Popular Route Planning with Travel Cost Estimation

Huiping Liu, Cheqing Jin(✉), and Aoying Zhou

School of Computer Science and Software Engineering, Institute for Data Science and Engineering, East China Normal University, Shanghai, China
hpliu@stu.ecnu.edu.cn,{cqjin,ayzhou}@sei.ecnu.edu.cn

Abstract. With the increasing number of GPS-equipped vehicles, more and more trajectories are generated continuously, based on which some urban applications become feasible, such as route planning. In general, route planning aims at finding a path from source to destination to meet some specific requirements, i.e., the minimal travel time, fee or fuel consumption. Especially, some users may prefer popular route that has been travelled frequently. However, the existing work to find the popular route does not consider how to estimate the travelling cost. In this paper, we address this issue by devising a novel structure, called popular traverse graph, to summarize historical trajectories. Based on which an efficient route planning algorithm is proposed to search the popular route with minimal travel cost. The extensive experimental reports show that our method is both effective and efficient.

1 Introduction

Route planning is important not only for our daily life, but also for business map engines like Google and Bing Maps [1–4]. Although the shortest/fastest paths are used commonly, they may be insufficient in some situations, while the popular route that refers to a path being travelled frequently is sometimes important. For example, drivers who travel in an unfamiliar city may prefer a popular path which may be safer with better traffic condition and road quality, and a taxi passenger may want to travel along a popular path in case of a roundabout trip. Moreover, people care more about the travel cost, i.e., how long it takes or how much it costs. An accurate travel cost estimation will improve people's satisfaction. In practice, there are several popular paths with different travel cost from source to destination, so a popular route with the minimal travel cost which saves resource consumption (such as time, money, fuel) is a better choice.

Route planning or driving direction planning has been studied in recent years and some influential works have been published. [3] proposes a framework to find out the practically fastest route at a given departure time based on a landmark graph learned from a large number of historical taxi trajectories. However, the fastest route is not always popular, some shortcuts may reduce the travel time but increase the risk and uncertainty of a trip. The work performs a two-stage routing algorithm based on the graph to find the fastest route. The first stage

© Springer International Publishing Switzerland 2016
S.B. Navathe et al. (Eds.): DASFAA 2016, Part II, LNCS 9643, pp. 403–418, 2016.
DOI: 10.1007/978-3-319-32049-6_25

is to find the rough route represented by a sequence of landmarks whose travel time can be estimated by their model and the second stage is to find a practically detailed fastest route in the road network based on speed constraints. But the travel time of the detailed fastest route may be different from the estimated travel time at the first stage. Since there may exist several different paths between two landmarks and the estimated travel time is just the mean travel time of all possible paths. In most cases, the travel time of the detailed fastest route is less than the estimated travel time, which will cause an inaccurate travel time estimation of the route. Moreover, the model proposed in [5] to estimate the travel time of a given path by using the optimal route concatenation cannot be applied to route planning directly.

In this paper, we aim at finding the popular route with minimal travel cost from source to destination and estimating the travel cost for this route. We propose a framework to achieve this goal. Firstly, we construct a popular traverse graph based on the historical trajectories, where each node is a popular location, and each edge is a popular route between two locations. Subsequently, for each popular route in this graph, we use the minimum description length (MDL) principle [6] to model the travel cost of the routes. Finally, based on the graph, given a source-destination pair and a leaving time, we find the fastest popular route in consideration of the optimal route concatenation [5] for an accurate travel cost estimation. The contributions of this paper are summarized below.

– We propose a novel structure, called popular traverse graph, from trajectories without road network information, which contains the popular routes between locations.
– We present a parameter-free method using the minimum description length (MDL) principle to model the travel cost on each popular route in the graph.
– We devise an efficient routing algorithm which combines optimal route concatenation with route planning on the popular traverse graph.
– We have conducted extensive experiments upon a real dataset of millions of trajectories generated by more than 10000 taxis over a month in Beijing. The results show that our method is both effective and efficient.

The remainder of the paper is organized as follows. Section 2 reviews the related work. Section 3 describes some preliminary knowledge. Section 4 illustrates the popular traverse graph, and a way to model the travel cost for each popular route. Section 5 details the routing algorithm. Section 6 reports the evaluation and a brief conclusion is given in Sect. 7.

2 Related Work

Route planning has been widely investigated in recent years, including popular route planning and the fastest route planning.

Popular Route Planning: [7,8] discover the top-k possible popular routes that sequentially pass given locations from historical uncertain trajectories. [9] studies how to discover the most popular route between any two locations. The authors introduce a transfer network model by exploiting intersections and the popularity of transfer nodes on the transfer network. They infer the most popular routes according to the turning probability of each intersection on the network. [10] tries to find the time period-based most frequent path. It firstly constructs a footmark graph which is used to calculate the frequencies of candidate paths, then they retrieve the most frequent path in arbitrary time periods specified by the users on this graph. All the above studies try to find the popular routes without considering the travel cost, whereas our focus is to find a popular route with the minimal travel cost.

The Fastest Route Planning: [11] proposes a fundamental algorithm (Dijkstra's algorithm) to find the shortest path between two nodes in a graph and the A^* algorithm proposed by [12] boosts the searching performance with a heuristic estimation. In [1,2,13], the authors study how to find the fastest path on a time-dependent graph. [14] computes the fastest path on a road network by considering speed and driving patterns. Yuan et al. proposed a framework to find the fastest route from taxi trajectories [3,4]. In [3], they construct a landmark graph and based on which, a two-stage routing algorithm is performed to find the fastest route. In [4], traffic prediction is employed for optimization. However, the estimated travel time is not actually the travel time of the practical fastest route as [3,4] recommended. [15] studies stochastic skyline route queries in road networks with multiple travel costs. The authors provide the travel cost distribution given a source-destination pair with a leaving time. [5] proposes a model to estimate the travel time of a given path in a road network, but it's unsuitable for route planning from a source to a destination. Our work significantly differs from the above methods, because we aim at finding the popular route and estimating the travel cost.

3 Preliminary

We define some terms and the problem addressed in this paper.

Definition 1 (*Trajectory*). A *Trajectory* Tr is a time-ordered sequence of points generated by a moving object. Each point p consists of a geographic location $p.l$ and a timestamp $p.t$, i.e., $Tr : p_1 \rightarrow p_2 \rightarrow ... \rightarrow p_n$, where $p_{i+1}.t > p_i.t$ $(1 \leq i < n)$.

The trajectory [16,17] is a real reflection of the travelling behaviour of the moving object and provides us a possible path from the start to the end location if no road network information is provided.

Definition 2 (*Popular Route*). A *Popular Route* is a path with alternative condition holds: (1) A path that has been traversed at least τ times, where τ is a pre-defined threshold; (2) A path consists of multiple sub-paths and each sub-path satisfies condition (1).

A popular route is always a candidate for a trip, because most of people have chosen it. However, not all source-destination pair have a direct popular route (condition 1). For instance, a long route as a whole may not be frequently passed. In this case, it can still be treated as a popular route if all its sub-paths are popular routes (condition 2) [9].

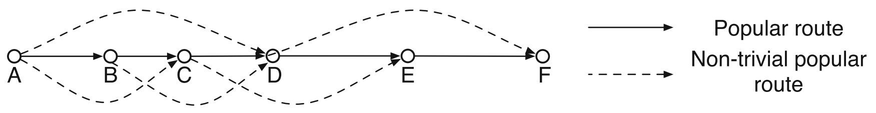


Fig. 1. An example of route concatenation

Definition 3 (*Popular Traverse Graph*). A *Popular Traverse Graph (PTG) G* = (*V, E*) is a directed graph where *V* is a set of popular locations and *E* is a set of popular routes between locations.

Since each edge in PTG is a popular route, the path between any two nodes is also popular by Definition 2.

Definition 4 (*Concatenation of Route*). Let $r : n_1 \rightarrow n_2 \rightarrow ... \rightarrow n_i$ denotes a route in the popular traverse graph G, where n_k $(1 \leq k \leq i)$ is a node in G. Denote $|r|$ as the size of r which is the number of nodes it contains. Route $r^* : n_1 \rightarrow ... \rightarrow n_i$ is a concatenation of r if $|r^*| \leq |r|$.

The concatenation of a route means some consecutive road segments in the route are frequently traversed as a whole, and we can regard the consecutive road segments as a united road segment when we estimate the travel cost of this route. Since the united road segment covers the entire path, it reflects the traffic conditions of this whole path, including intersections, traffic lights and direction turns, which will improve the accuracy of the travel cost estimation.

Example 1. Figure 1 is a route on PTG. For a sub-route $r : A \rightarrow B \rightarrow C \rightarrow D$, if r is travelled frequently as a whole, then r is a popular route due to the condition (1) in Definition 2. We call r the *non-trivial popular route*. Then $r^* : A \rightarrow D$ is a concatenation of r (there are 4 in total for r). Obviously, only non-trivial popular route has different concatenations and each non-trivial popular route r has $|r| > 2$. Moreover, every sub-route r' of r with $|r'| > 2$ is also a non-trivial popular route, such as $A \rightarrow B \rightarrow C$ and $B \rightarrow C \rightarrow D$.

The next issue is how to find an optimal one from different concatenations for a route. [5] finds the optimal concatenation of a path by making an object function minimized. In this paper, we apply the object function: $\sum_{i=1}^{k} \frac{1}{n_{s_i}} Var(c_{s_i})$, where s_i is the ith segment of the path, $Var(c_{s_i})$ is the variance of the travel costs on s_i, and n_{s_i} is the number of trajectories that travelled on s_i.

Problem Definition: Given a popular traverse graph G and a route planning query with a source s, a destination d and a departure time t, we find the popular route with minimal travel cost in regard to the optimal route concatenation.

4 Popular Traverse Graph

This section first describes how to construct a popular traverse graph from trajectories without road network information, and then details the travel cost modelling of popular routes.

4.1 Constructing the Popular Traverse Graph

As the road network may be unavailable in some situations [8,9], it is infeasible to compute the travelling frequency of the roads to find the popular routes. Fortunately, it is still feasible to find the popular locations (been visited frequently) from the trajectories, so that the transitions between locations can be extracted. Finally, we discover the popular routes on each transition to construct the popular traverse graph. The major processes are listed below.

Popular locations mining. Since the trajectories consist of points, the popular locations come from the points of trajectories. To reduce the size of points, we only consider the end points of the trajectories which can reflect the locations where people usually start from or go to. Thus, the query source or destination has a high probability to locate at the locations. To find the popular locations, we first partition the points into different zones and then the DBSCAN clustering algorithm [18] is invoked to generate clusters for the points in each zone. Finally, k clusters with the maximal number of points are chosen as the popular locations.

Transitions extraction. We then search the transitions between locations by scanning the trajectory dataset. For each trajectory tr, we first map it to the popular locations, then the trajectory can be represented as tr: $l_1 \rightarrow l_2 \rightarrow ... \rightarrow l_n$, where l_i $(1 \leq i \leq n)$ is a popular location. For each $l_i \rightarrow l_{i+1}$ $(1 \leq i < n)$, we generate a transition from l_i to l_{i+1} if it does not exist, and we also keep the segment of tr belongs to this transition for popular routes discovering.

Popular routes discovering. There may exist several different paths for a transition that connects two popular locations. In order to get the popular paths on the transitions, we cluster the trajectory segments pertaining to the transition. As a result, the cluster with at least τ trajectories is treated as popular route. To tackle the ununiform rate trajectories, we apply Edit Distance with Projections (EDwP) proposed in [19] to measure the similarity between trajectories, and we use the density-based method [20] to cluster the trajectories.

Note that trajectory clustering can not only discover the popular routes but also detect the outliers in the trajectory set (i.e., the roundabout trips), which helps to improve the accuracy of the travel cost estimation.

4.2 Modeling Travel Cost Using the MDL Principle

It is challenging to estimate the travel cost for an arbitrary path [3,15,21], since the cost of a route at different time varied a lot. The major difficulty is how to partition the travel costs into different time slots. [15,21] divide the time slots statically. The VE-Clustering algorithm [3] attempts to partition a day into different time slots according to the travel costs by two clustering methods, namely V-clustering and E-clustering, and each with a threshold. However, it is hard to set the values of such global thresholds for all routes. We note that a good partition for the travel costs should meet the following two requirements: (1) *homogeneity*: stable travel costs in each time slot, and (2) *conciseness*: the significant difference (i.e., distribution) between two adjacent time slots, otherwise, there is no need to split them apart. Fortunately, although challenging, the minimum description length (MDL) principle [6] which is widely used in information theory is suitable to solve this problem. Hence, we propose a parameter-free algorithm by integrating the MDL principle.

Let's review MDL briefly [6]. The MDL cost consists of two components: $L(H)$ and $L(D|H)$, where $L(H)$ is the length of the description of the hypothesis, and $L(D|H)$ is the length of the description of data under the hypothesis, both in bits. To get the best hypothesis H to explain the data D, the value of $L(H) + L(D|H)$ must be minimized. In our work, H refers to the time partition and D is the set of travel costs along the popular route. Hence, $L(H)$ and $L(D|H)$ are defined formally below.

$$L(H) = \log_2(num) + \sum_{i=1}^{num} \lceil \log_2 \mathsf{span}(slot_i) \rceil \tag{1}$$

$$L(D|H) = \sum_{i=1}^{num} \{ \lceil \log_2(N(slot_i) + 1) \rceil + \mathsf{Ent}(slot_i) \} \tag{2}$$

where $\mathsf{Ent}(slot_i) = -\sum_{k=1}^{num_{cla}} \frac{N_k(slot_i)}{N(slot_i)} \log_2 \frac{N_k(slot_i)}{N(slot_i)}$. Equation (1) encodes the hypothesis of a partitioning, the first term describes the number of partitions (num) on time, the second term describes the span of each time slot ($slot_i$). Equation (2) describes data under the hypothesis. The first term encodes the number of the travel costs in $slot_i$[1]. The second term computes the information entropy to describe the stability of the slot, which needs to map the travel costs to different class first, each with a different cost level. For example, in travel time cost, the costs in seconds can map to minute level, that is costs from 0 to 120 s correspond to two classes, which are (0,60] and (60,120].

Obviously, $L(H)$ stands for *conciseness* and $L(D|H)$ for *homogeneity*. The more homogenic and concise, the better. However, *conciseness* and *homogeneity* are contradictory, hence we need to find the optimal trade-off by minimizing $L(H) + L(D|H)$. As it is expensive to compute the minimal value of $L(H) + L(D|H)$ due to too many potential partitionings, we devise an approximate

[1] To avoid the case that $N(slot_i) = 0$, we increase the value by 1.

Algorithm 1. approPartition(D)

```
1  $S \leftarrow \{[t_1, t_2]\}$, where $t_1$ and $t_2$ are the earliest and latest time in $D$;
2  $minCost \leftarrow MDL(S)$        // $MDL(S) = L(H) + L(D|H)$;
3  while $true$ do
4      $s[t_1, t_2] \leftarrow$ a slot in $S$ with the maximal value of $\mathsf{Ent}(\cdot)$;
5      $t \leftarrow \arg\min_{t \in [s.t_1, s.t_2]}(\mathsf{Ent}([s.t_1, t]) + \mathsf{Ent}([t, s.t_2]))$;
6      $S' \leftarrow (S - \{s\}) \cup \{[s.t_1, t], [t, s.t_2]\}$, $newCost \leftarrow MDL(S')$;
7      if $newCost < minCost$ then
8          $minCost \leftarrow newCost$, $S \leftarrow S'$;
9      else
10         return $S$;
```

solution instead, as shown in Algorithm 1. This algorithm accepts a travel costs set D as input. Initially, a set S only contains one time slot that covers the whole data set D (at line 1). Let *minCost* denote the MDL cost of the current situation. We then probe the dataset S in greedy manner (at lines 3–10). At each time, we find a time slot s in S with the maximal entropy to split. Subsequently, the optimal splitting point t ($t \in [s.t_1, s.t_2]$) is found by minimizing the sum of entropies. Hence, the original time slot s in S is replaced by two new slots $[s.t_1, t]$ and $[t, s.t_2]$. The iteration will stop when $newCost \geq minCost$.

The time complexity of Algorithm 1 is $O(n \cdot k)$, where k is the number of time slots and n is the size of the dataset D, since it computes the MDL cost by scanning the whole dataset at each iteration. In this paper, we use the average travel cost in each time slot to represent the cost of it.

Example 2. Figure 2 illustrates an example of the popular traverse graph. The nodes A, B, C, D, E are popular locations and solid lines are the popular routes between them. The dash lines stand for the non-trivial popular routes mined from the trajectory dataset (see Sect. 5.2). For simplicity, we use a static number to represent the cost on each popular route and we assume that only one popular route exists on each transition.

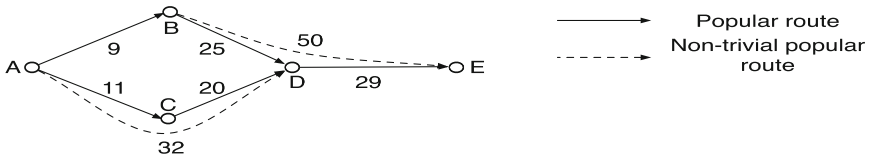


Fig. 2. An example of PTG

5 Route Planning on the Popular Traverse Graph

In this section, we introduce how to find the popular route with the minimal travel cost in consideration of the optimal route concatenation on the time-dependent popular traverse graph.

5.1 Routing Algorithm

Recall that we consider the optimal concatenation of route for accurate travel cost estimation, therefore the result route for a query should satisfy two conditions: (1) it's travel cost should be the cost of its optimal concatenation, (2) it spends minimal travel cost under condition (1). That is the route we expected has the optimal concatenation with the minimal travel cost.

Example 3. We show an example of route planning on PTG illustrated by Fig. 2. Suppose the non-trivial popular routes are also the optimal concatenations. Then the cost of route $A \rightarrow C \rightarrow D$ should be 32, since its optimal concatenation $A \rightarrow D$ costs 32, the same to $B \rightarrow D \rightarrow E$ which costs 50. So, the expected route from A to E should be $A \rightarrow B \rightarrow D \rightarrow E$ which has the minimal travel cost 59. An interesting observation is that the cost of sub-route on $A \rightarrow B \rightarrow D \rightarrow E$ may not be the minimal. For example, $A \rightarrow B \rightarrow D$ costs more than $A \rightarrow C \rightarrow D$ from A to D. That is the major difference from the shortest route planning.

A naive solution is to find all routes with optimal concatenation from source to destination, and choose the one with minimal travel cost. However, it's infeasible to enumerate all the possible routes with optimal concatenation, the cost of it is prohibitive. Instead, we propose a method to return the expected route more efficiently, as listed in Algorithm 2.

Given a PTG $G = (V, E)$ and the query $q = (s, d, t)$, Algorithm 2 returns the expected route. We use *cost*, *route* and *routeSet* for each node to keep the minimal cost, the optimal route and the routes have been considered from s respectively. In addition, we maintain a priority queue Q for nodes on PTG sorted by *cost* in ascending order. Initially we set *n.cost* to maximum, *n.route* to *null* and *n.routeSet* to *empty* except for the source node s, and add s to Q (lines 1–3). Then we keep searching until Q is empty or the destination node d is settled (lines 4–14). At each iteration, we get the head node n of Q and find its outgoing popular routes set where each route is either an outgoing edge of n on PTG or a non-trivial popular route (line 6). For each outgoing route of n, we consider the entire route from s to v passed n. If the route has not been checked, we then find its optimal concatenation and the corresponding travel cost. We keep the route in *v.routeSet* to avoid verifying twice (lines 7–11). If the new cost of v is less than its current cost, then we update v' cost and its route, and add v to Q if it's not in it (lines 12–14). In this way, we will get the route whose optimal concatenation has minimal travel cost by Lemma 1.

Lemma 1. *Algorithm 2 can return the expected route on popular traverse graph* $G = (V, E)$ *for a query* $q = (s, d, t)$ *if the route exists.*

Algorithm 2. routePlanning $(G = (V, E), s, d, t)$

```
1  foreach For each $n$ in $V$ do
2      $n.cost \leftarrow \infty$, $n.route \leftarrow null$, $n.routeSet \leftarrow \emptyset$;
3  $s.cost \leftarrow 0$, $s.route \leftarrow s$;
4  Create a new priority query $Q$, $Q.enqueue(s)$;
5  while $n \leftarrow Q.dequeue()$ and $n \neq NULL$ do
6      return $d.route$ if $n = d$;
7      $S \leftarrow \{r : n \rightarrow ... \rightarrow v | v$ is not settled and $r$ is a non-trivial popular route or
       an outgoing edge of $n\}$;
8      foreach route $r : n \rightarrow ... \rightarrow v$ in $S$ do
9          $r' \leftarrow n.route + r$;
10         if $r' \notin v.routeSet$ then
11             $r^* \leftarrow$ the optimal concatenation of $r'$;
12             $v.routeSet \leftarrow v.routeSet \cup \{r'\}$;
13             if $r^*.cost < v.cost$ then
14                 $v.cost \leftarrow r^*.cost$, $v.route \leftarrow r'$;
15                 $Q.enqueue(v)$ if $v$ is not in $Q$;
```

Proof. It is obvious that only the cost of optimal concatenations can be considered by Algorithm 2, and then we need to prove that the route returned by Algorithm 2 has minimal travel cost. We prove it by contradiction. Suppose there is a route whose optimal concatenation is $r' : s \rightarrow ... \rightarrow n_i \rightarrow d$ has less travel cost than the route r^* returned by Algorithm 2. Then the route $s \rightarrow ... \rightarrow n_i$ must cost less than r^*, too. That is n_i must have been settled before returning r^*, hence r' has been checked (by line 6). Since we choose the less cost optimal route concatenation, that is r^* has less cost than r', which is a contradiction. Thus the lemma is proved.

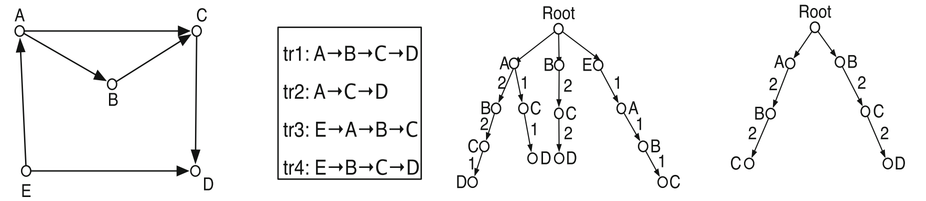
Example 4. We take Fig. 2 as the input PTG, and we find the expected route from A to E by Algorithm 2. We begin with A, and we check $A \rightarrow B$ and $A \rightarrow C$ whose optimal concatenation is themselves and $A \rightarrow C \rightarrow D$ with optimal concatenation $A \rightarrow D$. Then the cost of B, C and D will be 9, 10 and 32 respectively. We next settle B and check its outgoing routes, then the cost of E will be 59 with route $A \rightarrow B \rightarrow D \rightarrow E$. For C, since $A \rightarrow C \rightarrow D$ has been checked, we move to D whose cost is 32 with route $A \rightarrow C \rightarrow D$. Then we consider $A \rightarrow C \rightarrow D \rightarrow E$ whose cost is 61. Finally, we get the expected route $A \rightarrow B \rightarrow D \rightarrow E$ with cost 59.

Although Algorithm 2 can return the right route, the complexity is relatively high, especially in line 6 and line 10 when finding the non-trivial popular routes and computing the optimal concatenation of a route. We improve the procedures by introducing a suffix tree.

5.2 Indexing for Non-trivial Popular Routes

In this sub-section, we describe how to construct a suffix-tree-based index from trajectories dataset to quickly retrieve the non-trivial popular routes between nodes on PTG. For a route $r : n_1 \rightarrow n_2 \rightarrow ... \rightarrow n_k$, we define *prefix route* of r as the route $r' : n_1 \rightarrow ... \rightarrow n_i (1 < i < k)$, and *suffix route* of r as the route $r^* : n_i \rightarrow ... \rightarrow n_k (1 < i < k)$

For each trajectory, we first map it to a string of nodes on PTG, i.e., *tr*: $n_1 \rightarrow n_2 \rightarrow ... \rightarrow n_k$. For each $n_i \rightarrow n_{i+1} (1 \leq i < k)$, if it's not a popular route on PTG, then we know that $n_i \rightarrow n_{i+1}$ won't be a popular route or be a part of popular route, hence it will be deleted from *tr*. Now *tr* is split into several segments. For each segment $s : n_i \rightarrow ... \rightarrow n_j$, if $|s| > 2$, then it could be a non-trivial popular route. We create a path on the suffix-tree for each suffix route of s: $n_o \rightarrow ... \rightarrow n_j (i \leq o \leq j - 2)$ if it's not exists in the tree, otherwise, we increase its support. By this way, any path p with $|p| > 2$ in the suffix-tree whose support is no less than the threshold τ will be a non-trivial popular route. To reduce the size of the suffix-tree, we remove the paths with support less than τ. Given a starting node, we can find all the non-trivial popular routes in $O(1)$ with the reduced suffix-tree.



a) The popular traverse graph b) Trajectories dataset c) Suffix-tree d) Reduced suffix-tree with $\mathcal{T}$

Fig. 3. An example of constructing suffix-tree

Example 5. Figure 3 shows an running example of constructing a reduced suffix-tree based on the PTG and trajectories dataset with threshold $\tau = 2$. As we can see, there are two non-trivial popular routes in total which are $A \rightarrow B \rightarrow C$ and $B \rightarrow C \rightarrow D$.

5.3 Computing the Optimal Concatenation

As mentioned above, it is time-consuming when finding the optimal concatenation of a route with many different concatenations. In this paper, we pre-compute the optimal concatenations of the non-trivial popular routes on the reduced suffix-tree to improve the efficiency. For a non-trivial popular route $r : n_1 \rightarrow ... \rightarrow n_k$, there exist 2^{k-2} possible concatenations, we compute its optimal concatenation by using dynamic programming solution with complexity

$O(k^2)$. That is we calculate the minimal object function value (mentioned in Sect. 3) for each node by $opt_i = \mathsf{arg\ min}_{1\leq j<i}(opt_j + (n_j \rightarrow n_i).obj)(1 < i \leq k)$ with $opt_1 = 0$, where opt_i is the minimal object function value of each node, and obj is the object function value of the route.

With the optimal concatenations of non-trivial popular routes in the reduced suffix-tree, we compute the cost of the optimal concatenation of a route on the PTG, which is described in Algorithm 3. For each node in the route, we define *ocCost* as the cost of the optimal concatenation from the source. We maintain a priority queue for the nodes by their order in the route. For each node n_i from the queue, we only consider the node n_j that n_i can reach as far as possible on the route through r', where r' is a non-trivial popular route whose optimal concatenation has been found or an outgoing edge of n_i whose optimal concatenation is itself, then we refine n_j by r' using Algorithm 4 and add it to the queue (lines 5–6). For the nodes (except for n_i and n_j) in r', we refine the nodes that can reach beyond r' by a non-trivial popular route and add them to the queue (lines 7–10). Finally, We return the result until we get to the ending node. By using the pre-computed results of non-trivial popular routes, we save lots of computing while refining the nodes. The complexity of Algorithm 3 will be $O(|r|)$ since the worst case is that every node on the route is in the queue.

Algorithm 3. ocOnPTG $(r : n_1 \rightarrow ... \rightarrow n_k)$

```
1  $n_1.ocCost \leftarrow 0$, $n_1.opt \leftarrow 0$, $n_i.opt \leftarrow \infty$ for $1 < i \leq k$;
2  Initialize a priority queue $Q$, $Q.enqueue(n_1)$;
3  while $n_i \leftarrow Q.dequeue()$ and $n_i \neq NULL$ do
4      return $n_i.ocCost$ if $n_i = n_k$;
5      $r' \leftarrow \mathsf{arg\ max}_{i<j\leq k}|r : n_i \rightarrow ... \rightarrow n_j|$, where $r$ is either a non-trivial popular
       route or an outgoing edge of $n_i$;
6      Refinement$(r')$, $Q.enqueue(n_j)$;
7      foreach $o = i + 1\ ...\ j - 1$ do
8          if $r^p : n_o \rightarrow ... \rightarrow n_p (j < p \leq k)$ is a non-trivial popular route then
9              refinement$(n_i \rightarrow ... \rightarrow n_o)$;
10             refinement$(r^p)$, $Q.enqueue(n_p)$;
```

Algorithm 4. refinement $(r : n_i \rightarrow ... \rightarrow n_j)$

```
1  $obj \leftarrow n_i.opt + r.opt$;
2  if $obj < n_j.opt$ then
3      $n_j.opt \leftarrow obj$, $n_j.ocCost \leftarrow n_i.ocCost + r.ocCost$;
```

Example 6. Take Fig. 1 as an example. The nodes in the queue will be D, E, and F when compute the cost of the optimal concatenation from A to F.

6 Experiments

We report some experimental results in this section. Without loss of generality, in our experiments, we focus on travel time cost. All codes are written in Java, run at a computer with dual core 2.00 GHz CPU and 16 GB main memory.

We use a real-life dataset containing 7,122,320 paid trajectories generated by 13,007 taxicabs in Beijing from Oct. 1 to Oct. 31, 2013. The sampling rate is about 1 point per minute. The dataset is divided into two parts. The first part (the first 21 days) is used to construct a popular traverse graph ($\tau = 100$). The final PTG has 5,000 nodes and 33,357 popular routes, and the suffix-tree based on the PTG contains 210,338 different non-trivial popular routes. Such steps are implemented offline. The second part (the last 10 days) is used for evaluation. We randomly choose 30,000 trajectories from the dataset as the queries and regard the travel time as the ground truth. The length of the trajectories ranges from 3 km to 18 km, and the travel time varies from 3 min to 60 min.

As T-drive [3] is known to find the fastest route between two points at a departure time, we use it on PTG as the baseline method. Moreover, MDL+Dijkstra refers to our basic method that applies MDL to model the travel cost and Dijkstra algorithm on TPG to search route, and MDL+OC refers to our method that applies MDL and the optimal route concatenation.

6.1 Effectiveness

We use mean absolute error (MAE) and mean relative error (MRE) to measure the effectiveness of our method. Equation (3) describes the common definitions, where y_i is an estimate, $\widehat{y}_i$ is the ground truth and n is the number of samples.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \widehat{y}_i|}{n}, \quad MRE = \frac{\sum_{i=1}^{n} |y_i - \widehat{y}_i|}{\sum_{i=1}^{n} \widehat{y}_i} \tag{3}$$

Effectiveness of MDL. Table 1 compares MAE and MRE between MDL+Dijkstra and T-drive. Recall that T-drive needs to set two parameters in advance, one for V-Clustering and the other for E-Clustering. We consider 12 different parameter settings, and in the best case, MAE = 74.9 s and MRE = 0.224. Note that MDL+Dijkstra has MAE of 71.8 s and MRE of 0.214. Obviously, MDL+Dijkstra outperforms T-drive, since the MDL method we proposed can find an approximate optimal trade-off between homogeneity and conciseness on every popular route, which guarantees a good result. We use the best parameter setting which is 50 and 30 for T-drive in the following experiments.

Overall Effectiveness. We generate three groups of queries, each with top 10,000, 20,000 or 30,000 queries according to the length of trajectories in descending order, as shown in Table 2. Figure 4 shows the performance for these groups. Clearly, MDL+Dijkstra and MDL+OC behave better than T-drive, since MDL principle is suitable for travel cost modelling. Moreover, the performance gain

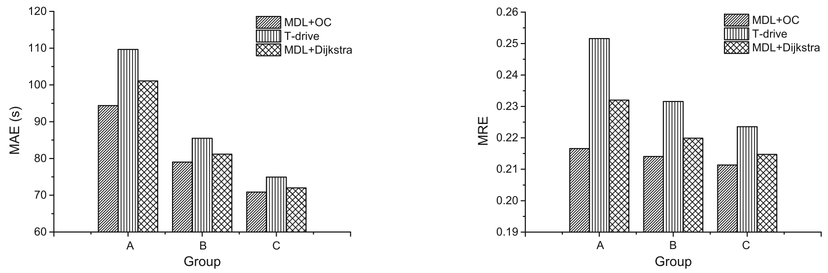
Table 1. Comparison of MDL+Dijkstra and T-drive

Methods	MDL+Dijkstra	T-Drive ($a = 10, b = 30, c = 50, d = 100$)											
		b, a	b, b	b, c	b, d	c, a	c, b	c, c	c, d	d, a	d, b	d, c	d, d
MAE (sec.)	**71.8**	75.3	75.0	76.1	77.9	75.5	**74.9**	76.1	77.9	75.6	75.0	76.2	78.2
MRE	**.214**	.225	.224	.227	.232	.225	**.224**	.227	.232	.226	.224	.227	.233

Table 2. Statistics of three query groups

Group	**Size**	**Avg time (s)**	**Avg length (m)**
A	10,000	435	6,226
B	20,000	369	5,266
C	30,000	335	4,751

of MDL+OC rises when the query distance increases. Since MDL+OC considers the optimal concatenation while routing and the longer the distance is, the more concatenations of the route we can find, which will benefit the travel time estimation. Although our method performs better on long distance query comparing to baselines, we cannot make sure that all queries follow the same route as we found, and the longer the distance is, the higher probability the real path will choose a different route. That's why the MRE increases with the increment of query distance.



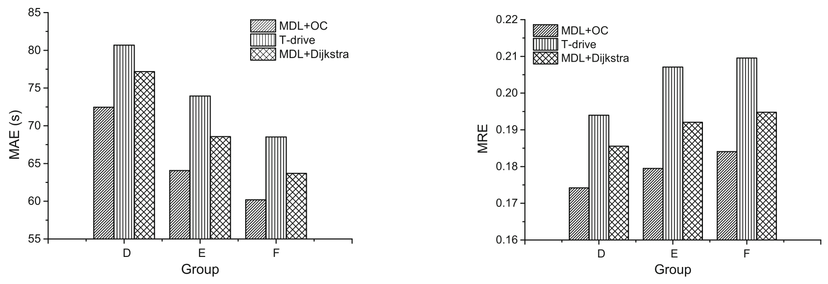
(a) MAE of different methods (b) MRE of different methods

Fig. 4. Comparison of different methods on query groups

An interesting finding is that around 60–70 % queries follow the same route as we found in each group. Hence, we refine each group, and only reserve the ones along the same route as we returned, and generate three new groups (see Table 3). Figure 5 shows the performance on these groups. Compared with Fig. 4, the performance gain of our method rises. Clearly, MDL+OC outperforms all the baselines in terms of the two metrics, and it has significant advantage over the baselines in each query group. Figure 5(a) illustrates the MAE of different methods. As the length decreases, the MAE decreases, too. Conversely, in Fig. 5(b),

Table 3. Statistics of refined query groups

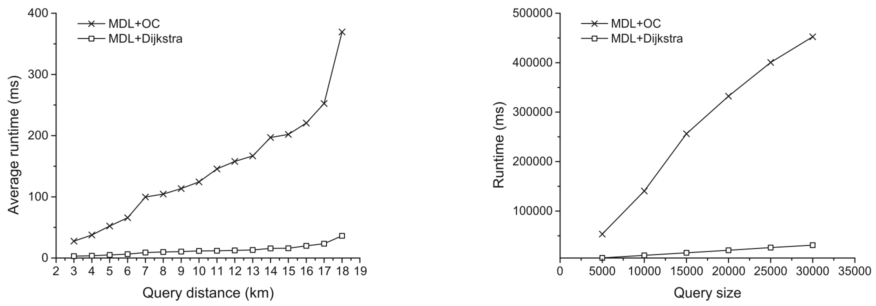
Group	Size	Avg time (s)	Avg length (m)	Percentage (%)
D	6,238	416	6,163	62.38
E	13,691	357	5,109	68.46
F	20,930	327	4,629	69.77



(a) MAE of different methods (b) MRE of different methods

Fig. 5. Comparison of different methods on refined query groups

the MRE of all methods increases as the length of query decreases. This is because the shorter a path is, the more unstable its travel time could be, since the traffic conditions will have a big influence on it. On the other hand, a long distance means more concatenations, it helps to find a "better" optimal concatenation, therefore MDL+OC behaves better when the length increases.



(a) Average runtime w.r.t. distance (b) Runtime w.r.t. query size

Fig. 6. Runtime of different methods

6.2 Efficiency

Figure 6 shows the runtime on different query distance and query size. Since MDL+Dijkstra does not need to find the optimal concatenation, it runs much more faster than MDL+OC. Figure 6(a) shows that the runtime of MDL+OC increases as the query distance rises, since large distance will increase the number of concatenations and the non-trivial popular routes to be checked while routing. However, MDL+OC can return the result very quickly, far less than 1 s. Figure 6(b) illustrates the runtime of MDL+OC grows linearly as the query size increases.

7 Conclusion

In this paper, we propose a framework to find the popular route in consideration of travel cost estimation. We firstly construct a popular traverse graph by discovering the popular routes from historical trajectories, and then use MDL principle to model the travel cost on each popular route. Finally, we devise an efficient routing algorithm to find the popular route with minimal travel cost in terms of optimal concatenation. We evaluate our methods with extensive experiments, showing that our methods are both effective and efficient.

Acknowledgment. Our research is supported by the 973 program of China (No. 2012CB316203), NSFC (61370101, U1401256 and 61402180), Shanghai Knowledge Service Platform Project (No. ZF1213), Innovation Program of Shanghai Municipal Education Commission (14ZZ045), and Natural Science Foundation of Shanghai (No. 14ZR1412600).

References

1. Kanoulas, E., Du, Y., Xia, T., Zhang, D.: Finding fastest paths on A road network with speed patterns. In: Proceedings of ICDE (2006)
2. Ding, B., Yu, J.X., Qin, L.: Finding time-dependent shortest paths over large graphs. In: Proceedings of EDBT, pp. 205–216 (2008)
3. Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., Huang, Y.: T-drive: driving directions based on taxi trajectories. In: Proceedings of SIGSPATIAL (2010)
4. Yuan, J., Zheng, Y., Xie, X., Sun, G.: Driving with knowledge from the physical world. In: Proceedings of KDD (2011)
5. Wang, Y., Zheng, Y., Xue, Y.: Travel time estimation of a path using sparse trajectories. In Proceedings of KDD (2014)
6. Grünwald, P.D., Myung, I.J., Pitt, M.A.: Advances in Minimum Description Length: Theory and Applications. MIT Press, Cambridge (2005)
7. Zheng, K., Zheng, Y., Xie, X., Zhou, X.: Reducing uncertainty of low-sampling-rate trajectories. In: Proceedings of ICDE (2012)
8. Wei, L.-Y., Zheng, Y., Peng, W.-C.: Constructing popular routes from uncertain trajectories. In: Proceedings of KDD (2012)
9. Chen, Z., Shen, H.T., Zhou, X.: Discovering popular routes from trajectories. In: Proceedings of ICDE (2011)
10. Luo, W., Tan, H., Chen, L., Ni, L.M.: Finding time period-based most frequent path in big trajectory data. In: Proceedings of SIGMOD (2013)

11. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische mathematik **1**(1), 269–271 (1959)
12. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
13. Kenneth, K.L., Halsey, E.: The shortest route through a network with time-dependent internodal transit times. J. Math. Anal. Appl. **14**(3), 493–498 (1966)
14. Gonzalez, H., Han, J., Li, X., Myslinska, M., Sondag, J.P.: Adaptive fastest path computation on a road network: a traffic mining approach. In: Proceedings of VLDB (2007)
15. Yang, B., Guo, C., Jensen, C.S., Kaul, M., Shang, S.: Stochastic skyline route planning under time-varying uncertainty. In: Proceedings of ICDE (2014)
16. Mao, J., Song, Q., Jin, C., Zhang, Z., Zhou, A.: Tscluwin: trajectory stream clustering over sliding window. In: Proceedings of DASFAA (2016)
17. Duan, X., Jin, C., Wang, X., Zhou, A., Yue, K.: Real-time personalized taxi-sharing. In: Proceedings of the DASFAA (2016)
18. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of KDD (1996)
19. Ranu, S., Deepak, P., Aditya, D. Telang, A., Deshpande, P., Raghavan, S.: Indexing and matching trajectories under inconsistent sampling rates. In: Proceedings of ICDE (2015)
20. Lee, J.-G., Han, J., Whang, K.-Y.: Trajectory clustering: a partition-and-group framework. In: Proceedings of SIGMOD (2007)
21. Balan, R.K., Nguyen, K.X., Jiang, L.: Real-time trip information service for a large taxi fleet. In: Proceedings of MobiSys (2011)