Discovering Popular Routes from Trajectories

Zaiben Chen, Heng Tao Shen, Xiaofang Zhou

School of Information Technology & Electrical Engineering
The University of Queensland, QLD 4072 Australia
{zaiben, shenht, zxf}@itee.uq.edu.au

***Abstract*—The booming industry of location-based services has accumulated a huge collection of users' location trajectories of driving, cycling, hiking, etc. In this work, we investigate the problem of discovering the *Most Popular Route* (MPR) between two locations by observing the traveling behaviors of many previous users. This new query is beneficial to travelers who are asking directions or planning a trip in an unfamiliar city/area, as historical traveling experiences can reveal how people usually choose routes between locations.**

To achieve this goal, we firstly develop a *Coherence Expanding* algorithm to retrieve a transfer network from raw trajectories, for indicating all the possible movements between locations. After that, the Absorbing Markov Chain model is applied to derive a reasonable *transfer probability* for each transfer node in the network, which is subsequently used as the popularity indicator in the search phase. Finally, we propose a *Maximum Probability Product* algorithm to discover the MPR from a transfer network based on the popularity indicators in a breadth-first manner, and we illustrate the results and performance of the algorithm by extensive experiments.

I. Introduction

The ubiquitousness of mobile devices has given rise to a new spectrum of location-based services, which are becoming increasingly popular nowadays. On Google maps, we can easily enjoy the convenience of location-based services such as asking directions, planning driving routes, finding restaurants, etc. In this work, we study the problem of planning a traveling route by considering other people's historical trajectories (traces) that are generated by GPS-enabled devices. Such a collection of trajectories give hints on how people usually travel between locations, and our aim is to discover the most popular route from one given location to another.

This is totally different from existing route planning methods that consider the shortest or fastest path. The most popular route is essentially a statistical result derived from the actual traveling routes conducted by other people in the past, and it is not necessarily to be the shortest path. This route planning service is useful especially for users who are traveling to unfamiliar areas. For example, tourists who travel in a national park are probably to follow a route from the entrance to the exit that covers most of the spots of interest, other than to drive along the shortest path that may miss many attractions. A truck delivery service may tend to use higher quality roads, while the shortest path may contain segments that are not sustainable for heavy vehicles. Thus, the shortest path is not always the most preferable route and we attempt to discover the popular route from historical trajectories. Notice that it does not mean the popular route is always better than the shortest path. What we can show is that in many cases the popular route is quite different from the shortest one. Additionally, for different route planning scenarios, different datasets of trajectories should be considered, e.g., for a tour planning, it is better to adopt the trajectories of previous tourists rather than local people's driving trajectories.

Given the start and destination locations, one can simply check all existing routes connecting the two locations and count the number of trajectories through each of the routes. Then the route with the highest support is supposed to be the most popular one. For instance in Figure 1(a), there are 2 trajectories (traj 2&3) go through route 1 from location A to B, while only one trajectory (traj 1) is on route 2, so we would say route 1 is more preferable. However, this is not always the case, as normally we are not able to find such well-divided groups of trajectories and take each group as a route. As exemplified in Figure 1(b), we got 4 trajectories (traj 1-4) connecting location A and B. All the trajectories intersect and 'twist' with each other, and there could be many possible routes (e.g., A-C-F-B, A-C-F-E-B, etc.). Here, a route can be a combination of different trajectory segments. Therefore, in this case, a specific and reasonable popularity function is necessary to measure how popular a route is. Notice that the term 'popular' is subjective. Different people might have different ideas of defining popularity, and we intend to propose a reasonable one and address the problem of how to discover the optimal route combining trajectory segments. The result should reveal the common traveling behaviors in the dataset that is used. In the case that there is no trajectory connecting A and B directly (see Figure 1(c)), the suggestion of a combined route would be even more helpful to users.

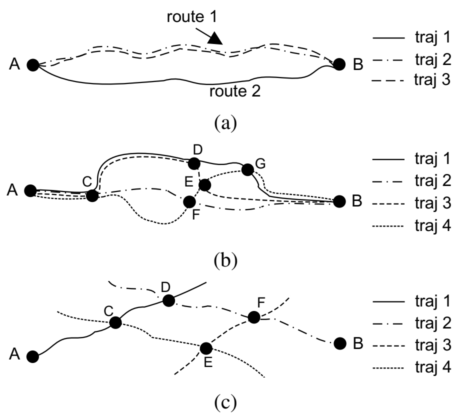


Fig. 1. Discovering Popular Routes - Examples

Route planning/prediction by driving patterns [1], [2], [3], [4] is more or less similar to our work in analyzing users' traveling behaviors, but they mainly focus on mining the sequential patterns of objects' trajectories. The sequential patterns may help in suggesting a drive turn at some intersection in a general case, but they are not sufficient and accurate to discover a popular route to some specified destination. For example, we postulate that A-C-D in Figure 1(c) is a sequence of locations with high support (i.e. many trajectories go through A-C-D), thus we say A-C-D is a driving pattern. However, people who drive following A-C-D might go to any location else rather than our destination B. It is possible that people usually go to B through A-C-E-F, even though the support of A-C-E-F is not that high. Therefore, the pattern A-C-D is not accurate to reflect users' behaviors with respect to the destination B, and we have to define new indicators to summarize the users' behaviors with the presence of a specified destination.

As mentioned above, simply counting the number of trajectories is not enough to discover the popular route between two locations, due to the large number of possible routes and the difficulty in combining trajectory segments. Our basic idea is to construct a transfer network from raw trajectories as an intermediate result to capture the moving behaviors between locations and to facilitate the search of the popular route. Each node in a transfer network is considered as a 'significant location'. We derive the probability of transferring from every 'significant location' to the destination based on the historical trajectories, and the transfer probability is used as an indicator of popularity. Subsequently, the popularity of a route to the destination is defined as the product of transfer probabilities of all 'significant locations' on the route. Thus we focus on using trajectories to create a general view of traffic which can be used in a wide range of applications, rather than focusing on common traffic which are of quite limited application only.

To achieve the goal, we propose to tackle a few problems as stated below. (1) Firstly, we need to retrieve a transfer network from a trajectory database to summarize users' movements. For example in Figure 1(b), the transfer network is comprised of a set of nodes A to G, which are intersections where people branch off, or the end points of trajectories. We define these nodes as *transfer nodes* which are the 'significant locations'. For any two nodes, if there is any contiguous trajectory connecting them without any other nodes in-between, then there is a *transfer edge* between them. So we need to discover all the transfer nodes and transfer edges as a pre-processing procedure. Here a *Coherence Expanding* algorithm that considers directional information is developed for mining the transfer network. (2) The indicators of popularity for transfer nodes and routes need to be established. Here we can no longer use the count number of trajectories as a measurement. The information of the given destination should be considered as well. We use the Absorbing Markov Chain [5] method to deduce the transfer probability for each transfer node. By doing so, we provide a reasonable way to measure the popularity of a route towards the destination, and importantly the transfer probability of each transfer node supplies a criteria for the search of the most popular route. (3) Finally, combining transfer edges to form the optimal route with respect to the popularity function is the target we expect to achieve. Based on the transfer network as well as the transfer probabilities, we propose a *Maximum Probability Product* algorithm for the search of popular routes and briefly prove the accuracy. The algorithm shares the same spirit with the Dijkstra's algorithm [6], and the result route is a path consists of a series of transfer nodes that maximizes the product of transfer probabilities.

As a summary, the essence of the route planning approach in this paper is to 'learn' from history, and suggest a route by mining the most popular path from a trajectory database which is modeled as a transfer network. We mainly make the following contributions:

- We present a new route planning approach that gives another option for users other than existing shortest path based methods.
- We develop an algorithm to establish the transfer network model of a collection of historical trajectories, and utilize the Absorbing Markov Chain model to derive the transfer probability for transfer nodes.
- We propose a reasonable popularity function as well as the search algorithm for discovering the most popular route over a transfer network.
- We demonstrate the results by extensive experiments.

The remainder of the paper is organized as follows. In section II, the related work is discussed. The mining algorithm for establishing a transfer network is introduced in section III, and we derive the transfer probabilities in section IV. The search algorithm for discovering the most popular route is studied in V. Finally we examine the approach in VI and draw a conclusion in section VII. A partial list of the notations used in this paper are summarized in Table I.

II. Related Work

The search of popular routes in light of past movements is highly relevant to trajectory processing/querying issues, including pattern mining [7], [2], [1], [8], trajectory clustering [9], [10], hot route discovery [11], [12], trajectory prediction [3], [4], [13], etc. However, none of them addresses the problem of discovering the most popular route/path from one given location to another. Our work is mainly regarding route planning issues, while the vast majority of existing work is dealing with a general mining/prediction problem.

The discovery of hot routes in [11] and [12] are the most similar ones to our work in identifying routes that are frequently visited by users. In [12], Li et al. propose a density-based algorithm FlowScan to extract hot routes according to the definition of 'traffic density-reachable'. It is essentially a trajectory clustering algorithm based on traffic density, which shares the same idea with [9] and [10] that cluster trajectories by line segment density. An on-line algorithm is also developed by Sacharidis et al. in [11] for searching and maintaining hot motion paths that are traveled by at least a certain number of moving objects. Yet, these two work are tailored for mining paths that are frequently visited from the whole map, while our work is designed to search the frequently visited path for a query with a start location and a destination.

Mining trajectory patterns [7], [2], [1], [8] could potentially help in finding a popular route. Giannotti et al. study in [2] the problem of mining T-pattern, which is a sequence of temporally annotated points, and the target to find out all T-patterns whose support is not less than a support threshold. A T-pattern can naturally be seen as a driving pattern that indicates a popular movement through a sequence of points. Hence, if the given start and end locations are just right on the sequence, we may suggest the sequence to the user as a recommended route. However, not every pair of start and end locations are able to match with an existing pattern, so this approach does not work for the planning of a route between two arbitrary locations. Besides, region of interest (ROI) is used for approximating a trajectory as a sequence of symbols, which is not accurate enough in showing detailed directions for route planning purpose. Similarly in [1] and [8], existing sequential pattern mining algorithms are adopted to explore frequent path segments or sequences of points. In [7], mining periodic movements through regions is investigated as well.

In the pre-processing phase of our solution, a coherence expanding algorithm is developed for retrieving all road intersections from raw trajectories, and subsequently the whole transfer network. The rationale of this algorithm is similar to the density-based clustering algorithm DBSCAN in [14], which expands a cluster from a seed point. However, we use a different connectivity function and different settings for capturing the specific features of road intersections, compared with merely the density of points used in DBSCAN. In [15], Cao et al. also propose an approach to retrieve a road network from trajectories. However, their method is mainly designed for identifying edges while road intersections are not elegantly clarified. The work in [16] is particularly designed for discovering road intersections, but they require an underlying road map available in advance for training a classifier, while in our algorithm, the trajectories may be un-constraint and the road map availability is not assumed.

Other related work includes path planning by considering traffic uncertainty [17], searching similar trajectories [18], [19], [20], [21], [22], shortest path [6], [23], shortest path on time-dependent networks [24], finding the fastest path by speed patterns [25], etc. Nevertheless, all the work above is not able to address the problem of capturing and deriving the popularity of a route between two given locations.

III. Mining Transfer Network

In order to systematically analyze the users' traveling behaviors through GPS trajectories, first of all we establish a *transfer network* from raw trajectories. The transfer network is in effect a directional graph $G(N, E)$ indicating the movements between locations. Here N is a set of *transfer nodes*, which can be an intersection of trajectories or just the end locations of a trajectory. An intersection is physically a small region that trajectories from/to different directions come across, and these locations are supposed to be 'significant' in our model since they are the positions where people can make a turn. E is a collection of *transfer edges* connecting transfer nodes. We say there exists an edge e from node A to B if there is at least one contiguous trajectory from A to B without any other transfer nodes in-between. Besides, for trajectories that move in the same direction between two adjacent nodes, we group them into the same edge. Consequently, we transform trajectories into a routable directional network. As illustrated in Figure 2, we aim to acquire the network (refer to Figure 2(b)) from a set of raw trajectories as plotted in Figure 2(a). In Figure 2(b), a dot represents a transfer node and a line indicates a transfer edge. Notice that if there is a road map available, we can find out the transfer network by map-matching [26] trajectories, but here we attempt to make this work compatible with both constraint and un-constraint trajectories. Typically, traces of hiking, boating, walking, and many out-door activities are not constrained by a road network, and most maps that people think of as free actually have legal or technical restrictions on their use [27], [28], which hold back people from using them in creating new applications.

TABLE I
A LIST OF NOTATIONS

Notation	Explanation
$coh(p, q)$	The coherence between point p and q
δ	The scaling factor
θ	The angle of difference between two moving directions
α, β	The tuning parameters
τ, φ	The coherence and the group size thresholds
$p \ominus q$	p is directly coherence-connected with q
$p \oslash q$	p is coherence-connected with q
$Pr(n_i \rightarrow n_j)$	The turning probability of moving from n_i to n_j
$Pr_d(n_i \rightarrow n_j)$	The turning probability of moving from n_i to n_j w.r.t. a destination d
$p^t_{n_i,n_j}$	The probability of first arrival at n_j, starting from n_i, in exactly t steps
$Pr^t(n_i \rightarrow d)$	The transfer probability of going from n_i to d within t steps
V	The column vector of transfer probabilities for all nodes
R	A route
$\rho(R)$	The popularity of route R

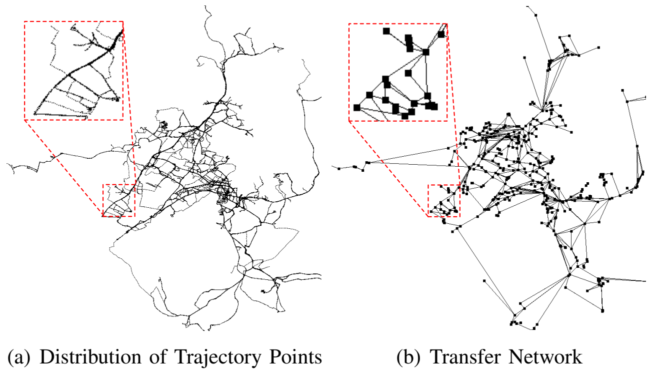

(a) Distribution of Trajectory Points (b) Transfer Network

Fig. 2. Mining Transfer Network (trajectory end points not shown)

The problem arising here is how to detect the intersections of trajectories if there is no map available. Firstly, let's represent a GPS trajectory by a series of points $\{p_1, p_2, \cdots, p_n\}$, where p_i indicates a recorded position $(longitude, latitude)$, and the moving direction of p_i is $\overrightarrow{p_i p_{i+1}}$. We have the following observations upon trajectory intersections:

1) Within an intersection region, the density of trajectory points is normally higher, in comparison with the density of points on an incoming/outgoing road edge, because it is the place where trajectories join together or drivers slow down to make a turn. If we consider an intersection as a group of points, then the size of the group should be greater than some threshold.
2) A number of trajectories change moving direction at an intersection, as some people make turns. The moving directions of trajectory points from/to different road edges are likely to be orthogonal (i.e., angle of difference tends to $\pi/2$). Within a small distance, points whose moving directions differ by 0 or π (i.e., in the same or opposite direction) are probably on the same road, while points with direction difference > 0 and $< \pi$ are possibly moving to different road branches of an intersection. The closer the angle of difference tends to $\pi/2$, the higher possibility that an intersection exists.

Thus the intuition is that we can find intersections by mining groups (clusters) of points that satisfy both density and direction conditions. However, for point density, it can differ greatly at different intersections since there may be tens of thousands of points recorded at a hot intersection while only a few points at an un-popular one. Therefore, we merely set group size threshold as a post-filtering parameter and the mining algorithm mainly distinguishes intersections by the moving direction information. Before describing our algorithm, we firstly list some definitions.

Definition 1: Coherence. Given two trajectory points p and q, the coherence coh between them is defined as:

$$coh(p,q) = \exp(-\left(\frac{dist(p,q)}{\delta}\right)^{\alpha}) \cdot |\sin\theta|^{\beta} \qquad (1)$$

Here $dist(p,q)$ is the Euclidean distance between p and q. δ is a scaling factor. θ is the angle of difference between p and q's moving directions, which ranges from 0 to π. α and β are tuning parameters, and we will discuss setting α, β in the experiment section. In Equation 1, the part $\exp(-(\frac{dist(p,q)}{\delta})^{\alpha})$ scores the coherence by distance, and it decreases exponentially as $dist(p,q)$ goes up. $\sin\theta$, on the other hand, specifies that only points with $\theta \to \pi/2$ can retain a strong coherence. Obviously, points on an transfer edge have a low coherence as they move in a similar direction ($\sin\theta \to 0$), and only points that are close to each other at an intersection and towards different directions will have a strong coherence.

Definition 2: Directly Coherence-Connected. Given a coherence threshold τ, a point p is directly coherence-connected with another point q w.r.t. τ if and only if $coh(p,q) \geq \tau$, and we denote this relation by $p \ominus q$.

It is straightforward that the relation of Directly Coherence-Connected is symmetric for any pair of points, since $coh(p,q) = coh(q,p)$. However, it is not transitive, which means $(p \ominus q \wedge q \ominus r)$ does not imply $p \ominus r$.

Definition 3: Coherence-Connected. A point p is coherence-connected with a point q w.r.t. τ if there is a chain of points $p_1, p_2, \cdots, p_n$, $p_1 = p, p_n = q$, such that $p_i \ominus p_{i+1}$. We denote this relation by $p \oslash q$.

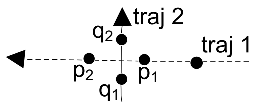


Fig. 3. Coherence-Connected

Considering the example in Figure 3, the coherence between p_1 and p_2 on $traj$ 1 is very low as the θ between their moving directions is about 0, thus they are not directly coherence-connected. However, q_2 on $traj$ 2 (in a very different direction) is directly coherence-connected with both p_1 and p_2, (i.e., $p_1 \ominus q_2$ & $q_2 \ominus p_2$), thus $p_1 \oslash p_2$. Obviously, Coherence-Connected is a symmetric and transitive relation. We have $p \oslash q \to q \oslash p$, and $p \oslash q \wedge q \oslash r \to p \oslash r$. Importantly, by using the coherence-connected relation, we are able to define a cluster as a set of coherence-connected trajectory points. The rationale is similar to that of the DBSCAN clustering [14]. Such a cluster contains a group of points sticked together by coherence, which typically appears only at an intersection (road cross or turning corner) where direction changes can happen. Note that GPS errors may also cause direction changes and that is why we clean the dataset first before clustering.

Definition 4: Cluster. Assume O is the complete set of trajectory points. Given the coherence threshold τ and the cluster size threshold φ, a cluster C w.r.t. τ and φ is a subset of O satisfying the following conditions:

1) If a point $p \in C$ and p is coherence-connected with q w.r.t. τ, then $q \in C$. (Maximality)
2) For any pair of $p, q \in C$ ($p \neq q$), p and q are coherence-connected w.r.t. τ. (Connectivity)
3) The size of $C \geq \varphi$.

This definition looks similar to the density-connected cluster [14], but here we apply a different connectivity function for clustering intersections other than finding groups of dense points. Any two points p, q in a cluster C are coherence-connected, which means there are always a series of points $p_1, p_2, \cdots, p_n$, ($p_1 = p, p_n = q$), such that p_i and p_{i+1} are directly coherence-connected. Therefore, we are able to explore from any p to any q through the Directly Coherence-Connected relation. Intuitively, given a point p in C as a seed of the cluster, we can discover the cluster by expanding from p outwards through exploring surrounding points that are directly coherent-connected with the seed. The new found points are then used as seeds for finding more directly coherent-connected points. This is also the basic idea of our *Coherence Expanding* algorithm.

Lemma 1: Let $p, q \in O$ be any two points that are coherence-connected, $C_1 = \{o | o \in O \wedge o \oslash p\}$, and $C_2 = \{o | o \in O \wedge o \oslash q\}$, then we have $C_1 = C_2$.

Proof: For any point $o \in C_1$, we have $o \oslash p$. Since $\oslash$ is a transitive relation and $p \oslash q$, it is clear that for any $o \in C_1$, we also have $o \oslash q$. Consequently, all points in C_1 are included in C_2 according to the definition of C_2, (i.e. $C_1 \subseteq C_2$). Similarly we can prove that $C_2 \subseteq C_1$. Therefore, we have $C_1 = C_2$. ■

Lemma 1 tells that the expanding results of any two coherence-connected points are exactly the same. For finding a cluster, we can arbitrarily choose any point of the cluster as a seed and expand for the whole set of points of the cluster. This also means that a cluster is uniquely determined by any of it's points.

***Lemma** 2:* Let $p \in O$ be any point of a cluster C. We have $C = \{o | o \in O \wedge o \oslash p\}$.

Lemma 2 is a straightforward conclusion according to Lemma 1 and Definition 4. Based on Lemma 2, we develop the *Coherence Expanding* algorithm for clustering intersections.

Algorithm 1: Coherence Expanding

```
input  : A set of trajectory points $P$; Threshold $\tau, \varphi$;
output: clusters[]
1  for each point $p \in P$ do
2      if p.classified=false then
3          $p$.classified ← true;
4          cluster = expand($p$);
5          if cluster.size $\geq \varphi$ then
6              clusters.add(cluster);
7  return clusters;
```

In Algorithm 1, we simply check each trajectory point in P sequentially. If it has not been classified to any cluster yet, we try to expand it by using the Directly Coherence-Connected relation at line 4. After that if the size of the returned set of points exceeds or is equal to the threshold φ, then the set is stored as a valid cluster. By doing so, eventually all valid clusters will be found, since once we start checking any point of a cluster, all the other points of the cluster will be retrieved in the expanding procedure. For those points not belonging to any valid cluster, we just skip them.

Algorithm 2: expand(p)

```
input  : A point $p$
output: A set of points $result$
1  Queue seeds ← new Queue();
2  seeds.add($p$);
3  $result$.add($p$);
4  while $seeds \neq null$ do
5      $seed$ ← seeds.pop();
6      points ← rangeQuery($seed$, $radius$);
7      for i=0 ; i < points.size ; i++ do
8          $pt$ ← points.get($i$);
9          if pt.classified=false $\wedge$ $coh(seed, pt) \geq \tau$ then
10             seeds.add($pt$);
11             $result$.add($pt$);
12 return $result$;
```

In the expanding procedure as shown in Algorithm 2, we maintain a queue of seeds, which contains only the given point p initially (line 1-2). Then we go to the $while$ loop to check each of the seeds and search for more surrounding points as seeds by a range query centered at $seed$ with a given $radius$ at line 6. Here, the range query is conducted over an R-tree [29] index of all trajectory points. After that, we examine each of the points that fall in range. If a point pt is not classified yet and is directly coherence-connected with the $seed$ from which pt is discovered by the range query (line 9), we add it to the queue as a new seed and append it to the result set (line 10-11). In such a way, we expand the result set from p until no more directly coherence-connected points can be found, and return the set as a final complete cluster (intersection).

Regarding the $radius$ of the range query at line 6 in Algorithm 2, if it is too small, we may miss some directly coherence-connected points. If it is too large, extra effort is needed to examine un-qualified points. Hence we set the radius as the largest distance $dist$ satisfying $coherence \geq \tau$. That is:

$$\exp(-(\frac{dist}{\delta})^{\alpha}) \cdot (\sin\theta)^{\beta} \geq \tau$$

Let $\theta = \pi/2$. By solving the inequation above, the maximal value of $dist$ is found out to be:

$$dist = \delta \cdot \sqrt[\alpha]{-\ln(\tau)}$$

For points with a larger distance than $dist$ from a $seed$, they must have a coherence less than τ, and thus $dist$ is a safe distance to include all possible cluster points.

In practice, as GPS data is more or less dirty, we first reduce outlier points that suddenly jump away by considering physical limits on vehicle speed, before running the clustering algorithm. Besides, linear interpolation is conducted for low sampling-rate trajectories to reduce the possibility that they are missed at some intersections that they do pass through. Direction smoothing is also carried out to alleviate the effect of position fluctuation caused by GPS inaccuracy. This cleaning procedure is mainly based on common sense, and it is just for providing a higher quality dataset.

After discovering all the clusters (intersections), we treat each of them as a transfer node whose location is approximated by the average coordinate, while transfer edges are constructed by checking trajectories between nodes. As exemplified in Figure 2, we group 292,394 trajectory points into a transfer network with 424 nodes (end points of trajectories are not shown here). The benefits of clustering are two-fold. Firstly it summarizes movements by a network which is easier to analyze, and secondly it significantly reduces the number of nodes that need to be considered in the analyzing step. The complexity of the Coherence Expanding algorithm is obviously: *number of points* $\times$ *cost of a range query*.

IV. Deriving Transfer Probability

Through the Coherence Expanding algorithm, we can retrieve a directional transfer network $G(N, E)$ from raw trajectories. In this section, we analyze the users' traveling behaviors on a network, and deduce the transfer probabilities of nodes w.r.t. a given destination. The aim is to find out which transfer node is more likely to lead a user to the destination, and this probability will serve as a popularity indicator.

At a transfer node n_i, a simple way of observing users' historical behaviors is to enumerate all adjacent edges that start from n_i and check how many people ever passed each of them. The *turning probability* of moving from n_i to an outgoing edge $e = (n_i, n_j)$ will then be:

$$Pr(n_i \to n_j) = \frac{\textit{number of trajectories on } (n_i, n_j)}{\textit{number of trajectories on all outgoing edges}}$$

However, this statistics of user behaviors is just for a general circumstance without the consideration of destination. That is, this statistics is purely about how people generally make turns at n_i, and people might just go to any destination. Therefore, when asking about the turning probability at a node w.r.t. a given destination, we should further consider if the historical trajectories that the node contains are (approximately) heading the destination or not to define a more reasonable probability function. We modify the previous equation and define the *turning probability* w.r.t. a destination d as follows:

$$Pr_d(n_i \to n_j) = \frac{\sum_{traj \in (n_i, n_j)} func(traj, d)}{\sum_{traj \in \textit{all outgoing edges}} func(traj, d)} \quad (2)$$

The only difference here is that we use a function $func(traj, d)$ to score how likely a trajectory $traj$ might suggest a correct route to d. We have confidence that a trajectory approximately heading the destination will probably give a correct hint on how to take the next edge to go. We estimate this *likelihood* by:

$$func(traj, d) = \exp\left(-dist_s(traj, d)\right)$$

where $dist_s(traj, d)$ is the shortest Euclidean/network distance between d and the front part of $traj$ that starts from n_i. Apparently, if the front part of $traj$ passes through d exactly, the distance is 0 and thus the likelihood is 1. The larger distance $traj$ deviates from d, the lower likelihood it will be assigned. Consequently, outgoing edges with trajectories close to the destination are associated with higher turning probability, compared with those edges that keep away from d. Therefore, in Equation 2, we provide a simple way to define the probability indicating how users made turns at a transfer node for the purpose of going to a given destination, by considering both the number of trajectories and their distances to the destination, which addresses the problems discussed in Figure 1(a) and 1(c) in the introduction section.

Furthermore, we can consider a travel on such a transfer network based on the turning probability as a Random Walk [30] on a directed graph with the transition probability from node n_i to n_j equals to $Pr_d(n_i \to n_j)$. If we conduct such a random walk on a transfer network following the turning probability, we will probably reach the destination as we always tend to select an edge that is most likely to lead to the destination. However, one question is that:

If we conduct such a random walk, what is the exact probability that, starting from a node n_i, we will eventually reach the destination d within t steps?

We call this probability the *transfer probability* which takes t following transfers into account. In this way, we further consider all possible connecting edges within t steps after leaving n_i, which solves the problem raised in Figure 1(b). Apparently, the larger transfer probability a node n_i holds, the higher confidence we have that n_i will lead us to the destination. Denote by N_t the node that we arrive at after t transfers, and by $p^t_{n_i,n_j}$ the probability that, starting at node n_i, we *first arrive* at node n_j in exactly t steps. We have:

$$p^t_{n_i,n_j} = Pr(N_t = n_j \text{ and, for } 1 \le l < t, N_l \ne n_j | N_0 = n_i) \quad (3)$$

In Equation 3, $p^t_{n_i,n_j}$ is defined as the probability that, starting from $N_0 = n_i$, all the intermediate nodes $N_1, N_2, \cdots, N_{t-1}$ are not n_j, and we arrive at n_j at exactly the t^{th} step. The *transfer probability* $Pr^t(n_i \to d)$ of going from any n_i to destination d within t steps is in fact the sum of probability that we first arrive at d in 1, 2, $\cdots$, t step. Consequently, we have:

$$Pr^t(n_i \to d) = \sum_{j=1}^{t} p^j_{n_i,d} \quad (4)$$

The idea is that the *transfer probability* $Pr^t(n_i \to d)$ can be used as an indicator to reflect how popular a transfer node n_i is, w.r.t. the given destination d. The intuition is that a higher transfer probability implies more historical trajectories (and also more following trajectories) head for the destination. As exemplified in Figure 4 (a sub-graph of Figure 2(b)), we draw transfer nodes by rectangles with the size in proportional to their transfer probabilities. The destination is shown as a circle. Here, we set $t = 20$, and it can be seen that more people travel to the destination through those transfer nodes in the left part (i.e. bigger rectangles). Regarding choosing a proper t, it is discussed later in this section.

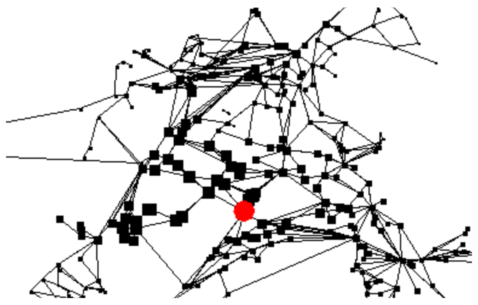


Fig. 4. Distribution of Transfer Probability

In order to model the Random Walk and to compute the transfer probability (i.e. $Pr^t(n_i \to d)$ in Equation 4) for all nodes in a transfer network, we adopt the Absorbing Markov Chain model [5], which is a special type of Markov Chains with at least one *absorbing* state. A state (node) n_i of a Markov chain is called absorbing if it's impossible to leave it, which means the transition probability from n_i to n_i (itself) is always

1, while those non-absorbing states are called *transient* states. In our directional transfer network, the destination node d is treated as an absorbing state, since whenever we arrive, we just stay there and we don't consider a route to d that passes the destination more than once. Additionally, those end points of trajectories without any outgoing edges are also considered as absorbing states since one can not move from them to another node in a directional network. All other transfer nodes are considered as transient states. The *transition matrix* P for m transfer nodes can be represented by:

$$P = \begin{array}{c|cccc} & n_1 & n_2 & \cdots & n_m \\ \hline n_1 & P(1,1) & P(1,2) & \cdots & P(1,m) \\ n_2 & P(2,1) & P(2,2) & \cdots & P(2,m) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ n_m & P(m,1) & P(m,2) & \cdots & P(m,m) \end{array}$$

where the entry $P(i,j)$ denotes the transition probability of moving from node n_i to n_j as defined in Equation 5.

$$P(i,j) = \begin{cases} 1 & \text{if } n_i \text{ is an absorbing state } \& \ i = j \\ Pr_d(n_i \to n_j) & \text{if } n_i \text{ is a transient state } \& \ i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

For absorbing states, they transfer to themselves with probability 1, while transient states make transitions to adjacent nodes according to the turning probability determined by Equation 2. The purpose of adopting the Absorbing Markov Chain model to represent a transfer network is for figuring out the probability of the first arrival to d (i.e., $p^t_{n_i,d}$), and consequently $Pr^t(n_i \to d)$.

Assume there are totally x absorbing states, and y transient states ($x + y = m$). We group absorbing states into ABS and transient states into TR, then the transition matrix P can be re-organized in the following canonical form [5]:

$$P = \begin{array}{c|cc} & \text{TR} & \text{ABS} \\ \hline \text{TR} & \text{Q} & \text{S} \\ \text{ABS} & 0 & \text{I} \end{array} \tag{6}$$

I is a $x - by - x$ identity matrix, 0 is a $x - by - y$ zero matrix, Q is a $y - by - y$ matrix indicating the transition probability between transient states, and S is a $y - by - x$ matrix indicating the transition probability from transient states to absorbing states. To acquire the transition probability from node n_i to n_j in exactly t steps, we take the t^{th} power of P and we get:

$$P^t = \begin{array}{c|cc} & \text{TR} & \text{ABS} \\ \hline \text{TR} & \text{Q}^t & * \\ \text{ABS} & 0 & \text{I} \end{array} \tag{7}$$

where Q^t is the t^{th} power of Q, and $*$ is a $y - by - x$ matrix written in terms of Q and S. From the Markov Chain theory, we know that the $(i,j)^{th}$ entry $P^t(i,j)$ of the matrix P^t is the probability of being at the state n_j after t steps, starting from n_i. Nevertheless, $P^t(i,j)$ is not equal to the $p^t_{n_i,n_j}$ defined in Equation 3, as $p^t_{n_i,n_j}$ has to be the probability of the first arrival while $P^t(i,j)$ does not guarantee this condition. However, we don't need to know $p^t_{n_i,n_j}$ for all n_j, but just $p^t_{n_i,d}$ for the destination d that is an absorbing state. This makes the problem simpler, and we have the following lemma.

***Lemma** 3:* A route that first visits the destination in exactly ($t > 0$) steps (transfers), must start from a transient state, and the state at $t = 1, 2, \cdots, t-1$ step is also a transient state.

Proof: If we start from or visit any absorbing state other than the destination before arriving at the destination node, we are not able to get to the destination as an absorbing state always transfers to itself. Therefore, the lemma is proved. ■

Since the 1^{st} to $(t-1)^{th}$ states of a route are transient, a route must transfer between transient states for $t-1$ times and finally jump from a transient state to the destination at the t^{th} step. For the first $(t-1)$-step transfer, it's probability can be acquired from Q^{t-1}, while the probability of moving from a transient state to the destination is given in S. Consequently, the $p^t_{n_i,d}$ for a given n_i can be computed in the following way:

$$p^t_{n_i,d} = \sum_{n_k \in \text{TR}} \left(P^{t-1}(i,k) \cdot P(k,d)\right) \tag{8}$$

where $P^{t-1}(i,k)$ is the probability of transferring from n_i to another transient state n_k in exactly $t-1$ steps, and $P(k,d)$ is the probability of transferring from n_k to the destination d in one step. Apparently, $P^{t-1}(i,k)$ is an entry of Q^{t-1} in the upper-left block of the matrix P^{t-1} (refer to the canonical form in Equation 7), and $P(k,d)$ is an entry of S in the upper-right block of the matrix P in Equation 6. Therefore, by combining Equation 4 and 8, the transfer probability of a given node n_i w.r.t. d and t is determined by:

$$\begin{aligned} Pr^t(n_i \to d) = & \ \textstyle\sum_{j=1}^t p^j_{n_i,d} \\ = & \ \textstyle\sum_{j=1}^t \sum_{n_k \in \text{TR}} \left(P^{j-1}(i,k) \cdot P(k,d)\right) \end{aligned} \tag{9}$$

Note that when $j = 1$, it goes from a transient state n_k to d directly in one step and we set $P^0(i,k) = 1$. To compute the transfer probability for each transfer node that belongs to transient states, we may conduct the computation by matrix multiplications. Assume $n_1, n_2, \cdots, n_l$ are the transient nodes in TR. We suppose to derive the column vector:

$$V = \left[Pr^t(n_1 \to d), Pr^t(n_2 \to d), \cdots, Pr^t(n_l \to d)\right]^T$$

for a given destination d and parameter t. Now we have Q and S from Equation 6, and d is included in ABS. Let's denote by D the column vector corresponding to node d in the sub-matrix S (i.e., $D = S[*, d]$). The result V is calculated by:

$$V = D + Q \cdot D + Q^2 \cdot D + \cdots + Q^{t-1} \cdot D \tag{10}$$

An example result of V has been shown in Figure 4 where we show the transfer probability of transfer nodes by rectangles in different sizes. Since each node can potentially be the destination, we pre-compute the vector V for each transfer node assuming it as the destination, and record all V for the purpose of searching the most popular route. Totally, it consumes $O(m^2)$ space for storing the pre-computed vectors, if there are m transfer nodes. The computation involves $t-1$ matrix multiplications of Q that causes $O(t \times m^3)$ complexity in CPU time. Algorithm 3 lists the procedures for deriving the transfer probabilities for a transfer network $G(N, E)$.

```
Algorithm 3: Deriving Transfer Probability
input : A transfer network $G(N, E)$
output: A vector $V$ for each node $\in N$
1 for each transfer node $n_i \in N$ do
2     set $n_i$ as the destination;
3     construct the transition matrix $P$ by Equation 5;
4     re-organize $P$ in a canonical form;
5     acquire $Q$, $S$ from $P$;
6     derive $V$ by Equation 10;
7     store $V$;
```

Choosing a proper t is also important in the derivation of transfer probabilities. It specifies the maximum step we take into account, and the length of the longest route that we consider. For a route whose length is excessively large, it does not make any sense as people would not take such a route to travel. On the other hand, if t is small, for example, even smaller than the step number of the shortest route to the destination, then we fail to discover a route for the user because there is no route that can reach the destination within t steps (i.e., transfer probability = 0). Considering the two factors, we set t as the diameter of the transfer network in our experiments, which guarantees at least one route can be found between any two nodes and also avoids considering those excessively long routes.

If a user starts from a trajectory end point that belongs to absorbing states, then no route exists in the directional network. An alternative solution is to extend the transfer network to an un-directional one with minor additional changes.

V. Searching the Most Popular Route

Through mining transfer network and the derivation of transfer probabilities, we acquire a directional transfer network $G(N, E)$ with a set of transfer probability vectors (V) indicating how possible a transfer node would lead one to his/her destination d. We take the transfer probability of a transfer node n_i w.r.t. d as the *popularity indicator*:

$$n_i.popularity(d) = Pr^t(n_i \to d)$$

If $n_i = d$, we assume $n_i.popularity(d) = 1$, and if n_i is a trajectory end point that belongs to absorbing states, we set $n_i.popularity(d) = 0$. Each transfer node n_i maintains m indicators: $n_i.popularity(n_1), \cdots, n_i.popularity(n_m)$, for all m nodes in the transfer network which are potential destinations. An indicator conveys the popularity that people take the transfer node for going to the corresponding destination. In the following we study how to discover the most popular route in light of the node popularity indicators. Firstly, we have some definitions:

Definition 5: **Route.** A route R is defined as a consecutive sequence of transfer nodes $n_1 \to n_2 \to \cdots n_i$, where $(n_j, n_{j+1}), (1 \leq j < i)$, is an existed transfer edge.

Definition 6: **Route Popularity.** The popularity $\rho(R)$ of a route $R = n_1 \to n_2 \to \cdots n_i$ w.r.t. a given destination d, is defined as the product of the popularity indicator of each transfer node w.r.t. d.

$$\rho(R) = \prod_{j=1}^{i} n_j.popularity(d) \tag{11}$$

Notice that when talking about the popularity of a route, a destination must be specified, and $\rho(R)$ is just a relative value reflecting the popularity of R w.r.t. going to d. $\rho(R)$ is not the accurate value of the actual probability that people travel through R. When a route is long, $\rho(R)$ may be very small.

Definition 7: **The Most Popular Route (MPR).** The MPR from a start node s to a destination node d is the route $R = n_1 \to n_2 \to \cdots n_i, (n_1 = s, n_i = d)$, such that the value $\rho(R)$ is maximized among all possible routes from s to d.

Obviously, the number of possible routes between two nodes can be very large in a transfer network, and the enumeration of all combinations of transfer edges that constitute a route can be computationally inefficient. However, we have the following observation which enables us to develop a breadth-first search algorithm that is similar to the Dijkstra's shortest path approach [6].

Lemma 4: If a route $R = n_1 \to n_2 \to \cdots \to n_i$ is the MPR from s to d, $(s = n_1, d = n_i)$, then for any sub-route $SR = n_j \to n_{j+1} \to \cdots n_k$, $(1 \leq j < k \leq i)$, the product $\rho(SR)$ of $n_j, n_{j+1} \cdots, n_k$'s popularity indicators is also maximized among all possible routes from n_j to n_k.

Proof: Suppose on the contrary that the product $\rho(SR)$ of the sub-route $SR = n_j \to n_{j+1} \to \cdots n_k$ is not maximized, then there exists another route SR' from n_j to n_k that produces a larger product of popularity indicators, i.e., $\rho(SR') > \rho(SR)$. Thus, we can construct a new route R^* from n_1 to n_i through SR', $(R^* = n_1 \to \cdots n_{j-1} \to SR' \to n_{k+1} \cdots \to n_i)$, such that $\rho(R^*) > \rho(R)$, which contradicts with the assumption that R is the MPR from n_1 to n_i. ■

Lemma 4 implies that the popularity of any sub-route of the MPR is also maximized. This poses a clue that we can construct the MPR between two nodes by conquering the sub-problems of finding it's sub-routes that also produce the maximum $\rho()$ value. Indicate by $R(n_i)$ the route from $s = n_1$ to another transfer node n_i $(i = 1, 2, \cdots, m)$ that maximizes the $\rho()$ value w.r.t. the destination d. We sort the m routes $R(n_i)$, $(i = 1, 2, \cdots, m)$, in the descending order of $\rho()$ value, as follows:

$$R(n_{i_1}) \succ R(n_{i_2}) \succ R(n_{i_3}) \cdots \succ R(n_{i_m})$$

where $n_{i_1} (i_1 = 1)$ is the start node s, and for any $1 \leq k < l \leq m$ we have $\rho(R(n_{i_k})) \geq \rho(R(n_{i_l}))$. Apparently, if $k < l$, $R(n_{i_l})$ must not be a sub-route of $R(n_{i_k})$ because a route's popularity must not be larger than it's sub-route's popularity. Therefore, for discovering any route $R(n_{i_l})$, we can firstly conquer all $R(n_{i_k})$, $(k < l)$, as shown in the following:

$$\begin{aligned}&\rho(R(n_{i_l})) = \\ &\max_{k<l \wedge (n_{i_k}, n_{i_l}) \text{ exists}}\{\rho(R(n_{i_k}))\} \times n_{i_l}.popularity(d)\end{aligned} \tag{12}$$

Here d is the destination. Equation 12 means that a route $R(n_{i_l})$ is comprised of the sub-route $R(n_{i_k})$, ($k < l$ and (n_{i_k}, n_{i_l}) is an existed transfer edge), that maximizes the popularity $\rho()$ value, plus the node n_{i_l} itself. Consequently, the idea is that we can search from the start node s and expand outwards in the descending order of the $\rho()$ value. Once all $R(n_{i_k})$ ($k < l$) are discovered, $R(n_{i_l})$ can be extended from one of them. This is similar to the Dijkstra's shortest path algorithm that constructs a shortest path tree from the start node by expanding in a breadth-first way. Based on Equation 12, we propose the *Maximum Probability Product* Algorithm for the discovery of MPR as demonstrated in Algorithm 4.

Algorithm 4: Maximum Probability Product

```
input : A transfer network $G(N, E)$,
        $N = \{n_1, n_2, \cdots, n_m\}$;
        Start node $s$; Destination node $d$
output: The most popular route MPR
1  For all $n_i \in N$, label $L(n_i) \leftarrow 0$;
2  $L(s) \leftarrow 1$;
3  Priority Queue $PQ \leftarrow null$;
4  Scanned Nodes $SN \leftarrow null$;
5  $PQ$.enqueue($s$);
6  while $PQ \neq null$ do
7      $u \leftarrow PQ$.extractMax();
8      if $u = d$ then
9          return MPR;
10     $SN$.add($u$);
11     for each $v \in u.adjacentNodes$ do
12         if $L(v) < L(u) \times v.popularity(d)$ then
13             $L(v) \leftarrow L(u) \times v.popularity(d)$;
14             $v$.predecessor $\leftarrow u$;
15             $PQ$.add($v$);
```

In the Maximum Probability Product Algorithm, we record the maximum $\rho()$ value of the route from the start node s to node n_i by a label $L(n_i)$ which is initialized to be 0 and only $L(s)$ is set to be 1 (line 1-2). A max priority queue PQ is utilized to determine the node with the maximum $\rho()$ label value from un-scanned nodes. At the beginning, all nodes are un-scanned, so SN is null, and PQ just contains the start node s. Then in the while loop (from line 6), we extract the node u with the maximum label from PQ, and update the labels of it's adjacent nodes ((u, v) is an existed transfer edge) in line (11-15). If $L(v) < L(u) \times v.popularity(d)$, which means that we find a more popular route to v through node u, then we update v's label and take u as v's predecessor in the route. Besides, all discovered nodes are added to the priority queue for further examination. Once the destination d is pop out from the queue (line 8), the most popular route from s to d is discovered, and we can retrieve the whole route by following the predecessor link of each node from d.

The complexity of Algorithm 4 is the same as that of the Dijkstra's algorithm, which is $O(|E|+|N|\log|N|)$, where $|E|$ is the number of edges and $|N|$ is the number of nodes. The proof of the correctness of the Algorithm is by an inductive-hypothesis method that tries to prove that for any node n_i, the final $L(n_i)$ label value is maximized among all possible cases, which is similar to the proof of the Dijkstra's algorithm [31].

In practice, if a user starts from a position on an transfer edge other than from a transfer node exactly, we may find out the MPRs from both end nodes of the edge to the destination and take the one with larger $\rho()$ value as the result. Moreover, a future work may also take the length of an edge into account to design another popularity function for the search, and Algorithm 4 can still be used without any change. Notice that, in a directional transfer network, a route to the destination might not exist in some cases. It is straightforward to solve the problem by extending to an un-directional network.

One may ask why we do not simply use the turning probability $Pr_d(n_i \rightarrow n_j)$ in Equation 2 as the popularity indicator for each transfer edge, and then the popularity of a route can be defined as the product of the turning probabilities of all edges on it. By doing so, we achieve a similar definition of route popularity as the one in Equation 11, and the Maximum Probability Product algorithm can still be used in a similar way. However, a problem with this alternative option is that the search algorithm just considers the local information of the current node other than t steps further, which possibly causes an incorrect result. We will demonstrate this alternative option (denoted by 'MPR-alternative') as well in the experiments.

VI. Experiments

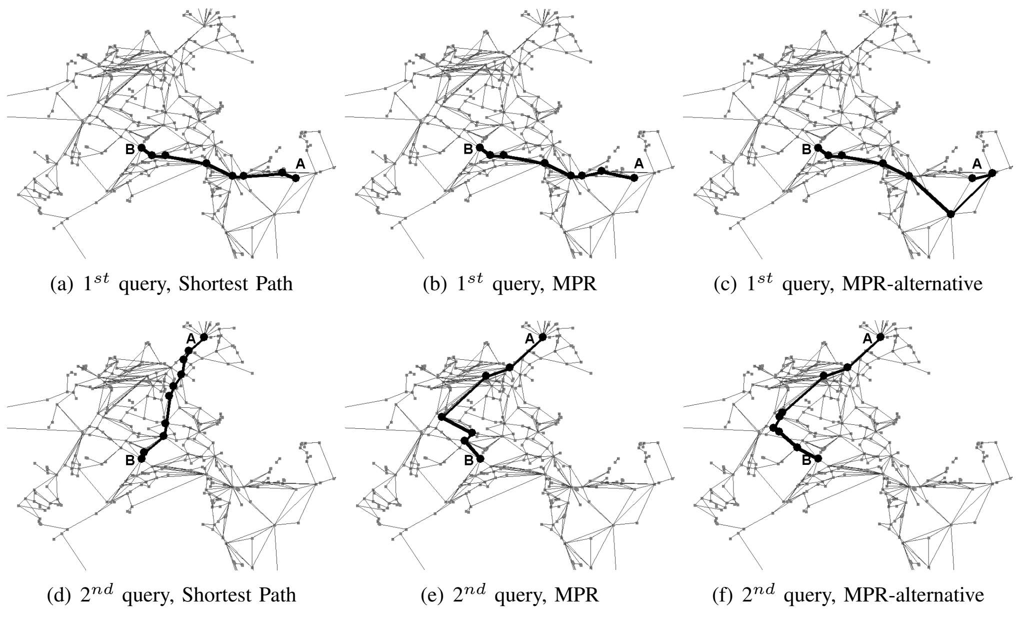
In this section, we conduct experiments on a real trajectory dataset[1] that consists of 276 truck trajectories in the Athens city. After interpolation, the dataset contains totally 292,394 trajectory points and the distance between any two consecutive points is guaranteed to be no more than 100 meters. The previous Figure 2(a) illustrates the distribution of the dataset by plotting all trajectory points, which has already illustrated the city's road network. In our work, the size of a dataset is much less critical than in many other performance-oriented experiments, as long as the truck dataset can reveal enough clues about how the truck traffic flows in the city. Certainly, if a larger dataset is available, then more precise results can be delivered, as a more complete description of the users' movements benefits our algorithms.

The *Coherence Expanding* algorithm and the *Maximum Probability Product* algorithm are implemented in Java and examined on a windows platform with Intel Core 2 CPU (2.13GHz) and 1.0GB Memory. The mining of transfer network and the derivation of transfer probabilities are executed off-line, so they are one-off pre-computation processes. The transfer network is maintained by adjacency lists, and the search of MPR is carried out in real time.

A. Mining Transfer Network

Firstly, when mining a transfer network from trajectories, the Coherence Expanding algorithm is sensitive to the coherence generated between points, which is susceptible to the

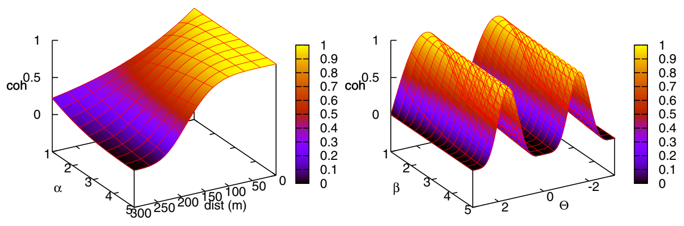
[1]http://www.rtreeportal.org/


(a) 1^{st} query, Shortest Path (b) 1^{st} query, MPR (c) 1^{st} query, MPR-alternative

(d) 2^{nd} query, Shortest Path (e) 2^{nd} query, MPR (f) 2^{nd} query, MPR-alternative

Fig. 6. Illustration of Example Queries (Start Node: A, End Node: B)

tuning parameters α and β. In the experiments, the scaling factor δ is set to be 200 meters, and we show in Figure 5(a) and 5(b) how the coherence is affected by α and β respectively. Let's fix $\theta = \pi/2$ in Equation 1, so the coherence $coh(p, q) = \exp(-\left(\frac{dist(p,q)}{\delta}\right)^{\alpha})$. From Figure 5(a) we can see that, with a relatively larger α, the coherence drops more sharply when the distance $dist(p, q)$ exceeds some value, e.g., the coherence



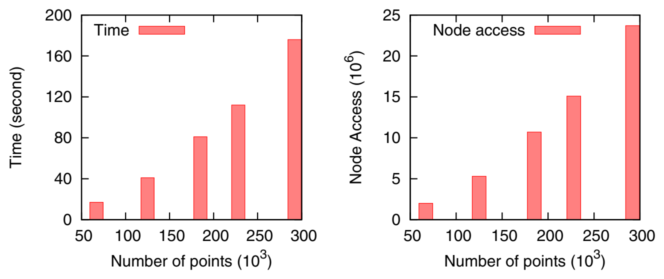
(a) Tuning α (b) Tuning β

Fig. 5. Tuning Parameters

starts to drop drastically when $dist(p, q) \approx 100m$ and approximates 0 when the distance is larger than 200m. Therefore, only points whose mutual distance is less than 100m are likely to be considered within the same cluster (intersection). Similarly, if we fix $dist(p, q) = 0$, then $coh(p, q) = |\sin\theta|^{\beta}$, and we can see from Figure 5(b) that a relatively larger β also makes the curve of coherence steeper. Only points whose moving directions differ by approximately $\pi/2$ can generate a strong coherence. Considering that roads are not always orthogonal at an intersection, we set $\beta = 2$ and then a comparatively strong coherence can still be generated even $\theta \approx \pi/4$.

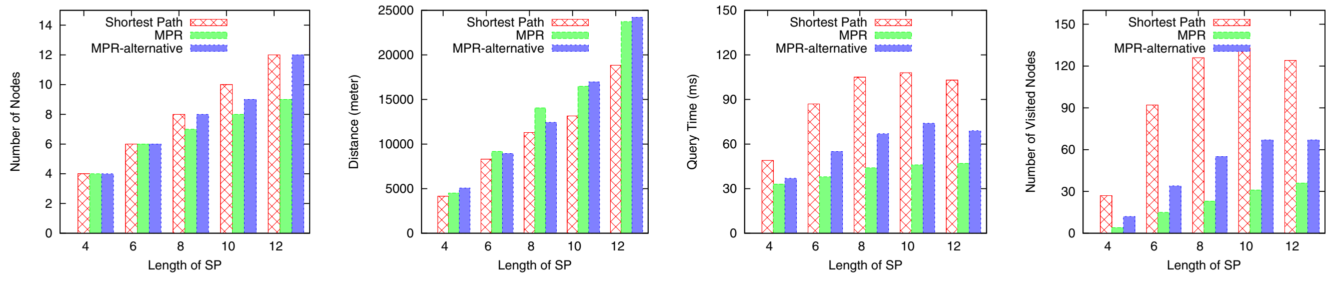
Having $\delta = 200$ meters, $\alpha = 5$ and $\beta = 2$, we also configure the coherence threshold $\tau = 0.5$ and the cluster size threshold $\varphi = 3$. Note that these parameters are dependent on the dataset we use, and a careful tuning process is required. By comparing with a generated road map downloaded from OpenStreetMap [27], 401 out of 424 transfer nodes are correctly clustered by our algorithm, which produces a *false rate* = 0.054. From Figure 2(b), it is easily seen that the retrieved transfer network preserves the shapes and movements of the original trajectories quite well. In the following, we demonstrate the cost of the Coherence Expanding algorithm.

We divide the dataset into groups with different numbers of trajectory points, and as shown in Figure 7, both the clustering time and R-tree node access increase linearly as the number of trajectory points goes up from 5×10^4 to about 3×10^5. This pre-computation process consumes about 180 seconds and involves around 2.3×10^7 node access for the complete dataset. After clustering, we take all the intersections and additionally the end points of trajectories as transfer nodes, and then construct transfer edges by checking the trajectories that go through each transfer node.



(a) Execution time (b) R-tree node access

Fig. 7. Performance of the Coherence Expanding Algorithm



(a) (b) (c) (d)

Fig. 8. The Shortest Path vs. The MPR

B. Deriving Transfer Probability

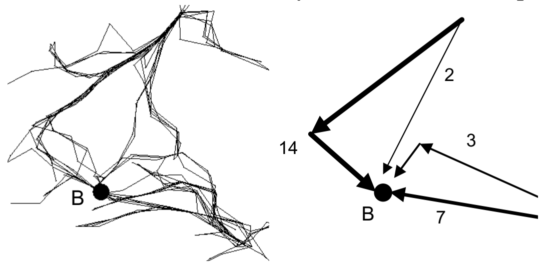
For figuring out the transfer probability and derive the vector V (see Equation 10) for each of the transfer nodes, we first of all calculate the transition probability w.r.t. each node by Equation 5. Here we simply enumerate every trajectory that goes through a transfer node and compute the probability by Equation 2. Then we acquire the transition matrix P for each transfer node, and the calculation of vector V in Equation 10 is conducted using Matlab off-line. After that, we attach the transfer probabilities in V as indicators to the corresponding transfer nodes. The details of this part are skipped as the matrix operations involved here are straightforward.

C. Illustration of the MPR

In the following, we illustrate the search results of our Maximum Probability Product algorithm and compare the results with the corresponding shortest paths using two example queries, and study the average performance of the algorithms. Additionally, we demonstrate the search results of the alternative solution mentioned in subsection V, in which the turning probability defined in Equation 2 is used as the popularity indicator, and we show that this simple alternative option may lead to not accurate enough results. Notice that the 'goodness' of a search result is hard to be measured by some ground truth, and here we just present the results virtually from which we can have an intuitive impression.

Let's denote the search output of our algorithm by 'MPR', the result of the alternative solution by 'MPR-alternative', and the shortest path by 'shortest path'. In the first example query, the most popular route (Figure 6(b)) is almost the same as the corresponding shortest path (Figure 6(a)), where the destination is drawn as a rectangle. However, if we use the alternative solution for finding a popular route, it leads to a route that moves oppositely in the beginning and then winds around to the destination as shown in Figure 6(c), which is intuitively not a good choice for truck deliveries. Consequently, we would say this alternative solution may fail to find a globally popular route as it looks at the turning probabilities of the immediately adjacent edges only, while our algorithm further considers t steps forward. Nevertheless, in many other cases, the MPR and the MPR-alternative may still be very similar, as we can see in Figure 6(e) and Figure 6(f) that are the results of the second example query.

Even though the MPR and the corresponding shortest path are nearly the same in the first query, we can still find a lot scenarios that the MPR is very different from the shortest path since drivers may not always follow the shortest path for truck deliveries. An example is in Figure 6(e) where the MPR deviates to the left, while the shortest path goes straight down to the destination in Figure 6(d). To explain this phenomenon, we may have a look at how trajectories connect to the destination exactly. Figure 9(a) depicts all the trajectories that go to the destination B, and the summary of the trajectory distribution is shown in Figure 9(b). There are totally 14 trajectories go to the destination from the left part while only 2 trajectories go straight down to the destination. Therefore, more truck drivers prefer the route through the left part and that is the reason why the MPR and also the MPR-alternative are found out to follow a different way from the shortest path. Furthermore, this example demonstrates that a preferable MPR is not necessarily to be the shortest path.



(a) Trajectories (b) Summary

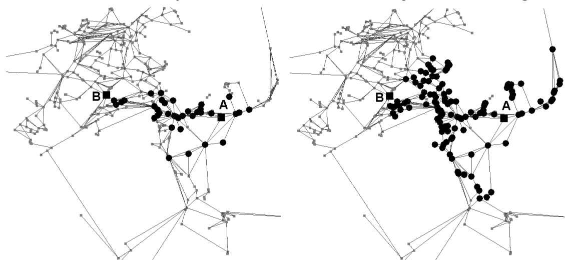
Fig. 9. Statistics of Trajectories

The difference between the MPR and the shortest path (SP) also lies in the number of transfer nodes contained by the route. As shown in Figure 8(a), for those shortest paths that contain 4 or 6 transfer nodes, the corresponding MPRs involve the same number of transfer nodes on average. However, for longer shortest paths that contain more than 6 transfer nodes, the corresponding MPRs normally involve fewer transfer nodes than the shortest paths do, which confirms that truck drivers would like to make fewer transfers in deliveries. In our dataset, the MPR contains 9 nodes on average while the corresponding shortest path consists of 12 nodes on average. Besides, a smaller number of transfer nodes produce a larger product of transfer probabilities, which is another reason that the MPR is with less transfer nodes.

In contrast, the total distance of the MPR is normally larger than that of the corresponding shortest path as illustrated in Figure 8(b). Compared to a shortest path that contains 12 nodes and with a length of 18km, the corresponding MPR is about 1/4 longer on average, which implies the fact that the shortest path is not always the most favorite one and drivers may take a slightly longer route in order to use higher quality roads, or to avoid traffic, or to maximize delivery efficiency, etc. Importantly, the driver behaviors can be partially discovered by searching the most popular routes.

D. The Efficiency of Searching the MPR

The efficiency of the Maximum Probability Product algorithm is recorded in Figure 8(c) and 8(d) respectively, where the performance is measured by query time and the number of transfer nodes that are visited during the search. It is interesting to observe that the search of the MPR requires less time than the Dijkstra's shortest path algorithm does. In Figure 8(c), the query time consumed by the Maximum Probability Product algorithm is approximately half of the query time consumed by the shortest path algorithm. The origin is the number of transfer nodes visited during the search. Generally, while the Dijkstra's shortest path algorithm expands the network outwards from the start node in a circle shape, the Maximum Probability Product algorithm is like a biased search towards the destination which is similar to the A* algorithm [23], because the transfer nodes on the way to the destination probably maintain a higher transfer probability in comparison with those nodes in a wrong direction. Therefore, the search region of the Maximum Probability Product algorithm is much smaller as we can confirm in Figure 10, where the visited nodes of the search ($A \to B$) are marked by circle dots. For the MPR-alternative, it has a performance in-between the Maximum Probability Product and the Dijkastra's algorithms.



(a) The MPR (b) The Shortest Path

Fig. 10. The Search Regions of the MPR and the Shortest Path

VII. Conclusions

In this paper, we study the problem of discovering the most popular route between any two given locations, by considering previous users' traveling trajectories. We propose a Coherence Expanding algorithm for mining a transfer network from trajectories and develop a reasonable popularity indicator for measuring the popularity of transfer nodes w.r.t. a designated destination. Based on the popularity indicator, the Maximum Probability Product algorithm is presented for searching the most popular route. In our experiments, we demonstrate the most popular routes discovered by our algorithm, with comparison to the corresponding shortest paths. Although there is no ground truth for verification, we virtually and quantitatively examine the search results and the algorithm performance.

References

[1] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. P. Sondag, "Adaptive fastest path computation on a road network: a traffic mining approach," in *VLDB*, 2007, pp. 794–805.

[2] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, "Trajectory pattern mining," in *SIGKDD*, 2007, pp. 330–339.

[3] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti, "Wherenext: a location predictor on trajectory pattern mining," in *SIGKDD*, 2009, pp. 637–646.

[4] H. Jeung, Q. Liu, H. T. Shen, and X. Zhou, "A hybrid prediction model for moving objects," in *ICDE*, 2008, pp. 70–79.

[5] C. M. Grinstead and J. L. Snell, *Introduction to Probability*, 2nd ed. American Mathematical Society, 1997.

[6] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Math*, vol. 1, pp. 269–271, 1959.

[7] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung, "Mining, indexing, and querying historical spatiotemporal data," in *SIGKDD*, 2004, pp. 236–245.

[8] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *WWW*, 2009, pp. 791–800.

[9] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: a partition-and-group framework," in *SIGMOD*, 2007, pp. 593–604.

[10] J.-G. Lee, J. Han, X. Li, and H. Gonzalez, "Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering," *PVLDB*, vol. 1, no. 1, pp. 1081–1094, 2008.

[11] D. Sacharidis, K. Patroumpas, M. Terrovitis, V. Kantere, M. Potamias, K. Mouratidis, and T. Sellis, "On-line discovery of hot motion paths," in *EDBT*, 2008, pp. 392–403.

[12] X. Li, J. Han, J.-G. Lee, and H. Gonzalez, "Traffic density-based discovery of hot routes in road networks," in *SSTD*, 2007, pp. 441–459.

[13] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu, "Prediction and indexing of moving objects with unknown motion patterns," in *SIGMOD*, 2004, pp. 611–622.

[14] M. Ester, H.-p. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *SIGKDD*, 1996, pp. 226–231.

[15] L. Cao and J. Krumm, "From gps traces to a routable road map," in *SIGSPATIAL*, 2009, pp. 3–12.

[16] A. Fathi and J. Krumm, "Detecting road intersections from gps traces," in *GIScience*, 2010.

[17] M. Hua and J. Pei, "Probabilistic path queries in road networks: traffic uncertainty aware path selection," in *EDBT*, 2010, pp. 347–358.

[18] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *SIGMOD*, 2005, pp. 491–502.

[19] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *ICDE*, 2002, pp. 673–684.

[20] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches in query processing for moving object trajectories," in *VLDB*, 2000, pp. 395–406.

[21] R. Sherkat and D. Rafiei, "On efficiently searching trajectories and archival data for historical similarities," *PVLDB*, pp. 896–908, 2008.

[22] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie, "Searching trajectories by locations – an efficiency study," in *SIGMOD*, 2010.

[23] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A* search meets graph theory," in *SODA*, 2005, pp. 156–165.

[24] B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in *EDBT*, 2008, pp. 205–216.

[25] E. Kanoulas, Y. Du, T. Xia, and D. Zhang, "Finding fastest paths on a road network with speed patterns," in *ICDE*, 2006, p. 10.

[26] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, "On map-matching vehicle tracking data," in *VLDB*, 2005, pp. 853–864.

[27] "Openstreetmap," http://www.openstreetmap.org/.

[28] M. M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[29] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *SIGMOD*, 1984, pp. 47–57.

[30] L. Lovász, "Random walks on graphs: A survey," *Combinatorics, Paul Erdős is Eighty*, vol. 2, pp. 1–46, 1993.

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.