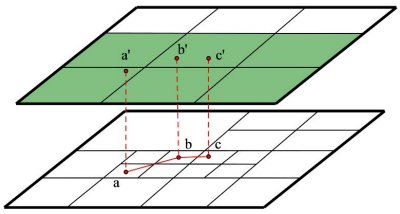


Figure 8. Illustration of constructing the corresponding set for a quadtree cell trajectory. The squares in the lower layer are the quadtree cells while the squares in the upper layer are grid cells (elements in the Minhash index).

Candidate Trajectory Retrieval

A candidate trajectory is a historical trajectory that may be similar to the partial trajectory. Now, we show how to retrieve candidate trajectories using the index.

Now we illustrate the details of the retrieval process. Given a partial trajectory, we only consider the Minhash indices for some cells near the starting location of the partial trajectory. We enlarge the starting location of the first cell s to a region $\mathsf{Region}(s,r)$ which contains cells that intersect with the disc centered at the first cell with the radius r. We can pre-compute a *neighboring cell list* $\mathscr{L}_c$ for each cell c, (here by saying n_c is a neighbor of c, we mean that $\mathrm{dist}(c,n_c) \le r$). Therefore, when we want to get the cells in $\mathsf{Region}(s,r)$, we only need to get the list $\mathscr{L}_s$. For a partial cell trajectory T_q, we search the Minhash indices in the cells in $\mathsf{Region}(s,r)$. Note that the methods of set construction for a partial cell trajectory and historical cell trajectories are different.

Next, we give a theorem to explain why we can use Minhash to index the similar trajectories. Theorem 1 and Corollary 1 demonstrate the relationship between the similar trajectories and the trajectories retrieved from the Minhash index. If a trajectory is similar to a partial trajectory, it can be retrieved from the index with a high probability.

THEOREM 1. *Suppose the cell trajectories are presented by uniform grid cells in $\mathscr{G}$, and the similarity threshold is θ. Given two cell trajectories Q and T, where T is a similar trajectory of Q, when we query the similar trajectories of Q in a Minhash index, the probability that T can be retrieved from one hash table is at least $\frac{1-\theta}{\theta+3\alpha}$, where $\alpha = \frac{\mathsf{clen}(T)+2}{\mathsf{clen}(Q)}$, $\mathsf{clen}(T)$ is the number of cells in T.*

PROOF. Given a cell trajectory $A = (a_1, a_2, \cdots, a_m)$, $a_i \in \mathscr{G}$, $i \in [m]$, the grid cell set $\mathscr{S}_A$ of A is a set of grid cells in A, defined as $\mathscr{S}_A = \{a_1, a_2, ..., a_m\}$. Then the grid cell sets of T and Q are $\mathscr{S}_T$ and $\mathscr{S}_Q$. Recall that the dissimilarity for a partial trajectory Q and a trajectory T is defined as $\mathsf{dsim}(Q,T) = 1 - \frac{\mathsf{Maxm}(Q,T)}{\mathsf{clen}(Q)}$. $\mathsf{dsim}(Q,T) \le \theta$ implies that

$$\begin{aligned}\mathsf{Maxm}(Q,T) &= (1-\mathsf{dsim}(Q,T)) \times \mathsf{clen}(Q) \\ &\ge (1-\theta) \times \mathsf{clen}(Q).\end{aligned}$$

T has been transformed into a set S_T for Minhash index. Recall that S_T is the union of all the elements in $\mathscr{S}_T$ and the 8

neighboring cells of each element. Thus, $|S_T| \leq 3|\mathscr{S}_T| + 6$ (this is because there are $|\mathscr{S}_T|$ elements in $\mathscr{S}_T$ and at most $2|\mathscr{S}_T| + 6$ neighboring cells. Hence, S_T contains at most $3|\mathscr{S}_T| + 6$ cells). Then, given query Q, the probability of finding the similar trajectory using in one hash table is

$$\Pr(h_\pi(\mathscr{S}_Q) = h_\pi(S_T)) = \frac{|\mathscr{S}_Q \cap S_T|}{|\mathscr{S}_Q \cup S_T|} = \frac{|\mathscr{S}_Q \cap S_T|}{|\mathscr{S}_Q| + |S_T| - |\mathscr{S}_Q \cap S_T|}$$

Note that $\mathsf{clen}(T) = |\mathscr{S}_T|$. Suppose $\mathsf{Maxm}(Q,T) = m$, we can see that $|\mathscr{S}_Q \cap S_T| = m$ and

$$|\mathscr{S}_Q| + |S_T| - |\mathscr{S}_Q \cap S_T| \leq \mathsf{clen}(Q) + 3\mathsf{clen}(T) + 6 - m.$$

Hence, we have that

$$\Pr(h_\pi(\mathscr{S}_Q) = h_\pi(S_T)) \geq \frac{(1-\theta) \times \mathsf{clen}(Q)}{\theta \times \mathsf{clen}(Q) + 3\mathsf{clen}(T) + 6} = \frac{1-\theta}{\theta + (3\mathsf{clen}(T) + 6)/\mathsf{clen}(Q)} = \frac{1-\theta}{\theta + 3\alpha}$$

□

COROLLARY 1. *The probability that a similar trajectory can be retrieved from the index is at least* $1 - (1 - \frac{1-\theta}{\theta+3\alpha})^{\mathsf{h}}$, *where* h *is the number of hash functions.*

PREDICTION ALGORITHMS

In this section, we present two destination prediction algorithms to predict destinations of using similar trajectories based on the index.

Destination Frequency based Algorithm

Before we introduce our prediction algorithm in DESTPRE , we first propose a destination frequency based algorithm FREQ as a comparison. Suppose the set of the destinations of the similar trajectories is $\mathscr{D}$. We calculate the frequency of each destination $d \in \mathscr{D}$. It is intuitive to predict destination by taking advantage of the frequency of the destinations. The predicted result is the most frequent destinations of $\mathscr{D}$.

First, in this algorithm FREQ, we retrieve the candidate trajectory set. Secondly, we use ALCSS as a filter to get the real similar trajectories from the possible ones. Then we get the destinations of the similar trajectories and calculate the frequency of each destination by maintaining a priority queue. Finally, the algorithm returns the top-k frequent destinations using the priority queue. If there is no similar trajectory in the index, we just return the current cell as the destination.

Diameter Cluster based Algorithm

In reality, similar trajectories have different diameters (Observation 2). This inspires us to propose the CLUSTER algorithm. It classifies the similar trajectories into different groups according to their diameters, and the destinations of similar trajectories are clustered by their geographic locations. The centers of all the clusters are returned as the predicted destinations.

Algorithm 1 The CLUSTER algorithm.

Input: The partial cell trajectory, $T_q = q_1, q_2, \cdots, q_m$; the matching threshold ε; the similar threshold θ; the diameter group number σ; the returned number k;
Output: The predicted destination set, S_d;

1: Initialize σ lists, say L_i, $i = 0, \cdots, \sigma - 1$;
2: Find candidate trajectory set $\mathscr{CT}$;
3: **for** each $T_h \in \mathscr{CT}$ **do**
4: **if** $\mathsf{dsim}_\varepsilon(T_q, T_h) \leq \theta$ **then**
5: Compute the diameter of T_q and T_h as $\mathsf{dia}(T_q)$ and $\mathsf{dia}(T_h)$;
6: Add the destination of T_h to L_i, $i = \lfloor \frac{\mathsf{dia}(T_q) \cdot \sigma}{\mathsf{dia}(T_h)} \rfloor$;
7: **end if**
8: **end for**
9: **for** each L_i **do**
10: Cluster the destinations in L_i and return the centers of each cluster;
11: Add the centers and q_m to S_d;
12: **end for**
13: **return** S_d;

We classify the similar trajectories by using the ratio between the diameter of the partial trajectory and the diameter of the similar trajectory to divide similar trajectories into different diameter groups, i.e., (0-50%,50-100%,100%). 100% is used for the case that the car has moved into the destination cell, but the trip has not finished. According to the diameter of the partial and the historical trajectories, similar trajectories could be classified into different diameter groups. The number of diameter groups is denoted by σ. In this paper, we set $\sigma = 2$. According to Observation 2, we know that the geographic distribution of the destinations of the similar trajectories in different diameter groups tends to be clustered. Therefore, instead of choosing the most frequent ones, we choose the cluster centers. Here we use the k-means algorithm for clustering. The number of centers k_c is determined by k (the returned number) and σ. Thus, the centers of the clusters for each diameter group and the current cell (for group 100%) are returned as the predicted destinations.

Algorithm 1 illustrates the details of the CLUSTER algorithm. Given a partial cell trajectory T_q, step 2 retrieves the candidate trajectory set $\mathscr{CT}$ of T_q. In step 4, we compute the similarity of T_q and each T_h in $\mathscr{CT}$ using ALCSS. If T_h is a similar trajectory, suppose the diameter of the T_q and T_h are $\mathsf{dia}(T_q)$ and $\mathsf{dia}(T_h)$, and the ratio between their diameters is $\mathsf{dia}(T_q)/\mathsf{dia}(T_h)$. Thus, T_h is added to the group with index $\lfloor \frac{\mathsf{dia}(T_q) \cdot \sigma}{\mathsf{dia}(T_h)} \rfloor$ in step 6. In step 9 to 13, we cluster the destinations of similar trajectories in each diameter group, and return the centers of each cluster and the current cell as the predicted destinations.

Further Improvement for $k = 1$

In this section, we consider the case in which we are required to return only $k = 1$ destination. We show that we can further improve the performance for the $k = 1$ case, by making

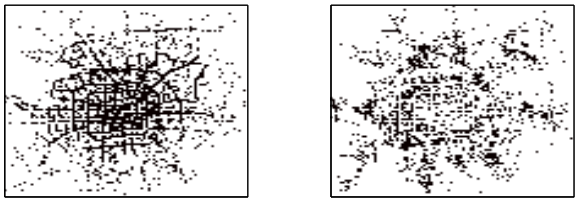


Figure 9. Figure on left shows the cells in which CLUSTER works better. Those appear to be the main roads of the city. Figure on right shows the cells that Naïve works better. Most of them are local small roads.

another useful observation. The new algorithm is denoted as CLUSTER ($k = 1$).

If the trip is nearly completed (the current place is very close to the destination), CLUSTER does not perform very well which can seen from Figure 10(a) in the experiment section. An interesting (yet somewhat trivial) observation is that the Naïve algorithm (predicting the last cell of the current trajectory) would be a very compelling algorithm in such case. However, we do not know what is the proportion of the trip that has been completed.

So, we proceed as follows. We evaluate the performance of CLUSTER and Naïve on the partial trajectories whose last GPS point located in each cell c. The algorithm performes better(above half of the test) on cell c is recorded. Then, given a partial cell trajectory with last location in cell c, if CLUSTER performes well, the Cluster algorithm is used. Otherwise, c is predicted as the destination. The intuition behind the CLUSTER ($k = 1$) algorithm can be seen from Figure 9. The cells for which CLUSTER performs better on are usually on the main road. On the other hand, if the last cells are not parts of the main road, it is more likely that the trip is to be completed soon, and predicting the current cell as the destination usually performs better.

EXPERIMENTAL EVALUATION

Experimental Setup

Dataset. In all experiments, we use the real dataset which consists of GPS points of 12000 taxis collected from October 1st to December 31st in 2012 and distributed in the urban area about $50km \times 50km$ in Beijing, China. [7] Since the dataset is about 90GB which is very large and the GPS points for different vehicles are mixed, we apply a MapReduce based secondary sorting algorithm to the feature "Car Id" and "Time Stamp" to generate sequences of GPS points for different taxis. Attribute "State" indicates whether the taxi is occupied or not. Therefore, we can use "State" to split the data into trajectories. A switch from the state "0 (unoccupied)" to "1 (occupied)" indicates that the taxi begins a new trip, and a reverse switch indicates the termination of a trip. We further clean the data by discarding the trips that are too short. The average length of the trajectories is about 11km.

After extracting the trajectory from the data, we randomly pick 1,000 trajectories from this dataset as the test set. The remaining ones are used to create an index storing about 11 million cell trajectories (which is of size approximately 1.97GB).

[7] http://www.datatang.com/data/45888

The algorithms are implemented in Java and run on a PC with Intel Core i5 CPU(3.2GHZ) and 16GB memory on Windows 7 platform. The number of hash tables for Minhash index is set to be 6. The grid granularity of $\mathscr{G}$ is set to be 100 (i.e., $\mathscr{G}$ contains 100×100 cells). The real geographic distance between two grid cells is about $500m$. The number of cells in $\mathscr{C}$ (the cells in the quadtree representation) is about 10000. Also, the smallest cell size is about $100m$. The hyperparameters used in the evaluation are the matching threshold ε, the similarity threshold θ and the radius of the starting region r.

Accuracy Evaluation

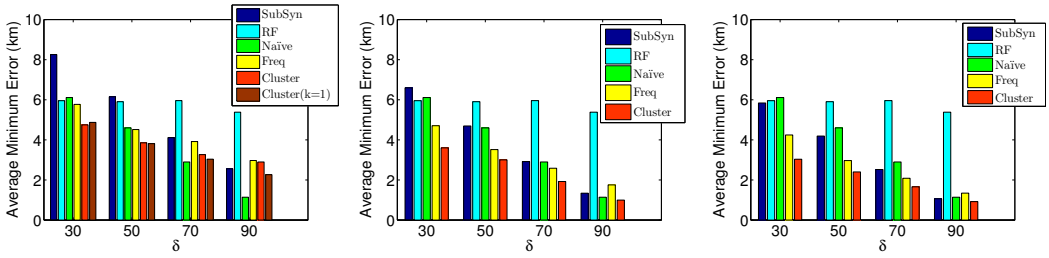
We compare the accuracy achieved by different prediction algorithms using our framework DESTPRE, the baseline algorithm, and the prior works.

We list the algorithms we compare against as follows:

- Naïve is the method that we predict the last cell of the partial trajectory as the destination. This shows the average remaining length of real trajectories.
- RF (Random Forests): It is one of the most popular of-the-shelf prediction algorithm. In our experiment, we use the following features: the start location, the travel time and the average driving speed of a current trip.
- SUBSYN[19]: This algorithm utilizes the same information of the data as ours. We implement SUBSYN with the grid granularity 100 which is the same with our setting. Because of the usage of large memory, it is run on another computer with 128GB memory.
- CLUSTER is the diameter cluster based algorithm, and FREQ is the destination frequency based algorithm. They are the prediction algorithms proposed in our paper. CLUSTER ($k = 1$) is an extension prediction algorithm as further improvement for $k = 1$.

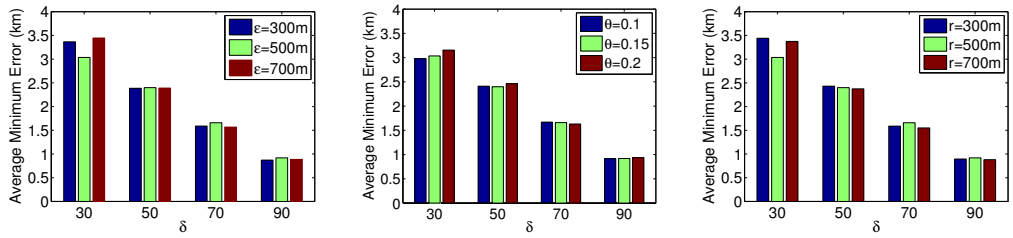
Performance metric: To evaluate the effectiveness of our method, we use *Average Minimum Error (*AvgMinErr*)* and *Distance-accuracy Value (*DAV*)* as our performance metrics. AvgMinErr is a measure to estimate the prediction error and the diversity performance of the algorithms. Suppose our test set Q has n queries. We define Average Minimum Error for all queries is the average of all errors over all n queries, i.e., $\mathsf{AvgMinErr} = \frac{1}{n}\sum_{i=1}^{n} \mathsf{MinErr}(q_i),\ q_i \in Q$, where $\mathsf{MinErr}(q_i)$ is the error of a single query q. Let the error of a single query q be the minimum L1 distance between any predicted destination cell d_{p_i} ($i \in [k]$) and the true destination cell d_r, which is $\mathsf{MinErr}(q) = \min_{i\in[k]} \mathsf{L}_1(d_{p_i}, d_r)$. For example, the L1 distances between the three predicted destinations and the true destination are $3km$, $4.5km$ and $1km$ respectively, then $\mathsf{MinErr}(q) = 1km$. It is likely that the true destination is among these three predicted destinations. We argue that AvgMinErr is an appropriate metric for our prediction task (especially when $k > 1$): [8] Consider a prediction result A with

[8] The metric has been widely used in several prediction problems in a variety of areas such as information retrieval, ad allocation (the allocation is good if the user clicks any ad), machine learning and computer vision (in image classification, such as the ImageNet competition, the result is correct if one (out of 3 or 5) classification result is correct). Using more sophisticated metrics (such as variants of DCG [9]) is left as an interesting future work.



(a) k=1 (b) k=3 (c) k=5

Figure 10. Average minimum error. (a),(b),(c) show the Average minimum error of different number of predicted destinations $k = 1,3,5$ ($\varepsilon = 500m$, $\theta = 0.15$, $r = 500m$). δ is the percentage of the trip completed so far.



(a) varying ε ($\theta = 0.15$,$r = 500m$) (b) varying θ ($\varepsilon = 500m$, $r = 500m$) (c) varying r ($\varepsilon = 500m$,$\theta = 0.15$)

Figure 11. Average minimum error for different hyperparameters ($k = 5$). (a),(b),(c) show the accuracy performance with various settings of the matching threshold ε, the similarity threshold θ and the radius of a start region r.

one very accurate point and others with fair accuracy, and another result B in which all predicted points are mediocre but the mean error is better. Arguably, A is more useful than B, especially for advertising/recommendation related applications. Furthermore, if we want to minimize the mean error, the optimal solution would be to return the *same* most frequent point k times (mean error does not reward diversity). So it does not make sense to consider the mean error for $k > 1$ (when $k = 1$, AvgMinErr is the same as the mean error though). In fact, we also did some experiments with the mean error metric. The results were not very informative (only suggested that higher k value is not useful for this metric). Due to the space limitation, we choose to omit those results here.

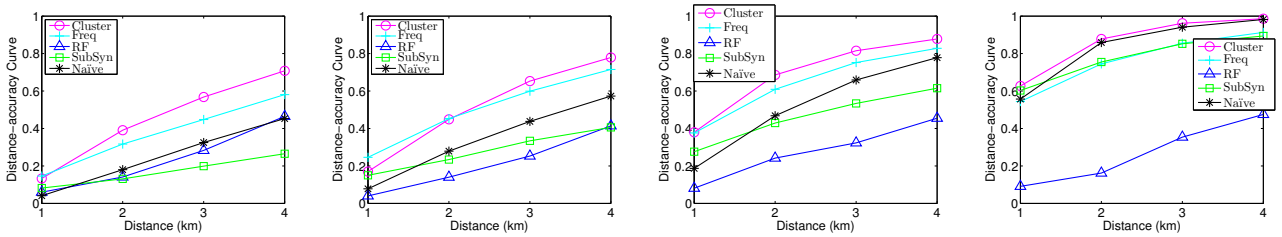
While AvgMinErr is a single average number to measure the prediction performance, DAV can provide more comprehensive distributional information. Formally, given a threshold λ, $\mathsf{DAV}(\lambda) = \frac{|Q_\lambda|}{|Q|}$ where $Q_\lambda = \{q \in Q, \mathsf{MinErr}(q) \leq \lambda\}$. In other words, $\mathsf{DAV}(\lambda)$ is the percentage of queries for which the error is no more than λ. For instance, suppose $\mathsf{DAV}(1\text{km}) = 40\%$. So if $1km$ is an acceptable error, then the algorithm makes correct predictions on 40% of the queries.

We define δ to be the percentage of the trip completed so far (the ratio between the length of current trip and the complete trip). k is defined as the number of predicted destinations. We evaluate AvgMinErr and DAV by varying $\delta = 30\%, 50\%, 70\%, 90\%$ and $k = 1,3,5$ for different algorithms.

Average Minimum Error

Figure 10 shows how AvgMinErr of different algorithms vary with δ and k (we return k locations). Here the matching threshold ε and the radius of the starting region r are set to be $500m$. The similarity threshold θ is 0.15. We can see significant improvement of AvgMinErr with the growth of θ. With $k = 1$, which means MinErr for a query is exactly the L1 distance between the real destination and the predicted destination, the CLUSTER ($k = 1$) algorithm performs better than the others. With $k = 3,5$, we observe that our proposed algorithms FREQ, CLUSTER achieve better accuracy than SUBSYN, RF. Among all those algorithms, CLUSTER provides the best accuracy. For example, with $k = 5$ and $\delta = 30\%$, AvgMinErr of SUBSYN is about $5.8km$, while that of CLUSTER is about $3km$ which is 52% better than the former one. All the figures in Figure 10 report that the performance gap is more significant when δ is smaller and CLUSTER always outperforms FREQ. This indicates the effectiveness of the clustering step in our framework.

Now we evaluate AvgMinErr of the CLUSTER algorithm under different hyperparameter settings with $k = 5$. The evaluated hyperparameters are ε, θ and r. The experimental results and the hyperparameter settings are shown in Figure 11. In Figure 11(a), it can be seen that $\varepsilon = 500m$ is a better choice according to the small AvgMinErr in general. When $\varepsilon = 300m$, the matching threshold is smaller than that of $\varepsilon = 500m$, so that it returns trajectories with less similarity and leads to a worse result. For the case $\varepsilon = 700m$, the matching threshold is large, which may consider irrelevant trajectories as similar ones. As Figure 11(b) shows, with the increase of θ from 0.1 to 0.2, AvgMinErr of $\theta = 0.2$ is larger



(a) $\delta = 30\%$ (b) $\delta = 50\%$ (c) $\delta = 70\%$ (d) $\delta = 90\%$

Figure 12. Distance-accuracy curve of $k = 3$ predicted destinations ($\varepsilon = 500m$, $\theta = 0.15$, $r = 500m$).

than that of the others. With larger θ, we get trajectories less similar to the query. The difference of AvgMinErr between $\theta = 0.1$ and $\theta = 0.15$ is not significant. In general, when $\delta > 30\%$, $\theta = 0.15$ is slightly better than $\theta = 0.1$. r controls the number of cells in the starting region. As shown in Figure 11(c), AvgMinErr first decreases and then increases with r from 300m to 700m when δ is 30%. The reason is that when $r = 300m$, less historical trajectories are considered, since the cell number is small. However, when $r = 700m$, the starting region is too coarse. Therefore, we choose $r = 500m$.

Distance-accuracy Value

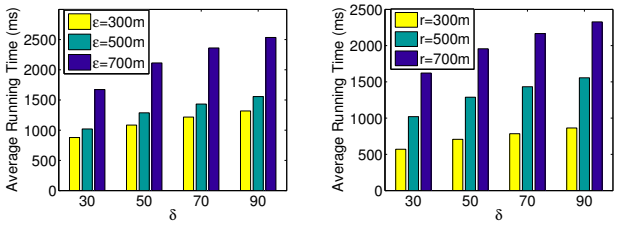
Note that larger DAV denotes better performance. As shown in Figure 12, DAV of CLUSTER or FREQ under each δ is larger than that of the Naïve, SUBSYN, RF algorithms. This implies that no matter how much δ is, with a specific threshold λ, the accurate rate of our proposed algorithms is larger than the others. CLUSTER algorithm leads ahead to the other algorithms. For example, with $k = 3$ and $\delta = 30\%$, $\mathsf{DAV}(2km) \approx 20\%$ for Naïve, $\mathsf{DAV}(2km) \approx 15\%$ for SUBSYN and RF, while $\mathsf{DAV}(2km) \approx 40\%$ for CLUSTER. Also, with $k = 5$ and $\delta = 50\%$, $\mathsf{DAV}(1km) \approx 30\%$ for CLUSTER.

Efficiency Evaluation

Index Efficiency. We test the time efficiency of CLUSTER. The measurement is the average running time per query. We evaluate the pruning capability of the index. Figure 14 shows the running time of CLUSTER and the algorithm without using the Minhash index (called "No-index") with $r = 500m$, $\theta = 0.15$, $\varepsilon = 500m$. As shown in Figure 14, when $\delta = 50\%$, the running time of CLUSTER is about $1.3s$ while that of No-index is about $2.3s$, which is nearly twice of the former one. This is because CLUSTER can prune most of the historical trajectories that are not possible to be the similar trajectories.

Running time for different ε. Figure 13(a) shows the running time of CLUSTER varying with ε under the condition that $\theta = 0.15$, $r = 500m$. The average running time of $\varepsilon = 300m$ and $\varepsilon = 500m$ are close, while that of $\varepsilon = 700m$ is far more than the others. This is because for $\varepsilon = 300m$ and $\varepsilon = 500m$, we choose the same grid granularity 100. While for $\varepsilon = 700m$, we set the grid granularity to 50. Smaller grid granularity leads to longer running time. The running time in fact does not directly depend on ε, but the grid granularity.

Running time for different r. When retrieving the similar trajectories, we consider the scale of the starting region. Under the condition that $\varepsilon = 500m$, $\theta = 0.15$, the efficiency of CLUSTER algorithm varying with different r is shown in Figure 13(b). We get more similar trajectories with larger r, which causes longer running time.



(a) varying ε ($\theta = 0.15$,$r = 500m$) (b) varying r ($\varepsilon = 500m$,$\theta = 0.15$)

Figure 13. Average running time for different hyperparameters. (a) and (b) show the average running times with various ε and r values.

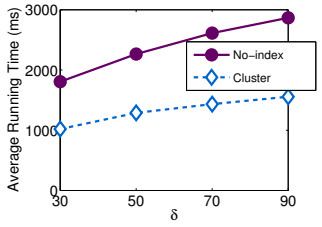


Figure 14. Index Efficiency ($r = 500m$, $\theta = 0.15$, $\varepsilon = 500m$).

CONCLUSDING REMARKS

In this paper, we propose DESTPRE, a data-driven method for predicting the destination of a partial trajectory. The design of DESTPRE is based on several interesting observations we made from the data. Our method is conceptually simple, has a high accuracy and can scale to fairly large dataset. Extensive experiments using real trajectory dataset have demonstrated that DESTPRE can efficiently predict the destination with lower error than existing methods.

We note that our method does not utilize other information like time, weather, and the identity of the driver/passenger. Hence, our method is more suitable for taxi rides, or when such information is not available. We believe incorporating such information can further increase the accuracy of the prediction and we leave it an important future direction. In addition, our indexing data structure stores all historical trajectories, which may be expensive in memory space usage. We plan to use compression or sampling techniques to further reduce the space usage.

REFERENCES

1. Juan Antonio Alvarez-Garcia, Juan Antonio Ortega, L Gonzalez-Abril, and Francisco Velasco. 2010. Trip destination prediction based on past GPS log using a Hidden Markov Model. *Expert Systems with Applications* 37, 12 (2010), 8166–8171.
2. Daniel Ashbrook and Thad Starner. 2003. Using GPS to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing* 7, 5 (2003), 275–286.
3. Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. 2000. Min-wise independent permutations. *J. Comput. System Sci.* 60, 3 (2000), 630–659.
4. Michelangelo Ceci, Annalisa Appice, and Donato Malerba. Time-Slice Density Estimation for Semantic-Based Tourist Destination Suggestion. In *ECAI (2010), 1107–1108*.
5. Ling Chen, Mingqi Lv, and Gencai Chen. 2010. A system for destination and future route prediction based on trajectory mining. *Pervasive and Mobile Computing* 6, 6 (2010), 657–676.
6. Aristides Gionis, Piotr Indyk, Rajeev Motwani, and others. 1999. Similarity search in high dimensions via hashing. In *VLDB*, Vol. 99. 518–529.
7. Ramaswamy Hariharan and Kentaro Toyama. Project Lachesis: parsing and modeling location histories. In *International Conference on Geographic Information Science (2004), 106–124*.
8. Eric Horvitz and John Krumm. Some help on the way: Opportunistic routing under uncertainty. In *UbiComp (2012), 371–380*.
9. Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.
10. John Krumm and Eric Horvitz. Predestination: Inferring destinations from partial trajectories. In *UbiComp (2006), 243–260*.
11. Julia Letchner, John Krumm, and Eric Horvitz. Trip router with individualized preferences (trip): Incorporating personalization into route planning. In *AAAI (2006)*, Vol. 21. 1795.
12. Mu Li, Amr Ahmed, and Alexander J Smola. Inferring movement trajectories from GPS snippets. In *WSDM (2015), 325–334*.
13. Lin Liao, Donald J Patterson, Dieter Fox, and Henry Kautz. 2007. Learning and inferring transportation routines. *Artificial Intelligence* 171, 5 (2007), 311–331.
14. Natalia Marmasse and Chris Schmandt. 2002. A user-centered location model. *Personal and ubiquitous computing* 6, 5-6 (2002), 318–321.
15. Hanan Samet. 2006. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann.
16. Reid Simmons, Brett Browning, Yilu Zhang, and Varsha Sadekar. Learning to predict driver route and destination intent. In *ITSC (2006), 127–132*.
17. Dalia Tiesyte and Christian S Jensen. Similarity-based prediction of travel times for vehicles traveling on known routes. In *SIGSPATIAL (2008), 14:1–14:10*.
18. Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *ICDE (2002), 673–684*.
19. Andy Yuan Xue, Rui Zhang, Yu Zheng, Xing Xie, Jin Huang, and Zhenghua Xu. Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. In *ICDE (2003), 254–265*.
20. Josh Jia-Ching Ying, Wang-Chien Lee, Tz-Chiao Weng, and Vincent S Tseng. Semantic trajectory mining for location prediction. In *SIGSPATIAL (2011), 34–43*.
21. Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. Driving with knowledge from the physical world. In *KDD (2011), 316–324*.
22. Brian D Ziebart, Andrew L Maas, Anind K Dey, and J Andrew Bagnell. Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. In *UbiComp (2008), 322–331*.