T-Drive: Driving Directions Based on Taxi Trajectories

Jing Yuan[1,2], Yu Zheng[2], Chengyang Zhang[3], Wenlei Xie[2]
Xing Xie[2], Guangzhong Sun[1], Yan Huang[3]
[1] School of Computer Science and Technology, University of Science and Technology of China
[2] Microsoft Research Asia, 4F, Sigma Building, No. 49 Zhichun Road, Beijing 100190, China
[3] Computer Science Department, University of North Texas
{yuzheng,xingx}@microsoft.com, yuanjing@mail.ustc.edu.cn, gzsun@ustc.edu.cn

ABSTRACT

GPS-equipped taxis can be regarded as mobile sensors probing traffic flows on road surfaces, and taxi drivers are usually experienced in finding the fastest (quickest) route to a destination based on their knowledge. In this paper, we mine smart driving directions from the historical GPS trajectories of a large number of taxis, and provide a user with the practically fastest route to a given destination at a given departure time. In our approach, we propose a time-dependent landmark graph, where a node (landmark) is a road segment frequently traversed by taxis, to model the intelligence of taxi drivers and the properties of dynamic road networks. Then, a Variance-Entropy-Based Clustering approach is devised to estimate the distribution of travel time between two landmarks in different time slots. Based on this graph, we design a two-stage routing algorithm to compute the practically fastest route. We build our system based on a real-world trajectory dataset generated by over 33,000 taxis in a period of 3 months, and evaluate the system by conducting both synthetic experiments and in-the-field evaluations. As a result, 60–70% of the routes suggested by our method are faster than the competing methods, and 20% of the routes share the same results. On average, 50% of our routes are at least 20% faster than the competing approaches.

Keywords

Driving directions, time-dependent fast route, taxi trajectories, T-Drive, landmark graph

Categories and Subject Descriptors

H.2.8. [**Database applications**]: Data mining, Spatial databases and GIS.

General Terms

Algorithms, Experimentation

1. INTRODUCTION

GIS '10, November 2-5, 2010. San Jose, CA, USA

Finding efficient driving directions has become a daily activity and been implemented as a key feature in many map services like Google and Bing Maps. A fast driving route saves not only the time of a driver but also energy consumption (as most gas is wasted in traffic jams). In practice, big cities with serious traffic problems usually have a large number of taxis traversing on road surfaces. For the sake of management and security, these taxis have already been embedded with a GPS sensor, which enables a taxi to report on its present location to a data center in a certain frequency. Thus, a large number of time-stamped GPS trajectories of taxis have been accumulated and are easy to obtain.

Intuitively, taxi drivers are experienced drivers who can usually find out the fastest route to send passengers to a destination based on their knowledge (we believe most taxi drivers are honest although a few of them might give passengers a roundabout trip). When selecting driving directions, besides the distance of a route, they also consider other factors, such as the time-variant traffic flows on road surfaces, traffic signals and direction changes contained in a route, as well as the probability of accidents. These factors can be learned by experienced drivers but are too subtle and difficult to incorporate into existing routing engines. Therefore, these historical taxi trajectories, which imply the intelligence of experienced drivers, provide us with a valuable resource to learn practically fast driving directions.

In this paper, we propose to mine smart driving directions from a large number of real-world historical GPS trajectories of taxis. As shown in Figure 1, taxi trajectories are aggregated and mined in the *Cloud* to answer queries from ordinary drivers or Internet users. Given a start point and destination, our method can suggest the practically fastest route to a user according to his/her departure time and based on the intelligence mined from the historical taxi trajectories. As the taxi trajectories are constantly updated in the *Cloud*, the suggested routes are state-of-the-art.

When proposing the above-mentioned strategy, two ma-

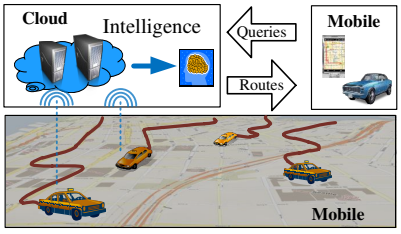


Figure 1: A cloud-based driving directions service

jor concerns come to people's minds. First, some routes on which a taxi can quickly traverse might not be feasible for normal drivers, e.g., the carpool tracks in some highways and a few bus-taxi-preserved tracks in a city. But, in most cases, especially in many urban cities like New York and Beijing, private cars can share the same tracks with taxis. That is, taxis' trajectories can still be referenced by other drivers when finding driving directions in an urban city. Second, the historical trajectory-based approach might not be agile enough to handle some urgent accidents in contrast to real-time traffic analysis. Intrinsically, the traffic flows of a city follow some patterns unless some emergent events happen, such as serious accidents, traffic control and road-works. Given that the probability of these events is much lower than that of regular traffic patterns, our method is still very useful in most situations. At the same time, besides the traffic flow, our method also implicitly incorporates additional factors, such as direction changes and traffic signals. Moreover, this method can find the fastest route in a future time and needs less online communication for data transition. Thus, our solution and the real-time-based approach can complement each other.

However, we need to face the following three challenges when performing our method.

Intelligence Modeling: As a user can select any place as a source or destination, there would be no taxi trajectory exactly passing the query points. That is, we cannot answer user queries by directly mining trajectory patterns from the data. Therefore, how to model taxi drivers' intelligence that can answer a variety of queries is a challenge.

Data Sparseness and Coverage: We cannot guarantee there are sufficient taxis traversing on each road segment even if we have a large number of taxis. That is, we cannot accurately estimate the speed pattern of each road segment.

Low-sampling-rate Problem: To save energy and communication loads, taxis usually report on their locations in a very low frequency, like 2-5 minutes per point. This increases the uncertainty of the routes traversed by a taxi[11]. As shown in Figure 2, there could exist four possible routes (R_1-R_4) traversing the sampling points a and b.

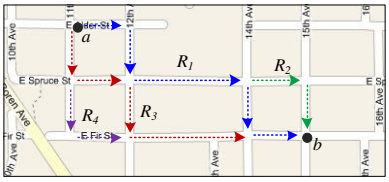


Figure 2: Low-sampling-rate problem

In our approach, we model a large number of historical taxi trajectories with a time-dependent landmark graph, in which a node (landmark) represents a road segment *frequently traversed* by taxis. Based on this landmark graph, we perform a two-stage routing algorithm that first searches the landmark graph for a rough route (represented by a sequence of landmarks) and then finds a refined route sequentially connecting these landmarks. The contributions of this paper lie in the following aspects:

- We propose the notion of a landmark graph, which well models the intelligence of taxi drivers based on the taxi trajectories and reduces the online computation of route-finding.
- We devise Variance-Entropy-Based Clustering (called VE-Clustering) to learn the time-variant distributions of the travel times between any two landmarks.
- We build our system by using a real-world trajectory dataset generated by 33,000+ taxis in a period of 3 months, and evaluate the system by conducting both synthetic experiments and in-the-field evaluations (performed by real drivers). The results show that our method can find out faster routes with less online computation than competing methods.

In the remainder of this paper, we first formally define our problem in Section 2 and give an overview of our approach in Section 3. Then we elaborate on the time-dependent landmark graph construction in Section 4 and route computing in Section 5. After that, we report on the evaluation in Section 6. Finally, we discuss related work in Section 7 and conclude this paper in Section 8.

2. PROBLEM DEFINITION

In this section, we first introduce some terms used in this paper, then define our problem.

Definition 1. (*Road Segment*): A *road segment* r is a directed (one-way or bidirectional) edge that is associated with a direction symbol ($r.dir$), two terminal points ($r.s$, $r.e$), and a list of intermediate points describing the segment using a polyline. If $r.dir$=one-way, r can only be traveled from $r.s$ to $r.e$, otherwise, people can start from both terminal points, i.e., $r.s \rightarrow r.e$ or $r.e \rightarrow r.s$. Each road segment has a length $r.length$ and a speed constraint $r.speed$, which is the maximum speed allowed on this road segment.

Definition 2. (*Dynamic Road Network*): A *dynamic road network* G_r is a directed graph, $G_r = (V_r, E_r)$, where V_r is a set of nodes representing the terminal points of road segments, and E_r is a set of edges denoting road segments. The time needed for traversing an edge is dynamic at least in the following two aspects: (1) *Time-dependent*. Typically, the traffic flow on a road surface varies over days of the week and time of day, e.g., a road could become crowded in rush hours while be quite smooth at other times. (2) *Location-variant*. Different roads have different time-variant traffic patterns. For instance, some streets could still be very fast even in the morning rush. However, the rush hours of a few roads may last for a whole day.

Definition 3. (*Route*): A *route* R is a set of connected road segments, i.e., $R: r_1 \rightarrow r_2 \rightarrow \cdots \rightarrow r_n$, where $r_{k+1}.s = r_k.e$, $(1 \leq k < n)$. The start point and end point of a route can be represented as $R.s = r_1.s$ and $R.e = r_n.e$.

Definition 4. (*Taxi Trajectory*): A *taxi trajectory* Tr is a sequence of GPS points pertaining to one trip. Each point p consists of a longitude, latitude and a time stamp $p.t$, i.e., $Tr: p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_n$, where $0 < p_{i+1}.t - p_i.t < \triangle T$ $(1 \leq i < n)$. $\triangle T$ defines the maximum sampling interval between two consecutive GPS points.

Problem Definition: Given a user query with a start point q_s, a destination q_d and a departure time t_d, find the fastest route R in a dynamic road network $G_r = (V_r, E_r)$ which is learned from a trajectory archive A.

3. OVERVIEW

As shown in Figure 3, the architecture of our system consists of three major components: Trajectory Preprocessing, Landmark Graph Construction, and Route Computing. The first two components operate offline and the third is running online. The offline parts only need to be performed once unless the trajectory archive is updated.

Trajectory Preprocessing: This component first segments GPS trajectories into effective trips, then matches each trip against the road network. 1) *Trajectory segmentation*: In practice, a GPS log may record a taxi's movement of several days, in which the taxi could send multiple passengers to a variety of destinations. Therefore, we partition a GPS log into some taxi trajectories representing individual trips according to the taximeter's transaction records. There is a tag associated with a taxi's reporting when the taximeter is turn on or off, i.e., a passenger gets in or out of the taxi. 2) *Map matching*: We employ our IVMM algorithm [14], which has a better performance than existing map-matching algorithms when dealing with the low-sampling-rate trajectories, to map each GPS point of a trip to the corresponding road segment where the point was recorded. As a result, a taxi trajectory is converted to a sequence of road segments.

Landmark Graph Construction: We separate the weekday trajectories from the weekend ones, and build a *landmark graph* for weekdays and weekends respectively. When building the graph, we first select the top-k road segments with relatively more projections (i.e., being frequently traversed by taxis) as the *landmarks*. Then, we connect two landmarks with a *landmark edge* if there are a certain number of trajectories passing these two landmarks. Later, we estimate the distribution of travel time of each landmark edge by using the VE-clustering algorithm. Now, a time-dependent landmark graph is ready for online computation. Figure 4 demonstrates the key concept of our work.

Route Computing: Given a query (q_s, q_d, t_d), we carry out a two-stage routing algorithm to find out the fastest route. In the first stage, we perform a *rough routing* that searches the time-dependent landmark graph for the fastest rough route represented by a sequence of landmarks. In the second stage, we conduct a *refined routing* algorithm, which computes a detailed route in the real road network to sequentially connect the landmarks in the rough route.

4. TIME-DEPENDENT LANDMARK GRAPH

This section first describes the construction of the time-dependent landmark graph, and then details the travel time estimation of landmark edges.

4.1 Building the Landmark Graph

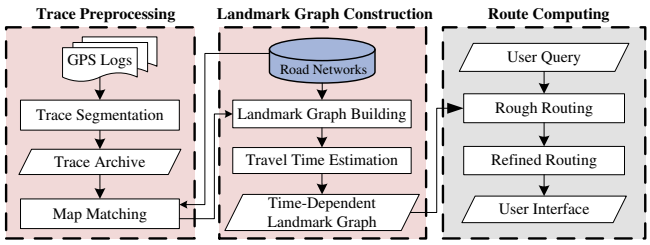


Figure 3: System overview

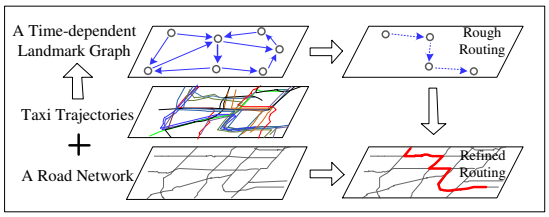


Figure 4: Hierarchical architecture

Definition 5. (*Landmark*): A *landmark* is one of the top-k road segments that are frequently traversed by taxi drivers according to the trajectory archive.

The reason why we use "landmark" to model the taxi drivers' intelligence is that: 1) The notion of landmarks follows the natural thinking pattern of people, and can give users a more understandable and memorable presentation of driving directions beyond detailed descriptions. For instance, the typical pattern that people introduce a route to a driver is like this "take I-405 South at NE 4th Street, then change to I-90 at exit 11, and finally exit at Qwest Field". Instead of giving turn-by-turn directions, which a driver cannot remember, people prefer to use a few landmarks (like NE 4th Street) that highlight key directions to the destination. 2) The sparseness and low-sampling-rate of the taxi trajectories do not support the speed estimation for each road segment while we can estimate the traveling time between two landmarks. Meanwhile, the low-sampling-rate trajectories cannot offer sufficient information for inferring the exact route traversed by a taxi (refer to Figure 2). Thus, we can only use a road segment instead of their terminal points as a landmark.

Here, we detect the top-k road segments as the landmarks instead of setting up a fixed threshold, since a threshold will vary in the scale of taxi trajectories. Later, we connect two landmarks according to definitions 6, 7 and 8.

Definition 6. (*Transition*): Given a trajectory archive A, a time threshold t_{max}, two landmarks u, v, arriving time t_a, leaving time t_l, we say $s = (u, v; t_a, t_l)$ is a *transition* if the following conditions are satisfied:
(I) There exists a trajectory $T_r = (p_1, p_2, \ldots, p_n) \in A$, after map matching, T_r is mapped to a road segment sequence $(r_1, r_2 \ldots, r_n)$. $\exists\ i, j, 1 \leq i < j \leq n$ s.t. $u = r_i, v = r_j$.
(II) $r_{i+1}, r_{i+2}, \ldots, r_{j-1}$ are not landmarks.
(III) $t_a = p_i.t, t_l = p_j.t$ and the *travel time* of this transition is $t_l - t_a \leq t_{max}$.

Definition 7. (*Candidate Edge* and *Frequency*): Given two landmarks u, v and the trajectory archive A, let S_{uv} be the set of the transitions connecting (u, v). If $S_{uv} \neq \varnothing$, we say $e = (u, v; \mathcal{T}_{uv})$ is a *candidate edge*, where

$$\mathcal{T}_{uv} = \{(t_a, t_l) | (u, v; t_a, t_l) \in S_{uv}\}$$

records all the historical arriving and leaving times. The *support* of e, denoted as $e.supp$, is the number of transitions connecting (u, v), i.e., $|S_{uv}|$. The *frequency* of e is $e.supp/\tau$, denoted as $e.freq$, where τ represents the total duration of trajectories in archive A.

Definition 8. (*Landmark Edge*): Given a candidate edge e and a minimum frequency threshold δ, we say e is a *landmark edge* if $e.freq \geq \delta$.

Definition 9. (*Landmark Graph*): A *landmark graph* $G_l = (V_l, E_l)$ is a directed graph that consists of a set of landmarks V_l (conditioned by k) and a set of landmark edges E conditioned by δ and t_{max}.

The threshold δ is used to eliminate the edges seldom traversed by taxis, as the fewer taxis that pass two landmarks, the lower accuracy of the estimated travel time (between the two landmarks) could be. Additionally, we set the t_{max} value to remove the landmark edges having a very long travel time. Due to the low-sampling-rate problem, sometimes, a taxi may consecutively traverse three landmarks while no point is recorded when passing the middle (second) one. This will result in that the travel time between the first and third landmark is very long. Such kinds of edges would not only increase the space complexity of a landmark graph but also bring inaccuracy to the travel time estimation (as a farther distance between landmarks leads to a higher uncertainty of the traversed routes).

We observe (from the taxi trajectories) that different weekdays (e.g., Tuesday and Wednesday) almost share similar traffic patterns while the weekdays and weekends have different patterns. Therefore, we build two different landmark graphs for weekdays and weekends respectively. That is, we project all the weekday trajectories (from different weeks and months) into one weekday landmark graph, and put all the weekend trajectories into the weekend landmark graph.

Figure 5 (A)-(C) illustrate an example of building the landmark graph. If we set $k = 4$, the top-4 road segments (r_1, r_3, r_6, r_9) with more projections are detected as landmarks. Note that the consecutive points (like p_3 and p_4) from a single trajectory (Tr_4) can only be counted once for a road segment (r_{10}). This aims to handle the situation that a taxi was stuck in a traffic jam or waiting at a traffic light where multiple points may be recorded on the same road segment (although the taxi driver only traversed the segment once), as shown in Figure 5 (C). After the detection of landmarks, we convert each taxi trajectory from a sequence of road segments to a landmark sequence, and then connect two landmarks with an edge if the transitions between these two landmarks conform to Definition 8 (supposing δ=1 in this example). Figure 6 shows the detailed algorithm for landmark graph building where M is a collection of road segment sequences (derived from original taxi trajectories).

Algorithm 1: LandmarkGraphConstruction

Input: a road network G_r, a collection of road segment sequences M, the number of landmarks k, the thresholds δ and t_{max}
Output: landmark graph G_l

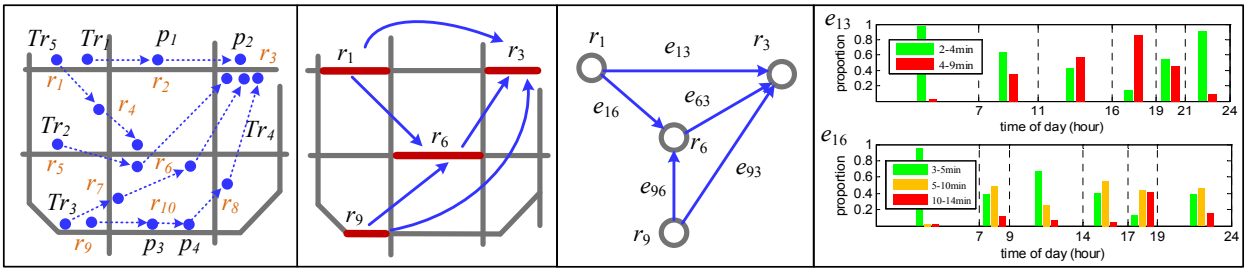
```
1  M ← ∅, E ← ∅, Count[ ]← 0;
2  foreach road segment sequence S ∈ M do
3      foreach road segment r ∈ S do
4          Count[r]++
5  V_l ← Top-k(Count[ ], k);
6  foreach road segment sequence S ∈ M do
7      S ←Convert(S,V_l)/* Converted to landmark sequence */
8      for i ← 1 to |S| − 1 do
9          u ← S[i], v ← S[i + 1] ;
10         if v.t − u.t < t_max then
11             if e_uv = (u, v; T_uv) ∉ E then
12                 E.Insert(e_uv)
13             e_uv.supp++;
14             T_uv.Add((u.t, v.t))
15 foreach edge e = (u, v; T_uv) ∈ E do
16     if e.freq < δ then E.Remove(e) ;
17 return G_l ← (V_l, E);
```

Figure 6: Landmark graph construction algorithm

4.2 Travel Time Estimation

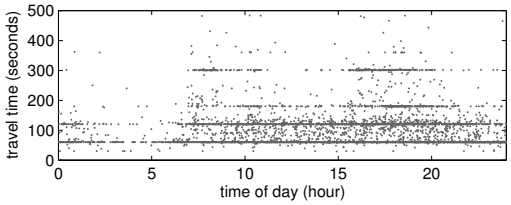
Since the road network is dynamic (refer to Definition 2), we can use neither the same nor a predefined time partition method for all the landmark edges. Meanwhile, as shown in Figure 7(a), the travel times of transitions pertaining to a landmark edge clearly gather around some values (like a set of clusters) rather than a single value or a typical Gaussian distribution, as many people expected. This may be induced by 1) the different number of traffic lights encountered by different drivers, 2) the different routes chosen by different drivers traveling the landmark edge, and 3) drivers' personal behavior, skill and preferences. Therefore, different from existing methods [9, 12] regarding the travel time of an edge as a single-valued function based on time of day, we consider a landmark edge's travel time as a set of distributions corresponding to different time slots. For example, as shown in the bottom part of Figure 5 D), 41 percent of drivers need to spend 10∼14 minutes (refer to the red bar) to traverse e_{16} ($r_1 \rightarrow r_6$) in the time slot 17:00–19:00, while 44 percent of drivers can accomplish this transition with a 5∼10 minute cost (represented by the yellow bar) and the rest of them need less than 5 minutes (denoted as the green bar).

Moreover, the distribution will change over time of day, e.g., the time slots 9:00–14:00 and 14:00–17:00 have different distributions. Additionally, the distributions of different edges, such as e_{13} and e_{16}, change differently over time. To address this issue, we develop the VE-Clustering algorithm, which is a two-phase clustering method, to learn different time partitions for different landmark edges based on the taxi trajectories. In the first phase, called V-clustering, we

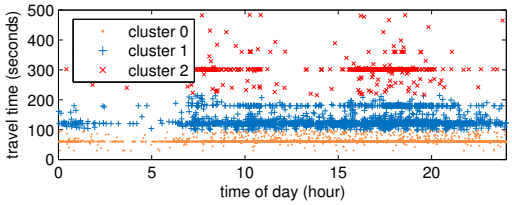


A) Matched taxi trajectories B) Detected landmarks C) A landmark graph D) Travel time estimation

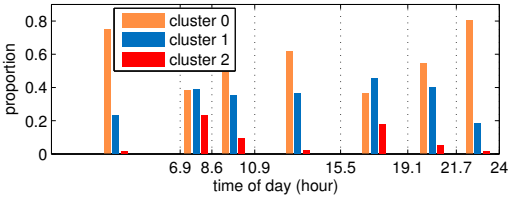
Figure 5: Landmark graph construction



(a) Transitions of a landmark Edge



(b) V-Clustering result



(c) VE-Clustering result

Figure 7: An example of VE-Clustering Algorithm

cluster the travel times of transitions pertaining to a landmark edge into several categories based on the variance of these transitions' travel times. In the second phase, termed E-clustering, we employ the information gain to automatically learn a proper time partition for each landmark edge. Later, we can estimate the distributions of travel times in different time slots of each landmark edge.

Figure 7 demonstrates an example of the VE-Clustering algorithm. Given a landmark edge $e = (u, v; \mathcal{T}_{uv})$, our goal is to estimate the travel time from u to v based on $\mathcal{T}_{uv}$ (T_{uv} is the collection of (t_a, t_l) pairs of e defined in Definition 7). Figure 7(a) plots the travel time of the transitions (on weekdays during 3 months) pertaining to a real landmark edge in a two dimensional space, where the x and y axes denote the arriving time (t_a) and travel time ($t_l - t_a$) respectively. As the number of clusters and the boundary of these clusters vary in different landmark edges, we conduct the following V-Clustering instead of using some k-means-like algorithm or a predefined partition.

V-Clustering: We first sort $\mathcal{T}_{uv}$ according to the values of travel time ($t_l - t_a$), and then partition the sorted list L into several sub-lists in a binary-recursive way. In each iteration, we first compute the variance of all the travel times in L. Later, we find the "best" split point having the minimal weighted average variance (WAV) defined as Equation 1:

$$\mathrm{WAV}(i; L) = \frac{|L_1(i)|}{|L|}\mathrm{Var}(L_1(i)) + \frac{|L_2(i)|}{|L|}\mathrm{Var}(L_2(i)) \quad (1)$$

where $L_1(i)$ and $L_2(i)$ are two sub-lists of L split at the i_{th} element and Var represents the variance. This best split point leads to a maximum decrease of

$$\triangle V(i) = \mathrm{Var}(L) - \mathrm{WAV}(i; L). \quad (2)$$

The algorithm terminates when $\max_i\{\triangle V(i)\}$ is less than a threshold. As a result, we can find out a set of split points dividing the whole list L into several clusters $C = \{c_1, c_2, \ldots, c_m\}$, each of which represents a category of travel times.[1] As shown in Figure 7(b), the travel times of the landmark edges have been clustered into three categories plotted in different colors and symbols.

E-Clustering: This step aims to split the x-axis into several time slots such that the travel times have a relatively stable distribution in each slot. After V-Clustering, we can represent each travel time y_i with the category it pertains to ($c(y_i)$), and then sort the pair collection $S^{xc} = \{(x_i, c(y_i))\}_{i=1}^n$ according to x_i (arriving time). The information entropy of the collection S^{xc} is given by:

$$\mathrm{Ent}(S^{xc}) = -\sum_{i=1}^{m} p_i \log(p_i) \quad (3)$$

where p_i is the proportion of a category c_i in the collection. The E-Clustering algorithm runs in a similar way to the V-Clustering to iteratively find out a set of split points. The only difference between them is that, instead of the WAV, we use the weighted average entropy of S^{xc} defined as:

$$\mathrm{WAE}(i; S^{xc}) = \frac{|S_1^{xc}(i)|}{|S^{xc}|}\mathrm{Ent}(S_1^{xc}(i)) + \frac{|S_2^{xc}(i)|}{|S^{xc}|}\mathrm{Ent}(S_2^{xc}(i))$$

in the E-Clustering, where S_1^{xc} and S_2^{xc} are two subsets of S^{xc} when split at the i_{th} pair. The best split point induces a maximum information gain[6] which is given by

$$\triangle E(i) = \mathrm{Ent}(S^{xc}) - \mathrm{WAE}(i; S^{xc}).$$

The recursive partition within a set stops iff the MDLPC criterion is satisfied[6]. As demonstrated in Figure 7(c), we can compute the distribution of the travel times in each time slot after the E-Clustering process.

Figure 8 shows the framework of the VE-Clustering algorithm where $x_i = t_a$, $y_i = t_l - t_a$. Figure 9 details the procedure of V-Clustering. As E-Clustering is similar to V-Clustering, we skip the details of the E-Clustering algorithm.

Algorithm 2: Variance-Entropy-Based Clustering

Input: a set of points $S = \{(x_i, y_i)_{i=1}^n\} \subseteq \mathbf{R} \times \mathbf{R}$
Output: a sequence of distributions $D_1, D_2, \ldots, D_k$

```
1  S^y ← sorted sequence {y_i}_{i=1}^n order by y_i asending;
2  y_split ← ∅;
3  y_split ←V-Clustering(S^y,δ_v,y_split);
4  C = {c_1, c_2, ..., c_m} ←Convert(S^y, y_split);
   /* Convert S^y into clusters according to y_split        */
5  S^xc ← sort {(x_i, c(y_i))}_{i=1}^n} order by x_i asending;
   /* c(y_i) ∈ C is the cluster of y_i                      */
6  x_split ← ∅;
7  x_split ←E-Clustering(S^xc,δ_e,x_split);
   /* Divide x-axis into several slots                      */
8  for i ← 1 to |x_split| do
9      D_i ←ComputeDistribution(S^xc,i,x_split);
       /* Compute the distribution of slot i                */
10 return D = {D_1, D_2, ..., D_k};
```

Figure 8: Variance-Entropy-Based Clustering

[1]We can use some outlier detection algorithms or interval estimation approaches to handle noisy points.

Algorithm 3: V-Clustering

Input: sorted sequence $L = \{y_i\}_{i=1}^n$, threshold δ_v, a set of split points $\&y\_split$; `/* a global variable */`
Output: a set of split points y_split

```
1 V ←Var({y_i}_{i=1}^n);                    /* the initial variance */
2 V' ← min_i{WAV(i; L)};
3 j ← arg min_i{WAV(i; L)};
4 if V − V' < δ_v/|L| then return y_split;
5 else
6     y_split.Add(j);
7     y_split ←V-Clustering(L_1^j, δ_v, y_split);
8     y_split ←V-Clustering(L_2^j, δ_v, y_split);
```

Figure 9: V-Clustering procedure

5. ROUTE COMPUTING

This section introduces the routing algorithm, which consists of two stages: rough routing in the landmark graph and refined routing in the real road network.

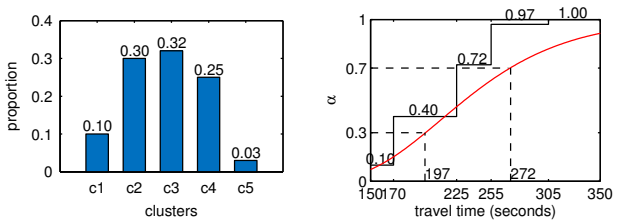
5.1 Rough Routing

Besides the traffic condition of a road, the travel time of a route also depends on drivers. Sometimes, different drivers take different amounts of time to traverse the same route at the same time slot. The reasons lie in a driver's driving habit, skills and familiarity of routes. For example, people familiar with a route can usually pass the route faster than a new-comer. Also, even on the same path, cautious people will likely drive relatively slower than those preferring to drive very fast and aggressively. To catch the above factor caused by individual drivers, we define the *optimism index* as follows:

Definition 10. (*Optimism Index*) The optimism index α indicates how fast a person would like to drive as compared to taxi drivers. The higher rank (position in taxi drivers), the faster the person would like to drive.

For example, $\alpha = 0.9$ means a person usually drives as fast as the top 10% (i.e., 1-0.9) fast-driving taxi drivers. 0.2 means that drivers can only outperform the bottom 20% of taxi drivers. In practise, the α can be learned from a driver's historical trajectories, or set by themselves, or be configured to the mean expected value (or the median) if not given.

Given a user's optimism index α, we can determine his/her time cost for traversing a landmark edge e in each time slot based on the learnt travel time distribution. For example, Figure 10(a) depicts the travel time distribution of an landmark edge in a given time slot ($c_1 \sim c_5$ denotes 5 categories of travel times). Then, we convert this distribution into a cumulative frequency distribution function and fit a continuous cumulative frequency curve [1] shown in Figure 10(b).



(a) Travel time distribution (b) Cumulative frequency

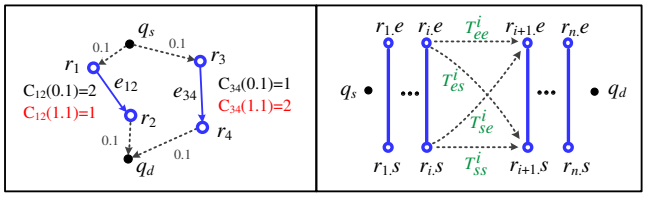
Figure 10: Travel time w.r.t. optimism index

Note this curve represents the distribution of travel time in a given time slot. That is, the travel times of different drivers in the same time slot are different. So, we cannot use a single-valued function. For example, given α=0.7, we can find out the corresponding travel time is 272 seconds, while if we set α=0.3 the travel time becomes 197 seconds.

Now the rough routing problem becomes the typical *time-dependent fastest path* problem. The complexity of solving this problem depends on whether the network satisfies the "FIFO" (first in, first out) property[2]: "In a network $G = (V, E)$, if A leaves node u starting at time t_1 and B leaves node u at time $t_2 \geq t_1$, then B cannot arrive at v before A for any arc (u,v) in E". In practise, many networks, particularly transportation networks, exhibit this behavior [3]. If a driver's route spans more than one time slot, we use the method proposed in[4] to refine the travel time cost to be FIFO.

In the rough routing, we first search m (in our system, we set $m = 3$) nearest landmarks for q_s and q_d respectively (a spatial index is used), and formulate $m \times m$ pair of landmarks. For each pair of landmarks, we find the time-dependent fastest route on the landmark graph by using the Label-Setting algorithm [3]. For any visited landmark edge, we use the optimism index (or expected travel time) to determine the travel time. The time costs for traveling from q_s and q_e to their nearest landmarks are estimated in terms of speed constraint.

For example, in Figure 11 (A), if we start at time $t_d = 0$, the fastest route from q_s to q_d is $q_s \rightarrow r_3 \rightarrow r_4 \rightarrow q_d$. When we arrive at r_3, the time stamp is 0.1, the travel time of e_{34} is 1, then the total time of this route is 0.1+1+0.1=1.2. However, if we start at $t_d = 1$, the route $q_s \rightarrow r_1 \rightarrow r_2 \rightarrow q_d$ now becomes the fastest rough route since when we arrive at r_3, the travel time of the e_{34} becomes 2 and the total time of the previous route is now 2.2.



A) Rough routing B) Refined routing

Figure 11: Route computing

5.2 Refined Routing

This stage finds in the real road network a detailed fastest route that sequentially passes the landmarks of a rough route by dynamic programming. Assume $r_1, r_2, \ldots, r_n$ is the landmark sequence obtained from the rough routing. Recall Definition 1, each landmark r_i has its start point $r_i.s$ and end point $r_i.e$. Let $f_s(i)$ and $f_e(i)$ be the earliest leaving times (after traversing r_i) at nodes $r_i.s$ and $r_i.e$ respectively. Let $T(a, b, c)$ be the travel time of the fastest route from road node a to b without crossing node c. Let $t_{se}(i) = r_i.length/r_i.speed$, i.e., the time (estimated based on speed constraint) for traveling from $r_i.s$ to $r_i.e$, and

$$t_{es}(i) = \begin{cases} t_{se}(i) & \text{if } r_i \text{ is bidirectional} \\ \infty & \text{if } r_i \text{ is one-way.} \end{cases}$$

[2]If the network is non-FIFO, the problem is at least NP-Hard when waiting at a node is not allowed[12].

Using these notations, we have the initial states $f_s(1)$ and $f_e(1)$ as follows:

$$f_s(1) = T(q_s, r_1.e, r_1.s) + t_{es}(1)$$
$$f_e(1) = T(q_s, r_1.s, r_1.e) + t_{se}(1) \quad (4)$$

As shown in Figure 11 (B), let $T^i_{se} = T(r_i.s, r_{i+1}.e, r_{i+1}.s)$ denote the time of the fastest route (using speed constraint in real road network) which starts from point $r_i.s$ and ends at point $r_{i+1}.e$ without crossing $r_{i+1}.s$ in road network G_r. Then T^i_{ee}, T^i_{ss}, T^i_{es} can be similarly defined. Now we have the state transition equations:

$$f_s(i+1) = \min\{f_s(i) + T^i_{se}, f_e(i) + T^i_{ee}\} + t_{es}(i+1)$$
$$f_e(i+1) = \min\{f_s(i) + T^i_{ss}, f_e(i) + T^i_{es}\} + t_{se}(i+1) \quad (5)$$

After $f_s(n)$ and $f_e(n)$ are computed, the total travel time for the optimal route in the real road network is:

$$\min\{f_s(n) + T(r_n.s, q_d, r_n.e), f_e(n) + T(r_n.e, q_d, r_n.s)\}$$

In practise, we can compute T^i_{se}, T^i_{ee}, T^i_{ss}, T^i_{es} and corresponding routes in parallel (for $1 \le i \le n-1$) by utilizing the Dijkstra or A*-like Algorithms with a simple modification (by ignoring node c). Then the final route is a by-product of the dynamic programming since we only need to determine the direction for each landmark road segment.

6. EVALUATION

In this section, we conduct extensive experiments using both synthetic queries and in-the-field evaluations.

6.1 Settings

6.1.1 Data

Road Network: We perform the evaluation based on the road network of Beijing, which has 106,579 road nodes and 141,380 road segments.

Taxi Trajectories: We build our system based on a real trajectory dataset generated by over 33,000 taxis over a period of 3 months. The total distance of the data set is more than 400 million kilometers and the total number of GPS points reaches 790 million. The average sampling interval of the data set is 3.1 minutes per point and the average distance between two consecutive points is about 600 meters. After the preprocessing, we obtain a trajectory archive containing 4.96 million trajectories.

Real-User Trajectories: We use a 2-month driving history of 30 real drivers recorded by GPS trajectories to evaluate travel time estimation. This data is a part of the released GeoLife dataset[17, 16], and the average sampling interval is about 10s. That is, we can easily determine the exact road segments a driver traversed and corresponding travel times.

6.1.2 Evaluation Framework

We evaluate our work according to the following 3 steps.

1) *Evaluating landmark graphs*: We build a set of landmark graphs with different values of k ranging from 500 to 13000. The threshold δ is set to 10, i.e., at least ten times per day traversed by taxis (in total over 900 times in a period of 3 months) and t_{max} is set to 30 minutes. We project each real-user trajectory to our time-dependent landmark graph, and use the landmark graph to estimate the travel time of the trajectory. We study the accuracy of the time estimation changing over k and α. We also investigate the accuracy changing over the scale of the taxi trajectory dataset.

2) *Evaluation based on synthetic queries*: We generate 1200 queries with different geo-distances and departure times. The geo-distances between the start point and destination ranges from 3 to 23km and follows a uniform distribution. The departure time ranges from 6am to 10pm and was generated randomly in different time slots.

We compare our approach with the speed-constraint-based (denoted as SC) method and a real-time-traffic-analysis-based (termed RT) method in the aspects of efficiency and effectiveness. The SC method (offered by Google and Bing Maps) is based on the shortest path algorithm like A* using the speed constraint of each road segment. The RT method first estimates the speed of each segment at a given time according to the road sensor readings and the GPS readings of the taxis traversing on the road segment, and then calculates the fastest route according to the estimated speeds.

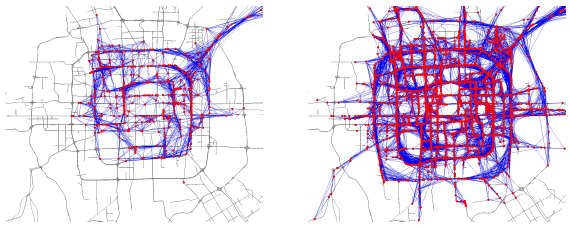
3) *In-the-field evaluation*: We conduct two types of in the field studies: 1) The *same* driver traverses the routes suggested by our method and baselines at different times. 2) Two drivers (with similar driving skills and habits) travel different routes (recommended by different methods) *simultaneously*. As shown in Table 1, we conducted both types of evaluations multiple times and recorded each traverse with a GPS logger. Later, we perform a statistical comparison in the aspects of travel time and distance.

Table 1: Trajectories of the In-the-field Study

	Evaluation 1	Evaluation 2
Num. Trajectories	360	60
Num. Users	30	2
Total Distance (km)	5304	814
Total Duration (hour)	165.24	25.09
Evaluation Days	10	6

6.2 Evaluating Landmark Graphs

Figure 12 visualizes two landmark graphs when $k = 500$ and $k = 4000$. The red points represent landmarks and blue lines denote landmark edges. Generally, the graph ($k = 4000$) well covers Beijing city, and its distribution follows our commonsense knowledge.



(a) k=500 (b) k=4000

Figure 12: Visualized landmark graphs

In Table 3, the second column (SR(N)) denotes the storage ratio between the number of landmarks and that of original road nodes. The third column shows the number of landmark edges. The last column (SR(E)) is the storage ratio between the number of landmark edges and that of road segments. Clearly, our landmark graph is only a small subset of

Table 3: Storage Cost of Landmark Graphs

k	SR(N)	Num. Edges	SR(E)
2,000	0.019	5,518	0.039
4,000	0.038	10,999	0.078
6,000	0.056	15,850	0.112
8,000	0.075	21,219	0.150
1,0000	0.094	25,901	0.183
1,2000	0.113	30,901	0.219

the original road network and the storage cost is lightweight.

By mapping the real-user trajectories to the road segments, we calculate the travel times of these trajectories based on our landmark graph. Then, we compare our estimated time with the real travel time (logged by GPS) using the criteria *error ratio* (ER) defined by:

$$\text{ER} = \frac{\text{estimated time} - \text{real travel time}}{\text{real travel time}}.$$

For example, as shown in Figure 13, a route is comprised of 7 road segments, where r_1, r_3, r_5 and r_7 are landmarks and $r_1 \rightarrow r_3$ and $r_5 \rightarrow r_7$ are two landmark edges. Let t_1 represent the travel time of landmark edge $r_1 \rightarrow r_3$ given a departure time t_0 and an optimism index α. If a road segment is not covered by a landmark edge, like r_4, we use its length divided by the speed constraint to estimate the time cost. So, the estimated time cost of this route is $t_1 + t_2 + t_3$, and ER=$((t_1 + t_2 + t_3) - T)/T$ where T is the real travel time of this route.

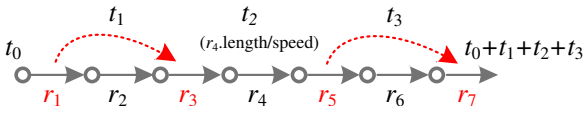


Figure 13: Time estimation for users' routes

Figure 14 shows the ER changing over k and α based on the real-user trajectories. Clearly, when $\alpha = 0.6$, the error ratio achieves the best performance (especially when $k = 10000$, ER$\leq$ 1%). The results validate that the landmark graph can well model the dynamic road network and precisely estimate the travel time of a route.

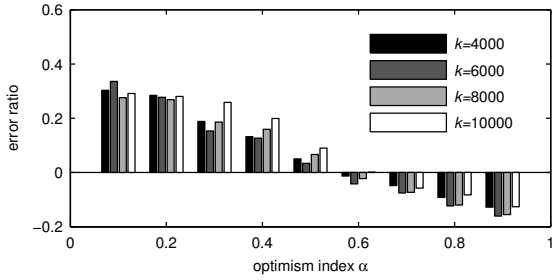
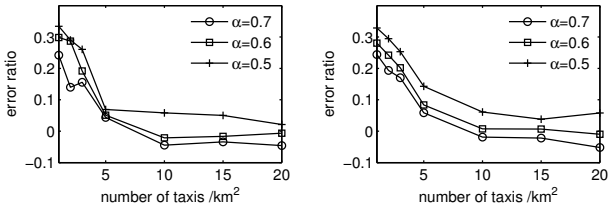


Figure 14: Error ratio w.r.t. optimism index

Figure 15 illustrates the ER changing over the the scale of taxi trajectories (measured by the number of taxis per km^2). The results show that we can get an acceptable performance as long as there are over 5 taxis in a region of 1km^2.

6.3 Evaluation Based on Synthetic Queries

We use two criteria (Fast Rate 1 and Fast Rate 2) to compare the effectiveness between method A and method B (B is the baseline):



(a) k=4000 (b) k=10000

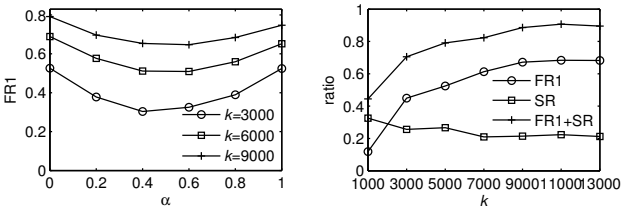
Figure 15: Error ratio over scale of taxi trajectories

$$\text{FR1} = \frac{\text{Number(A's travel time<B's travel time)}}{\text{Number(queries)}}$$

$$\text{FR2} = \frac{\text{B's travel time} - \text{A's travel time}}{\text{B's travel time}}.$$

FR1 represents how many routes suggested by method A are faster than that of baseline method B, and FR2 reflects to what extent the routes suggested by A are faster than the baseline's. Meanwhile, we use SR to represent the ratio of method A's routes being equivalent to the baseline's.

Figure 16 and 17 and Table 4 show the overall performance (FR1, FR2 and SR) of our method. When calculating the FR1, FR2, and SR, both our method and the RT approach use the SC method as a baseline.



(a) w.r.t. α (b) w.r.t. k

Figure 16: Overall performance

Table 4: FR1, SR of TDrive and RT methods

	α	k	FR1	SR
TDrive	0.4	6,000	0.509	0.281
	0.4	9,000	0.647	0.222
	0.6	6,000	0.511	0.272
	0.6	9,000	0.653	0.216
	0.7	6,000	0.544	0.227
	0.7	9,000	0.672	0.214
RT approach			0.206	0.671

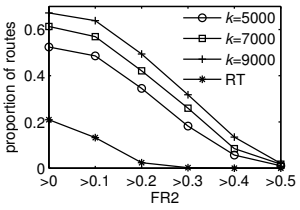
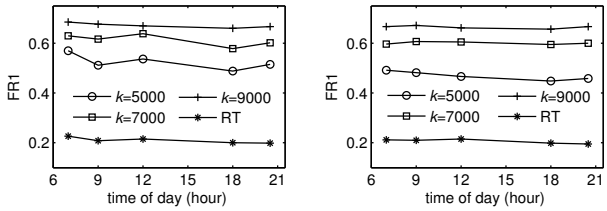


Figure 17: FR2 over k

Figure 16 studies the overall FR1 of our method changing over k and α. When $k = 9000$, the lowest FR1 is still over 60%, i.e., 60% of the routes suggested by our method are faster than that of the SC approach. Figure 16(b) further details the FR1 and SR of our method when $\alpha = 0.7$ (due to the page limitation, we only present the results of a few α in the later evaluations). Here, FR1 is being enhanced with the increase of k when $k < 9000$, and becomes stable when $k >$ 9000. That is, it is not necessary to keep on expanding the scale of a landmark graph to achieve a better performance. Also, as shown in Table 4, our method outperforms the RT approach in terms of FR1, and most routes (67%) suggested by the RT approach are the same as that of the SC method. Figure 17 plots the FR2 of ours and RT. For example, when

$k = 9000$, over 50% routes suggested by our method are at least 20% faster than the SC approach.

We further study the FR1 of our approach and RT in different time slots in Figure 18, and investigate their performance influenced by the geo-distance between the start point and destination of a query in Figure 19. As shown in Figure 18, both our method and the RT approach have a stable performance in different time slots on weekdays and weekends. Moreover, our method has a 30% (on average) improvement over the RT approach when $k \geq 5000$. As depicted in Figure 19, the FR1 of both our method and the RT approach grow as the distance increases, and our method is more capable of answering queries with a longer distance, e.g., FR1>0.8 when distance>20KM. As a longer distance between a source and destination means that more road segments with various traffic conditions are involved, our method and RT both show a clear advantage over the SC approach.



(a) Weekdays (b) Weekends

Figure 18: FR1 w.r.t. time of day

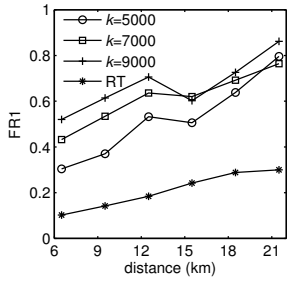


Figure 19: FR1 over geo-distance

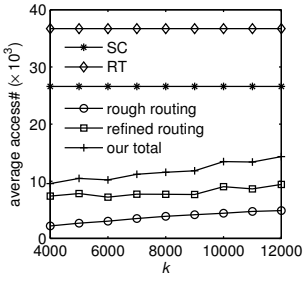


Figure 20: Average access cost per query

The reason why our method outperforms the RT approach is: 1) Coverage: Many road segments have neither embedded road sensors nor taxis traveling on them at a given time. At this moment, the speed constraint of a road segment is used to represent the real time traffic on the road segment. That is also the reason why the RT approach returns many of the same routes as the SC method. 2) Spareness: in a time interval, the estimated travel time is still not very accurate if the number of the taxis is not large enough. 3) Open challenges: As compared to the history-based method, the RT approach is more vulnerable to noise, such as traffic lights, human factors (pedestrians crossing a street), and taxis looking for parking places and passengers.

Though outperforming the baselines, our method still has less than 12% (see Table 4, α=0.7, k=9000) of routes falling behind the SC method. The reason is that: 1) our time estimation method is not perfect, and 2) the three challenges mentioned in the Introduction cannot be fully tackled 100%. However, after studying these fall-behind routes, we find that they are only slightly (on average, FR2=-3%) slower than the SC method.

Figure 20 reports the efficiency of our method by using the average node accesses per query. Obviously, our two-stage routing approach is more efficient than the baselines. The reasons are that: 1) The landmark graph is a small subset of the original road network; 2) the rough route has specified the key directions, hence, reduces the searching area (Figure 21 gives an example); 3) the detailed route between two consecutive landmarks can be computed in parallel.

(a) The SC method (b) T-Drive

Figure 21: An example of searching areas

6.4 In-the-Field Evaluation

Table 5 and Table 6 respectively show the results of the two types of in-the-field evaluations (refer to Section 6.1.2 for details). In these two tables, the symbol $\triangle$ stands for the difference value of distance or duration, R_1 represents the ratio of our routes outperforming the baseline (Google Maps), and R_2 denotes to what extent our routes are beyond that of the baseline. For example, as shown in Table 5, 80.8% of the routes suggested by our T-Drive system are faster than that of Google Maps and on average our routes save 11.9% of time (T-test: $p < 0.005$). In Table 6, we also record the wait time of a route which indicates how long a drive remained stationary due to the traffic lights or traffic jams, e.g., on average, our routes save 26.7% more time than the baseline approach. When doing the in-the-field evaluations, we set α=0.6 and k=9000, where α is learned from these users' driving histories (trajectories, refer to Figure 14).

Table 5: In-the-field Evaluation 1

	T-Drive	Google	$\triangle$	R1	R2
Distance	13.91km	15.56km	1.65km	0.517	0.106
Duration	25.80min	29.28min	3.48min	0.808	0.119

Table 6: In-the-field Evaluation 2

	T-Drive	Google	$\triangle$	R1	R2
Distance	13.58km	13.55km	-0.03km	0.367	-0.002
Duration	23.18min	27.00min	3.82min	0.750	0.141
WaitTime	4.77min	6.50min	1.73min	0.633	0.267

7. RELATED WORK

7.1 Time-Dependent Fastest Route

The time-dependent fastest route problem is first considered in [2]. [5] suggested a straightforward generalization of the Dijkstra algorithm but the authors did not notice it does not work for a non-FIFO network[12]. However, under the FIFO assumption, paper [3] provides a generalization of Dijkstra algorithm that can solve the problem with the same time complexity as the static fastest route problem.

7.2 Traffic-Analysis-Based Approaches

As a very complex problem, urban traffic flow analysis has been studied based on the readings of road sensors and floating-car-data [9, 13]. These works follow the paradigm of "sensor data→traffic flow→driving direction", and are useful in detecting unexpected traffic jams and accidents. The major challenge of such kinds of solutions is the small coverage and sparse density of the sensor data. For example, the traversing speed of a highway with enough road sensors or floating cars can be accurately estimated, while the inferred speed of many service roads, streets and lanes (without enough sensors) are not that precise[8]. Given that users can select any locations as destinations, sometimes the path finding algorithms based on the inferred real-time traffic might not perform as well as we expect.

Different from the above methods, our approach is based on many taxi drivers' intelligence mined from their historical trajectories. This intelligence has implied all the key factors (including traffic flows and signals, etc.) for finding a fast driving route. Actually, GPS-embedded taxis can be regarded as mobile sensors probing real-time traffic on roads, and the accumulated historical GPS trajectories reflect the long-term traffic patterns of a city. As the traffic flows of a city follow some patterns in most cases, our method is very valuable in finding practically fast driving routes for users.

7.3 History-Learning-Based Approaches

Zheng et al.[17, 16, 15] propose several novel approaches to learn the transportation modes from GPS data. Papers [10, 18] present some probabilistic based methods to predict a driver's destination and route based on historical GPS trajectories. Although [18] also uses GPS trajectories generated by 25 taxis, this work aims to predict a driver's destination instead of providing the fastest route that a user can follow. Paper [7] computes the fastest route by taking into account the driving and speed patterns learned from historical GPS trajectories. Our method differs from this work in the following aspects. First, we do not explicitly detect speed and driving patterns from the taxi trajectories. Instead, we use the concept of landmarks to summarize the intelligence of taxi drivers. The notion of landmarks follows people's natural thinking patterns, and can improve efficiency of route finding. Second, our approach is driven by a real dataset while paper [7] is based on the assumption of synthetic data. Actually, real data causes some challenges, e.g., low sampling rate and sparseness of trajectories. Moreover, we consider the time-variant and location-dependent properties of real-world traffic flows.

8. CONCLUSION

This paper presents an approach that finds out the practically fastest route to a destination at a given departure time in terms of taxi drivers' intelligence learned from a large number of historical taxi trajectories. In our method, we first construct a time-dependent landmark graph, and then perform a two-stage routing algorithm based on this graph to find the fastest route. We build a real system with real-world GPS trajectories generated by over 33,000 taxis in a period of 3 months, and evaluate the system with extensive experiments and in-the-field evaluations. The results show that our method significantly outperforms both the speed-constraint-based and the real-time-traffic-based method in the aspects of effectiveness and efficiency. Given over 5 taxis in a region of $1km^2$, more than 60% of our routes are faster than that of the speed-constraint-based approach, and 50% of these routes are at least 20% faster than the latter. On average, our method can save about 16% of time for a trip, i.e., 5 minutes per 30-minutes driving.

We agree that a recommended route would become crowded if many people take it. This is the common problem of path-finding, and this problem is even worse (than ours) in present shortest-path and real-time-traffic-based methods (as our method can be customized for different drivers). In the future, we can reduce this problem by using some strategies, such as load balance (offer top three routes) and data update (in a relatively fast frequency). Another direction in which we are going to move forward is combining real-time traffic information with our approach.

9. REFERENCES

[1] R. Chhikara and L. Folks. *The inverse Gaussian distribution: theory, methodology, and applications*. 1989.

[2] K. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *J. Math. Anal. Appl*, 14(492-498):78.

[3] B. C. Dean. Continuous-time dynamic shortest path algorithms. Master's thesis, Massachusetts Institute of Technology, 1999.

[4] B. Ding, J. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *Proc. EDBT*, pages 205–216. ACM, 2008.

[5] S. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3).

[6] Fayyad and Irani. Multi-interval discretization of continuous-valued attributes for classification learning. *Proc. IJCAI*, pages 1022–1027, 1993.

[7] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. Sondag. Adaptive fastest path computation on a road network: A traffic mining approach. In *Proc. VLDB*, 2007.

[8] A. Gühnemann, R. Schäfer, K. Thiessenhusen, and P. Wagner. Monitoring traffic and emissions by floating car data. *Institute of transport studies Australia*, 2004.

[9] E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding fastest paths on a road network with speed patterns. In *Proc. ICDE*, 2006.

[10] J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. *LNCS*, 4206:243–260.

[11] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate GPS trajectories. In *Proc. GIS*. ACM, 2009.

[12] A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *JACM*, 37(3):625, 1990.

[13] D. Pfoser, S. Brakatsoulas, P. Brosch, M. Umlauft, N. Tryfona, and G. Tsironis. Dynamic travel time provision for road networks. In *Proc. GIS*. ACM, 2008.

[14] J. Yuan, Y. Zheng, C. Zhang, and X. Xie. An interactive-voting based map matching algorithm. In *Proc MDM*, 2010.

[15] Y. Zheng, Y. Chen, Q. Li, X. Xie, and W. Ma. Understanding transportation modes based on GPS data for Web applications. *ACM Transactions on the Web*, 4(1):1–36, 2010.

[16] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Ma. Understanding mobility based on GPS data. In *Proc. Ubicomp*, pages 312–321, 2008.

[17] Y. Zheng, L. Liu, L. Wang, and X. Xie. Learning transportation mode from raw gps data for geographic applications on the web. In *Proc. WWW*, pages 247–256, 2008.

[18] B. Ziebart, A. Maas, A. Dey, and J. Bagnell. Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. In *Proc. Ubicomp*, pages 322–331.