Frontiers of Computer Science CN 10-1014/TP
DOI 10.1007/s11704-018-7249-z ISSN 2095-2228
RESEARCH ARTICLE

Popular Route Planning with Travel Cost Estimation from Trajectories

Huiping LIU, Cheqing JIN, Aoying ZHOU

School of Data Science and Engineering, School of Computer Science and Software Engineering, East China Normal University, Shanghai 200062, China

Front. Comput. Sci., **Just Accepted Manuscript** • 10.1007/s11704-018-7249-z
http://journal.hep.com.cn on January 30, 2018

Front.Comput.Sci.
DOI

RESEARCH ARTICLE

Popular Route Planning with Travel Cost Estimation from Trajectories

Huiping Liu, Cheqing Jin, Aoying Zhou

School of Data Science and Engineering, School of Computer Science and Software Engineering,
East China Normal University, Shanghai, China, 200062

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, popular route that has been travelled frequently is a good choice, especially for people who are not familiar with the road networks. Moreover, accurate estimation of the travel cost (such as travel time, travel fee and fuel consumption) will benefit a well-scheduled trip plan. In this paper, we address this issue by finding the popular route with travel cost estimation. To this end, we design a system consists of three main components. First, we propose a novel structure, called popular traverse graph where each node is a popular location and each edge is a popular route between locations, to summarize historical trajectories without road network information. Second, we propose a self-adaptive method to model the travel cost on each popular route at different time interval, so that each time interval has a stable travel cost. Finally, based on the graph, given a query consists of source, destination and leaving time, we devise an efficient route planning algorithm which considers optimal route concatenation to search the popular route from source to destination at the leaving time with accurate travel cost estimation. Moreover, we conduct comprehensive experiments and implement our system by a mobile App, the results show that our method is both effective and efficient.

Keywords Location-based services, route planning, travel cost estimation, minimum description length, optimal road concatenation

Received month dd, yyyy; accepted month dd, yyyy

E-mail: *hpliu@stu.ecnu.edu.cn*

1 Introduction

Route planning [2, 16–18, 26–28] is important not only for our daily life, but also for business map engines like Google and Bing Maps. Although the shortest/fastest paths are used commonly in road networks, 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, since travel cost dynamically depends on the traffic conditions and other factors, such as traffic lights, people care more about the travel cost, i.e., how long it takes or how much it costs at the time they are leaving, so that accurate travel cost estimation will improve people's satisfaction.

With the development of intelligent transportation in cities, more and more GPS-equipped vehicles are running on the road networks. As a result, a large number of time-stamped GPS trajectories are consecutively generated. Based on the historical citywide trajectories, popular paths can be constructed by finding out how people usually travel between locations, and with the temporal information in the trajectories, it's possible to estimate the travel cost of paths.

Route planning or driving direction planning has been studied in recent years and some influential works have been published. [2, 21] propose 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 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 inaccurate travel time estimation of the route. [3] proposes a model to estimate the travel time of a given path by using the optimal route concatenation which considers dependencies between road segments. However, it cannot be applied to route planning directly.

To find the popular route with travel cost estimation, three challenges must be addressed.

Data sparseness and coverage Due to the complexity of road network, we cannot guarantee that all roads have been covered by trajectories. Moreover, for each road, it is practically impossible that there are sufficient trajectories during all time intervals. We must thus contend with data sparseness to response users' path queries between any locations at any time intervals. On the other hand, road network is not always available, it's unpractical to find the popular routes/roads by counting the support on each road. Thus, in this occasion, how to get the paths/roads from trajectories is meaningful.

Travel cost modeling Different roads have different travel cost, because road quality and road condition are not the same. For example, the highway with less traffic lights travel faster than the roads with more traffic lights. Meanwhile, the travel cost of the same road also varies at different time. For example, the travel time at rush hour is usually longer than that at the non-rush hour due to traffic jam. Hence, an intelligence modeling for the travel cost on each road at different time that can benefit accurate travel cost estimation is a challenge, especially under the fact of data sparseness.

Path cost estimation The cost of a path can be estimated by summing up, however, the results are only accurate if the travel costs of roads are independent. In practice, the travel costs may be dependent due to intersection or turning. To derive accurate travel cost of path, the dependencies must be considered. Moreover, estimating the travel cost for a given path is not enough to answer a path query, so it's challenging to estimate the travel cost of paths while routing.

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. To deal with the above challenges, we devise a system 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 [4] to model the travel cost for each popular route at different time intervals, so that each time interval has a stable travel cost. Finally, based on the graph, given a source-destination pair and a leaving time, for accurate travel cost estimation, we find the fastest popular route in consideration of the optimal route concatenation [3] which considers the dependencies between road segments. 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 popular locations.
- We present a self-adaptive 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 conduct extensive experiments upon a real dataset of millions of trajectories generated by more than 10,000 taxis over a month in Beijing. The results show that our method is both effective and efficient. Moreover, we implement and visualize our system through a mobile App.

The remainder of the paper is organized as follows. Section 2 reviews the related work. Section 3 describes some preliminary knowledge. Section 4 describes the overview of our framework. Section 5 introduces the popular traverse graph, and the method to model the travel cost for each popular route. Section 6 details the routing algorithm. Section 7 reports the evaluation and a brief conclusion is given in Section 8.

Compared with our earlier proceeding paper [1], this extended paper mainly claims following contributions. First, we define our problem more formally and devise and implement a cloud-mobile based popular route planning system for solving it, where the mobile App is designed. The architecture of our system is introduced in Section 4, and the detailed

interactions of the mobile App is presented in Section 7.4. Second, in Section 6, we improve the efficiency of our routing algorithm by more efficient optimal concatenation computation, which enables faster response to user query. In addition, we discuss the time complexity of our routing algorithm. Third, we conduct more comprehensive experiments to validate our system. For example, we evaluate the popular traverse graph with different parameter settings (Section 7.2) and test the performance of our method on different popular traverse graph (Section 7.3.1). In addition, we evaluate the efficiency of our improved routing algorithm compared to the method in our previous work [1] (Section 7.3.2). Moreover, a case study on the mobile App is illustrated (Section 7.4).

2 Related work

In general, our problem on popular route planning with travel cost estimation relates to two problems in the literature, namely route planning and path cost estimation.

Route planning: Route planning, also referred to as driving direction planning or route finding or path finding, has been studied for decades. [12] proposes a fundamental algorithm (Dijkstra's algorithm) to find the shortest path between two nodes in a graph and the A^* algorithm proposed by Hart [22] boosts the searching performance with heuristic estimation. In [15–17], the authors study how to find the fastest path on a time-dependent graph, where the weight of edge is dynamic. [23] computes the fastest path on a road network by considering speed and driving patterns. All above works find route from source to destination on a given weighted graph. However, this kind of weighted graph, such as the road networks, is not always available. Moreover, the weight of each edge is difficult to determine. Yuan et al. [2, 18] proposed a framework to find the fastest route from taxi trajectories. In [2], they construct a landmark graph and based on that, a two-stage routing algorithm is performed to find the fastest route. In [18], traffic prediction is employed for optimization. However, the estimated travel time is not actually the travel time of the practical fastest route as [2, 18] recommended and dependencies between road segments are not considered.

[6, 14] discover the top-k possible popular routes that sequentially pass given locations from historical uncertain trajectories. [5] 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. [7] 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 travel cost estimation.

Path cost estimation: [2, 18, 24, 25] estimate path cost by summing up the travel costs of the edges in the path. [11] studies stochastic skyline route queries in road networks with multiple travel costs, it provides the travel cost distribution given a source-destination pair with a leaving time. However, the above works ignore road dependencies, such as intersection and turning, which can lead to inaccurate estimation. [3, 20] propose a model to estimate the travel time of a given path in a road network by considering road dependencies, but they cannot be directly applied to route planning from a source to a destination. Our work differs from the above works, because we aim at finding the popular route between locations as well as estimating the travel cost by considering road dependencies.

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 $p_{i+1}.t > p_i.t$ $(1 \leq i < n)$.*

Trajectory 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 that has been travelled frequently is always a candidate for a trip, because most of people have chosen it. However, not all source-destination pairs have a direct popular route (condition 1). For instance, a long path may not

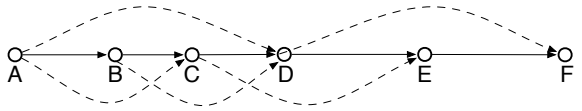


Fig. 1: An example of route concatenation

be frequently passed as a whole. In this case, it can still be treated as a popular route if all its sub-paths are popular routes (condition 2) [5]. To distinguish, we call the popular route that holds condition (1) *trivial popular route*, and the popular route only holds condition (2) *non-trivial popular route*.

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 trivial popular routes between locations.*

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

Definition 4 (Concatenation of Route)**.** *Let* $r : n_1 \rightarrow n_2 \rightarrow ... \rightarrow n_q$ *denotes a route in the popular traverse graph G, where* n_k $(1 \leq k \leq q)$ *is a node in G. Denote* $|r|$ *as the size of r which is the number of nodes it contains. A concatenation of r is* $r^* : p_1||p_2||\cdots||p_k$*, where* $p_i (1 \leq i \leq k)$ *is a trivial popular sub-path of r and all* p_i *together cover r without intersection.*

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

Example 1. *Figure 1 is a path from A to F on PTG. For sub-path* $r : A \rightarrow B \rightarrow C \rightarrow D$*, if r is travelled frequently as a whole, then r is a trivial popular route due to the condition (1) in definition 2, we use dashed lines to represents the trivial popular routes with size greater than 2. Clearly, each sub-path of r is also a trivial popular route, such as* $A \rightarrow B \rightarrow C$ *and* $B \rightarrow C \rightarrow D$*. Then, the concatenations of path* $P : A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F$ *can be* $A \rightarrow B \rightarrow C \rightarrow D||D \rightarrow E \rightarrow F$*,* $A \rightarrow B \rightarrow C||C \rightarrow D \rightarrow E||E \rightarrow F$*,* $A \rightarrow B||B \rightarrow C||C \rightarrow D||D \rightarrow E||E \rightarrow F$ *and so on, there are totally 10 different ways of concatenations for P. Note that only path that contains trivial popular route with size greater than 2 has different ways of concatenations.*

Different ways of concatenations lead to different travel cost estimations when summing up the travel costs of all sub-paths. The next issue is how to find an optimal one from all possible ways of concatenations for a path. [3] finds the optimal concatenation of a path P by making an object function minimized, so that the empirical risk of travel cost estimation will be the minimal. In this paper, we apply the same object function: $\sum_{i=1}^{k} \frac{1}{n_{p_i}} Var(c_{p_i})$, where p_i is the ith sub-path of P, $Var(c_{p_i})$ is the variance of the travel costs on p_i, and n_{p_i} is the number of trajectories that travelled on p_i.

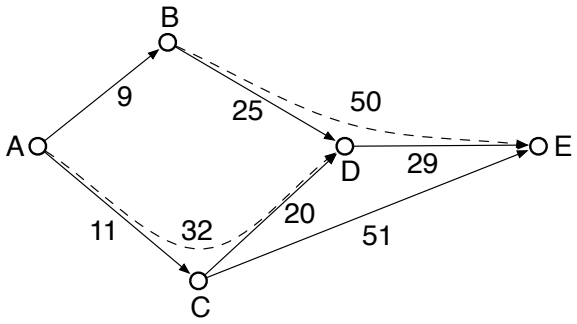


Fig. 2: An example of popular traverse graph

Table 1: Different paths from A to E

Path	Optimal concatenation	Travel cost
$A \rightarrow B \rightarrow D \rightarrow E$	$A \rightarrow B\|\|B \rightarrow D \rightarrow E$	9+50=59
$A \rightarrow C \rightarrow D \rightarrow E$	$A \rightarrow C \rightarrow D\|\|D \rightarrow E$	32+29=61
$A \rightarrow C \rightarrow E$	$A \rightarrow C\|\|C \rightarrow E$	11+51=62

Finally, we formally define the problem addressed in this paper.

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 a popular route from s to d such that the travel cost of the optimal concatenation of this route is minimal at time t.

Example 2. *Figure 2 shows a popular traverse graph with static weight on each edge. Suppose the dashed lines that represent the trivial popular routes with size greater than 2 are also the optimal concatenation, that is the optimal concatenation of* $A \rightarrow C \rightarrow D$ *and* $B \rightarrow D \rightarrow E$ *are* $A \rightarrow C \rightarrow D$ *and* $B \rightarrow D \rightarrow E$ *respectively. Let's find the expected path from A to E. Table 1 lists all the possible paths from A to E and their corresponding optimal concatenations and travel costs. Since path* $A \rightarrow B \rightarrow D \rightarrow E$ *has the optimal concatenation with minimal travel cost, the returned path should be* $A \rightarrow B \rightarrow D \rightarrow E$ *with travel cost 59.*

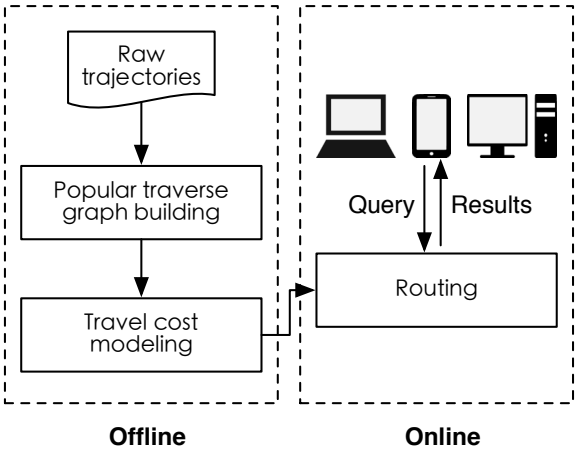


Fig. 3: System overview

4 Overview

Figure 3 illustrates the overview of our system to find the popular route with travel cost estimation from trajectories. Our system is implemented by using client/server model, and it consists of three major components: popular traverse graph building, travel cost modeling, and routing. The first two components operate offline and the third is running online. The offline parts only need to be performed once unless the trajectory archive is updated, and the online part runs on our background server and executes real-time responds to user's instant requests via App at different devices.

Popular traverse graph building. Since the road network is not always available, based on the raw trajectory dataset, we construct a popular traverse graph, where each node is a popular location, and each edge is a trivial popular route between two locations. Accordingly, each path on this graph from one location to another is a popular route.

Travel cost modeling. Since the cost of a route varies a lot at different time, for each popular route (an edge) of popular traverse graph, we need to model its travel cost at different time intervals according to historical trajectories. For trajectories pertaining to a popular route, we try to partition their travel costs into different time intervals, so that each time interval has a stable travel cost. After travel cost partitioning on all popular routes, the popular traverse graph becomes time-dependent.

Routing. Based on popular traverse graph, given a user query consists of source, destination and leaving time, we find the fastest popular route as well as its travel cost from source to destination at the leaving time, in consideration of the optimal route concatenation [3] for accurate travel cost estimation.

5 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 modeling of popular routes.

5.1 Constructing Popular Traverse Graph

As the road network may be unavailable or incomplete in some situations [5,6], 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 (frequently visited) locations 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. Algorithm 1 describes the major processes.

Popular location mining. Since trajectories consist of points, the popular locations can 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 same locations. To find the popular locations, we first partition the points into different zones and then the DBSCAN clustering algorithm [9] is invoked to generate clusters for the points in each zone. Finally, top-k clusters with the maximal number of points are chosen as the popular locations.

Transition 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 sub-segment of tr that belongs to this transition for popular routes discovering.

Popular route 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 sub-trajectories pertaining to each transition. As a result, the cluster with at least τ trajectories is treated as a popular route. To tackle the ununiform rate trajectories, we apply Edit Distance with Projections (EDwP) proposed in [19] to

Algorithm 1: ConstructPTG(T)

Input: Raw trajectory dataset T
Output: Popular traverse graph $G = (V, E)$

//Popular locations mining
$v \leftarrow \emptyset$;
foreach *Trajectory t in T* **do**
  $v \leftarrow v \cup t.endPoints$;
$V \leftarrow$ point clustering for v;
//Transitions extraction
$T' \leftarrow$ map all trajectories in T to V;
$d \leftarrow [< key, valueList >]$; //Dictionary data structure
foreach *Trajectory* $tr : l_1 \rightarrow l_2 \rightarrow ... \rightarrow l_n(l_i \in V(1 \leq i \leq n))$ *in* T' **do**
  $tr' \leftarrow$ sub-trajectory of tr from l_i to $l_{i+1}(1 \leq i < n)$;
  $d.key(l_i, l_{i+1}).valueList.add(tr')$;
//Popular routes discovering
$E \leftarrow \emptyset$;
foreach *key* (l_i, l_j) *in* d **do**
  $E \leftarrow E \cup$ trajectory clustering for $d.key(l_i, l_j).valueList$;
return *Popular traverse graph* $G = (V, E)$;

Algorithm 2: ConstructSuffixTree(T, G)

Input: Raw trajectory dataset T, popular traverse graph G
Output: Suffix tree for trivial popular routes with size greater than 2

$T' \leftarrow$ map all trajectories in T to V;
$S \leftarrow \emptyset$;
foreach *Trajectory* $tr : l_1 \rightarrow l_2 \rightarrow ... \rightarrow l_n(l_i \in V(1 \leq i \leq n))$ *in* T' **do**
  if $(l_i, l_{i+1}) \notin E(1 \leq i < n)$, remove (l_i, l_{i+1}) from tr;
  $S \leftarrow S \cup$ remaining segments in tr;
Suffix tree $ST \leftarrow \emptyset$;
foreach *Segment* $s : l_1 \rightarrow l_2 \rightarrow ... \rightarrow l_n(l_i \in V(1 \leq i \leq n))$ *in* S **do**
  foreach *Suffix route* $s^*(|s^*| > 2)$ *of* s **do**
    if $s^* \notin ST$ **then**
      Create a path s^* on ST and set its support to 1;
    else
      Increment the support of s^* by 1;
Remove all routes with support less than τ from ST;
return ST;

measure the similarity between trajectories, and we use the density-based method [10] 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. In case there are more than 1 popular routes in a transition, we add a flag to distinguish them. The time complexity of Algorithm 1 consists of one point clustering on end points of all trajectories, one trajectory clustering on each transition and one trajectory dataset scanning for transitions extraction. In practice, we use distributed computing platform, i.e., Hadoop, to process massive trajectory dataset (see experiment setting in Section 1).

5.2 Constructing Suffix Tree

Since the trivial popular routes on PTG constructed by Algorithm 1 are all with size equals to 2, to find the optimal concatenation of a popular route, we need to get the trivial popular routes on PTG with size greater than 2. To achieve this goal, we construct a suffix-tree-based index from trajectory dataset to quickly retrieve such trivial popular routes between nodes on PTG. For a popular route $r : n_1 \rightarrow n_2 \rightarrow ... \rightarrow n_k$ on PTG, we define *suffix route* of r as the route $r^* : n_i \rightarrow ... \rightarrow n_k(1 \leq i < k)$. Algorithm 2 shows the details on building suffix-tree-based index for the trivial popular routes with size greater than 2.

For each trajectory, we first map it to a string of nodes on PTG, i.e., t: $l_1 \rightarrow l_2 \rightarrow ... \rightarrow l_n$. For each transition $l_i \rightarrow l_{i+1}(1 \leq i < n)$, if it's not a popular route on PTG, then we know that $l_i \rightarrow l_{i+1}$ won't be a popular route or be a part of popular route, hence it will be removed from t. Now t is split into several segments. For each segment $s : l_i \rightarrow ... \rightarrow l_j$, if $|s| > 2$, it could be a trivial popular route with size greater than 2. Then, for each suffix route s^*: $l_o \rightarrow ... \rightarrow l_j(i \leq o \leq j - 2)$ that its sub-segments are included in the corresponding clusters of $l_k \rightarrow l_{k+1}(o \leq k < j)$, we create a path on the suffix-tree if s^* does not exist 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 trivial popular route. To reduce the size of the suffix-tree, we remove the paths with support less than τ. As a result, given a starting node, we can find all the trivial popular routes with size greater than 2 in $O(1)$ with the reduced suffix-tree.

Suppose the total number of points in the raw trajectory dataset T is $|T|$, the time complexity of Algorithm 2 is $O(|T|)$, since several linear point scanning on T is needed to build the suffix tree.

Example 3. *Figure 4 shows a running example of constructing a reduced suffix-tree based on the PTG and trajectory dataset that the PTG is built from with threshold* $\tau = 2$. *As*

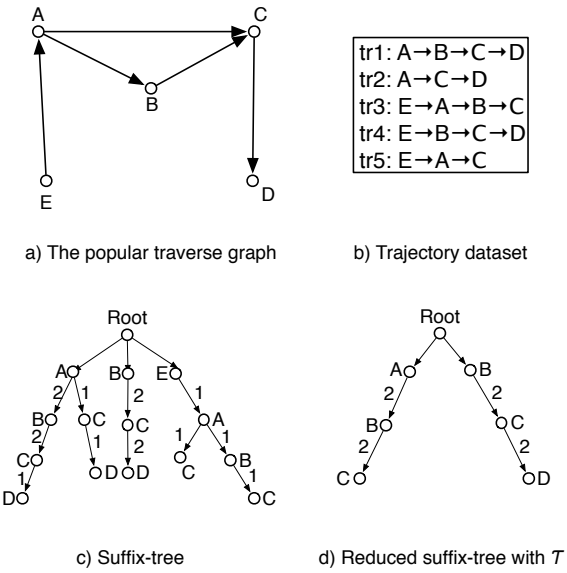


Fig. 4: An example of constructing suffix-tree with $\tau = 2$

we can see, in the PTG, there are totally 2 trivial popular routes with size greater than 2 which are $A \rightarrow B \rightarrow C$ and $B \rightarrow C \rightarrow D$.

5.3 Modeling Travel Cost Using the MDL Principle

The travel cost of a path varies a lot for different drivers at different time [2, 3, 11], for accurate travel cost estimation, we need to find the travel cost pattern on each popular route at different time slots, for example, the rush hour and non-rush hour intervals. However, it's challenging to mine such patterns due to data sparseness and skewness. [8, 11] statically divide a day into different time slots, which is hardly suitable for all routes. The VE-Clustering algorithm [2, 21] 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 due to different distributions. Fortunately, although challenging, the minimum description length (MDL) principle [4, 10] which is widely used in information theory is suitable to solve this problem. Hence, we propose a parameter-free and self-adaptive algorithm by integrating the MDL principle.

Before introducing our method, we note that a good partition for travel costs should meet the following two requirements: (1)*homogeneity*: stable travel costs in each time slot, which can reflect the traffic pattern, and (2) *conciseness*: the significant difference (i.e., distribution of the travel costs) between two adjacent time slots, otherwise, there is no need to split them apart. That is, a good partition should possess both two properties: *homogeneity* and *conciseness*, and the more homogenic and concise, the better. However, *homogeneity* and *conciseness* are contradictory to each other. For example, if the partition has only one time slot, then all travel costs will be in the same slot. In this case, the *conciseness* is maximized and the *homogeneity* is minimized. In contrast, if every single travel cost possesses a time slot, then *homogeneity* is maximized and the *conciseness* is minimized. Therefore, we need to find an optimal trade-off between the two properties. To this end, we introduce the minimum description length (MDL) principle to solve this problem.

Let's review MDL briefly [4]. 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 $MDL = L(H) + L(D|H)$ must be minimized. In our problem, H refers to the time partitions and D corresponds to the set of travel costs on each popular route. Hence, $L(H)$ and $L(D|H)$ are defined formally below.

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

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

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 $slot_i$. The second term computes the information entropy of the travel costs in time slot $slot_i$ to describe the stability of the slot, where $N(slot_i)$ is the total number of travel costs in $slot_i$ and $N_k(slot_i)$ is the total number of travel costs in the kth class in $slot_i$. That is, we need to first map the travel costs to different classes, and each class represents a different cost level. For example, in terms of travel time, 5 minutes (or less) may be a cost level, i.e., costs from 0 minute to 5 minutes correspond to class 1 and 5 minutes to 10 minutes correspond to class 2 and so on.

Obviously, $L(H)$ measures the degree of *conciseness*, a maximum *conciseness* leads to a minimum $L(H)$. And $L(D|H)$ measures the degree of *homogeneity*, a maximum *homogeneity* leads to a minimum $L(D|H)$. Therefore, finding an optimal tradeoff between *homogeneity* and *conciseness* is to find a partition that minimizes $L(H) + L(D|H)$. However,

Algorithm 3: ApproPartition(D)

Input: Travel cost set D on a popular route
Output: Approximate optimal partition set S

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 $\quad s[t_1, t_2] \leftarrow$ a slot in S with the maximal value of $\mathsf{Ent}(\cdot)$;
5 $\quad t \leftarrow \arg\min_{t \in [s.t_1, s.t_2]}(\mathsf{Ent}([s.t_1, t]) + \mathsf{Ent}([t, s.t_2]))$;
6 $\quad S' \leftarrow (S - \{s\}) \cup \{[s.t_1, t], [t, s.t_2]\}$;
7 $\quad newCost \leftarrow MDL(S')$;
8 $\quad$ **if** $newCost < minCost$ **then**
9 $\quad\quad minCost \leftarrow newCost$;
10 $\quad\quad S \leftarrow S'$;
11 $\quad$ **else**
12 $\quad\quad$ **return** S;

computing the theoretically optimal partitioning is expensive due to too many potential partitionings. Thus, we devise an approximate solution instead, as shown in Algorithm 3. This algorithm accepts a travel cost set D on a popular route as input. Initially, a partition 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 iteration, we find a time slot 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 sub-slots $[s.t_1, t]$ and $[t, s.t_2]$. The above iterative procedure won't stop until $newCost \geq minCost$, then we return the partition set S as the approximate optimal result.

The time complexity of Algorithm 3 is $O(n \cdot k)$, where k is the number of time slots and n is the size of the dataset D, because it computes the MDL cost by scanning the whole dataset at each iteration. For simplicity, we use the expected travel cost in each time slot to represent the cost of it. After we execute Algorithm 3 on every popular route in PTG, we will get a time-dependent popular traverse graph where the travel cost on each trivial popular route is dynamic, which means that each trivial popular route in the graph has a travel cost function $f(t) : t \rightarrow R_{\geq 0}$, where t is the leaving time from this popular route.

Example 4. *Table 2 shows an example of the travel cost partitions of all trivial popular routes on the popular traverse graph illustrated by Figure 2. Each partition in this table consists of time slot and the travel cost in that slot, and the travel cost is represented by the expected value, variance and number of all travel costs in the slot. For example, popular route $A \rightarrow B$ contains 4 partitions after travel cost modeling. The first partition is from 21pm to 7am, whose expected travel cost and variance are 7 and 0.8 respectively, and the number of travel costs is 50, which means that there are 50 different trajectories have visited popular route $A \rightarrow B$. If the leaving time from A to B is 10am, then the expected travel cost will be 9. Since popular route $A \rightarrow C \rightarrow D$ has two different concatenations which are $A \rightarrow C||C \rightarrow D$ and $A \rightarrow C \rightarrow D$, the value of aforementioned object function of concatenation $A \rightarrow C||C \rightarrow D$ is 1/150*0.8+1/180*0.85=0.01, and the corresponding value of concatenation $A \rightarrow C \rightarrow D$ is 1/130*0.95=0.007 < 0.01. Hence, the optimal concatenation of popular route $A \rightarrow C \rightarrow D$ will be $A \rightarrow C \rightarrow D$. That is, the expected travel cost of $A \rightarrow C \rightarrow D$ is 32 whenever you leave.*

Table 2: An example of travel cost partitions of different trivial popular routes on Figure 2

Trivial popular route	Travel cost partition
$A \rightarrow B$	[21,7):(7,0.8,50) [7,9):(10,0.9,90) [9,16):(9,0.85,80) [16,21):(12,1.0,100)
$A \rightarrow C$	[0,24):(11,0.8,150)
$B \rightarrow D$	[22,7):(22,0.6,50) [7,22):(28,0.7,150)
$C \rightarrow D$	[0,24):(20,0.85,180)
$C \rightarrow E$	[21,8):(45,0.65,30) [8,21):(55,0.8,160)
$D \rightarrow E$	[0,24):(29,1.1,190)
$A \rightarrow C \rightarrow D$	[0,24):(32,0.95,130)
$B \rightarrow D \rightarrow E$	[21,6):(45,0.65,40) [6,21):(55,0.95,120)

6 Route Planning On the Popular Traverse Graph

In this section, given the time-dependent popular traverse graph, we introduce how to find the popular route with the minimal travel cost in consideration of the optimal route concatenation. We note that road network is a FIFO graph [16, 29], hence no waiting time is needed in the popular traverse graph when we do a route planning.

Algorithm 4: RoutePlanning ($G = (V, E), s, d, t$)

Input: Popular traverse graph G, source s, destination d and leaving time t
Output: The popular route from s to d such that the travel cost of its optimal concatenation is minimal at leaving time t

1 $s.cost \leftarrow 0$, $s.route \leftarrow s$, $s.routeSet \leftarrow \emptyset$, $s.settled \leftarrow false$;
2 Create a priority query Q;
3 $Q.enqueue(s)$;
4 **while** $Q \neq \emptyset$ **do**
5 $\quad n \leftarrow Q.dequeue()$;
6 $\quad n.settled \leftarrow true$;
7 $\quad$ **if** $n = d$ **then**
8 $\quad\quad$ **return** $d.route$;
9 $\quad S \leftarrow \{r : n \rightarrow ... \rightarrow v | v$ is not settled and r is a trivial popular route$\}$;
10 $\quad$ **foreach** *route* $r : n \rightarrow ... \rightarrow v$ *in* S *sorted by* $|r|$ *in ascending order* **do**
11 $\quad\quad r' \leftarrow n.route + r$;
12 $\quad\quad$ **if** r' *is not in* $v.routeSet$ **then**
13 $\quad\quad\quad$ //**Compute the optimal concatenation**
14 $\quad\quad\quad obj, cost \leftarrow$ **ComputeOC**(r', t);
15 $\quad\quad\quad v.routeSet \leftarrow v.routeSet \cup \{< r', obj, cost >\}$;
16 $\quad\quad\quad$ **if** $cost < v.cost$ **then**
17 $\quad\quad\quad\quad v.cost \leftarrow cost, v.route \leftarrow r'$;
18 $\quad\quad\quad\quad$ **if** v *is not in* Q **then**
19 $\quad\quad\quad\quad\quad Q.enqueue(v)$;

6.1 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) its 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.

A naive solution is to find all possible routes from source to destination, then for each possible route, we compute its optimal concatenation and the corresponding travel cost at the leaving time. Finally, the expected route will be the one with minimal travel cost. However, it's prohibitive to enumerate all the possible routes and compute their optimal concatenations. Instead, we propose a method to return the expected route more efficiently, as listed in Algorithm 4.

Given a PTG $G = (V, E)$ and the query $q = (s, d, t)$, Algorithm 4 returns the expected route. For each processing node n, we use attributes *cost* and *route* to keep the minimal cost and the optimal route from source s to n respectively. In addition, we use a set $routeSet$ whose element is a triple tuple $< route, obj, cost >$ with key *route* to record the route has been checked from s to n and its corresponding minimal object function value obj and optimal cost *cost* with minimal estimated error. Moreover, attribute *settled* indicates whether n has been settled or not during extending. In the process, we maintain a priority queue Q for nodes on PTG sorted by *cost* in ascending order. Initially we set all nodes' *cost* to ∞, *route* to *null*, $routeSet$ to *empty* and *settled* to $false$ except for the source node s whose initial *cost* equals to 0. At the beginning, s is added to Q (lines 1-3). Then we keep extending nodes until Q is empty or the destination node d is settled (lines 4-19). At each iteration, we get the head node n of Q and find its outgoing popular route set where each route is a trivial popular route (line 9). For each trivial popular route $r : n \rightarrow ... \rightarrow v$ ordered by $|r|$, we consider the entire route r' from s to v via n by concatenating $n.route$ with r. If r' has not been checked, we then find its optimal concatenation with minimal object function value obj and the corresponding travel cost *cost*. We then insert the triple tuple $< r', obj, cost >$ into $v.routeSet$ to avoid verifying r' twice (lines 13-15). Subsequently, if the current cost of v is less than its old cost, then we update v' cost and its route, and add v to Q if v is not in Q (lines 16-19). In this way, we get the route whose optimal concatenation has minimal travel cost according to Lemma 1.

Lemma 1. *Algorithm 4 returns the optimal concatenation route with minimal travel cost on popular traverse graph* $G = (V, E)$ *for a query* $q = (s, d, t)$ *if the route exists.*

Proof. It is obvious that only the cost of optimal concatenations can be considered by Algorithm 4, then we need to prove that the route returned by Algorithm 4 has minimal travel cost. We prove it by contradiction. Suppose there exists a route r' whose optimal concatenation is $s \rightarrow ... \rightarrow n_1 || n_1 \rightarrow ... \rightarrow n_2 || ... || n_i \rightarrow ... \rightarrow d$ that has less travel cost than the route r^* returned by Algorithm 4. Then each sub-route of r', say $r'' : s \rightarrow ... \rightarrow n_j (1 \leq j \leq i)$, has optimal concatenation $s \rightarrow ... \rightarrow n_1 || ... || n_{j-1} \rightarrow ... \rightarrow n_j$ and less travel cost than r^*. As a result, r' will be returned before r^*, which is a contradiction. Thus the lemma is proved. □

Example 5. *Let's take Figure 2 as the input PTG, suppose the dashed line stands for the optimal concatenation. We now find the expected route from A to E by using Algorithm 4. Starting from A, we check* $A \rightarrow B$ *and* $A \rightarrow C$ *whose optimal concatenation is themselves and* $A \rightarrow C \rightarrow D$ *with optimal*

concatenation $A \rightarrow C \rightarrow D$. Then the cost of B, C and D will be 9, 11 *and* 32 *respectively. We next settle B and check its outgoing trivial popular routes. Since the cost of $A \rightarrow B||B \rightarrow D$ is 34 which is larger than the cost of $A \rightarrow C \rightarrow D$, the cost of D won't be updated. Subsequently, the cost of E will be* 59 *due to optimal concatenation $A \rightarrow B||B \rightarrow D \rightarrow E$. Then we settle C and check routes $C \rightarrow D$ and $C \rightarrow E$, since $A \rightarrow C \rightarrow D$ has been checked (exists in the routeSet of D), nothing needs to be done. For $A \rightarrow C \rightarrow E$ whose cost is 62, the cost of E won't be updated. We proceed to settle D and consider $A \rightarrow C \rightarrow D||D \rightarrow E$ with cost* 61. *Finally, we settle E and get the expected route $A \rightarrow B \rightarrow D \rightarrow E$ with cost* 59.

It's worth noting that we find the optimal concatenation of r' by using the entire route, not simply combine the optimal concatenation of *n.route* and r. Consider an example of Figure 1, suppose we go from A to E and we now at node C, that is the optimal concatenation from A to C has been found, say $A \rightarrow B||B \rightarrow C$. If we combine $A \rightarrow B||B \rightarrow C$ and the optimal concatenation from C to E (say $C \rightarrow D \rightarrow E$) and regard it ($A \rightarrow B||B \rightarrow C||C \rightarrow D \rightarrow E$) as the optimal concatenation of the route from A to E, then the concatenations $A \rightarrow B \rightarrow C \rightarrow D||D \rightarrow E$ and $A \rightarrow B||B \rightarrow C \rightarrow D||D \rightarrow E$ will be missed, and they may have less object function value than $A \rightarrow B||B \rightarrow C||C \rightarrow D \rightarrow E$. Hence, we need to find the optimal concatenation from A to E by using the whole route and considering all possible different concatenations.

Although Algorithm 4 returns the right route, the bottleneck is computing the optimal concatenation at line 14. To efficiently find the expected route by Algorithm 4, we devise an efficient method to compute the optimal concatenation for a given route.

6.2 Computing the optimal concatenation

As mentioned above, finding the optimal concatenation of a route $r : n_1 \rightarrow n_2 \rightarrow \cdots \rightarrow n_k$ is time-consuming due to too many different concatenations ($O(2^{k-2})$). [1] proposes a method by using a dominance relation between different concatenations, and thus only the non-dominated concatenations are considered to reduce the searching space, however, the number of non-dominated concatenations may be still large for a long path and the computations of the object function value of a path may be redundant when it has many different non-dominated concatenations. To overcome above weaknesses, we propose a new and more efficient method by reusing the pre-computed optimal concatenation results. Recall that the optimal concatenation of $n_1 \rightarrow \cdots \rightarrow n_i (1 < i < k)$ has been computed, because we first compute the path with smaller size (line 10 in Algorithm 4). Accordingly, the optimal concatenation of r can be computed by using the state transition function shown by Equation 3, where $n_i.minObj (1 \leq i < k)$ is the minimal object function value of route $n_1 \rightarrow ... \rightarrow n_i$ that has been known and can be found in $n_i.routeSet$, $n_i \rightarrow ... \rightarrow n_k$ is the trivial popular route from n_i to n_k and obj is the object function value that has been computed in Section 5.3. Finally, the object function value of r's optimal concatenation is $n_k.minObj$, and its cost can then be computed.

$$n_k.minObj = \min_{1 \leq j < k}\{n_j.minObj + (n_j \rightarrow ... \rightarrow n_k).obj\} \quad (3)$$

Algorithm 5 shows the details of our method. Given a path r and leaving time t, Algorithm 5 returns the optimal concatenation of r at leaving time t. Initially, we set r's minimal objective function value $n_k.minObj = \infty$ and its corresponding travel cost $n_k.optCost = \infty$. Since n_1 is the source node, we insert triple tuple $< n_1, 0, 0 >$ into its *routeSet*. Subsequently, for each node n_i in r, we first check if $n_i \rightarrow \cdots \rightarrow n_k$ is a trivial popular route, if so, that means there is a possible concatenation $n_1 \rightarrow \cdots n_i || n_i \rightarrow \cdots \rightarrow n_k$, we compute its objective function value by summing up the minimal objective function value of $n_1 \rightarrow \cdots \rightarrow n_i$ and the objective function value of $n_i \rightarrow \cdots \rightarrow n_k$, which already have been computed. Then if the objective function value of the possible concatenation is less than $n_k.minObj$, we have found a better concatenation and update $n_k.minObj$ and $n_k.optCost$. Finally, we get r's optimal concatenation with minimal objective function value and its corresponding cost. The complexity of Algorithm 5 is $O(k)$ due to a linear scan.

Example 6. *Let's reconsider the popular traverse graph in Figure 2, suppose the travel cost partitions of trivial popular routes are shown by Table 2 and we find the expected route from A to E at 8:00am. According to Algorithm 4, we first compute the optimal concatenations for $A \rightarrow B$ and $A \rightarrow C$, and add $< A \rightarrow B, 1/90 * 0.9 = 0.01, 10 >$, $< A \rightarrow C, 1/150 * 0.8 = 0.005, 11 >$ to the routeSet of B and C, respectively. Then we consider $A \rightarrow C \rightarrow D$ by using Algorithm 5, the first possible concatenation is $A \rightarrow C \rightarrow D$ with $obj = 1/130 * 0.95 = 0.007$ and $cost = 32$ at leaving time 8:00am. The next possible concatenation is $A \rightarrow C||C \rightarrow D$, and the minObj and optCost of $A \rightarrow C$ is 0.005 and 11, respectively, which is kept in C.routeSet. Since the obj and cost of $C \rightarrow D$ at leaving time 8:11am is $1/180 * 0.85 = 0.005$ and 20, the obj and cost of concatenation $A \rightarrow C||C \rightarrow D$ will be $0.005 + 0.005 = 0.01 > 0.007$. Hence, the optimal concatenation of $A \rightarrow C \rightarrow D$ is $A \rightarrow C \rightarrow D$, and it will be kept in D.routeSet. In this way, the expected route can be computed.*

Algorithm 5: ComputeOC(r, t)

Input: Popular route $r : n_1 \rightarrow \cdots \rightarrow n_k$, leaving time t
Output: The object function value and travel cost of r's optimal concatenation

$n_k.minObj \leftarrow \infty, n_k.optCost \leftarrow \infty$;
$n_1.routeSet \leftarrow \{< n_1, 0, 0 >\}$;
foreach $i = 1...k-1$ **do**
  $r' \leftarrow n_i \rightarrow \cdots \rightarrow n_k$;
  if r' *is a trivial popular route* **then**
    $r'' \leftarrow n_1 \rightarrow \cdots \rightarrow n_i$;
    $< r'', minObj, optCost >\leftarrow$ get the triple tuple of r'' in $n_i.routeSet$;
    $obj \leftarrow r'.obj$ at time $(optCost + t)$;
    $curObj \leftarrow minObj + obj$;
    if $curObj < n_k.minObj$ **then**
      $n_k.minObj \leftarrow curObj$;
      $cost \leftarrow r'.cost$ at time $(optCost + t)$;
      $n_k.optCost \leftarrow optCost + cost$;
return $n_k.minObj$ *and* $n_k.optCost$;

Lemma 2 shows the time complexity of Algorithm 4 by calling Algorithm 5.

Lemma 2. *The time complexity of Algorithm 4 is* $O(mn + n \log n)$*, where* $m = |E|$ *is the number of trivial popular routes and* $n = |V|$ *is number of nodes in the popular traverse graph* $G = (E, V)$.

Proof. Recall that we use a priority queue to manage nodes during processing, the complexity of the queue's operations is $O(\log n)$ if we use Fibonacci heap [13] to implement the priority queue. Since each node is settled at most once, the complexity of operating nodes in the priority queue is $O(n \log n)$. In addition, each trivial popular route will be visited at most once while extending and Algorithm 5 will be invoked, hence the complexity of this part is $O(mn)$. As a result, the time complexity of Algorithm 4 is $O(mn + n \log n)$. □

7 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 2.00GHz CPU and 16GB main memory.

7.1 Settings

Dataset We use a real-life dataset generated by 13,007 taxicabs in Beijing from Oct. 1 to Oct. 31, 2013. The sampling rate is about 1 point per minute. The travel time cost of a trajectory can be computed as the difference between the timestamps of the endpoints (end.timestamp-start.timestamp).

We divide the dataset into two parts, one for training and the other for evaluation. The first part (dataset of the first 21 days) is used to construct a popular traverse graph. It contains 68,686,579 individual trajectories. By removing invalid trajectories (such as crossing forbidden area, too high or too low average speed, too short distance), we finally get 7,122,320 paid meaningful trajectories with distance greater than 3km. Figure 5 shows the statistics of the trajectories. Clearly, the number of trajectory decreases as the length increases, which shows that it's meaningful to estimate the travel cost of path, especially for long path with less support. Since our dataset is very large and building the popular traverse graph is time-consuming, we run it on a 8-node HP ProLiant DL360 distributed cluster. Each node contains 8 Intel Xeon E5606 2.13GHz quad-core processors, 32GB physical memory and 1TB disk, and installs Hadoop 1.0.3 on a CentOS operation system.The second part (dataset of 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.

Baseline The prior work [2,21] proposes a framework, called T-drive, to find the fastest route between two locations at a departure time. It first builds a landmark graph based on the road network, then VE-Clustering is used to partition the travel cost on each edge in the graph, finally a Dijkstra-like routing method is applied on the landmark graph to find the fastest route from the source to the destination at the leaving time and the travel cost is estimated by summing up the travel costs of all edges in the route. In this paper, we use T-drive on popular traverse graph as the baseline method. Moreover, for comparison purpose, we use MDL+Dijkstra to refer to our basic method that applies MDL to model the travel cost and the Dijkstra algorithm to search route on TPG, and MDL+OC (Algorithm 4) refers to our method that applies both MDL and the optimal route concatenation.

7.2 Evaluating popular traverse graph

By setting $\tau = 100$, Figure 6 visualizes the popular traverse graph with different k (number of popular locations), where black points represent popular locations and green lines represent popular routes (In each trajectory cluster, we use the representative trajectory whose total distance between other trajectories is minimized, to represent the popular route when

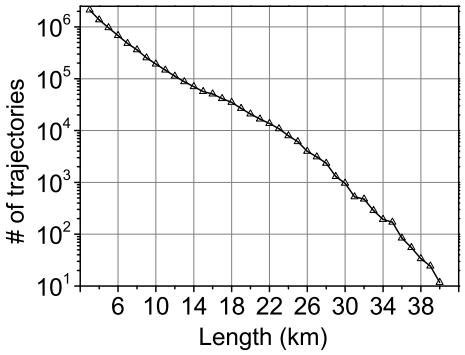


Fig. 5: Statistics of training trajectories

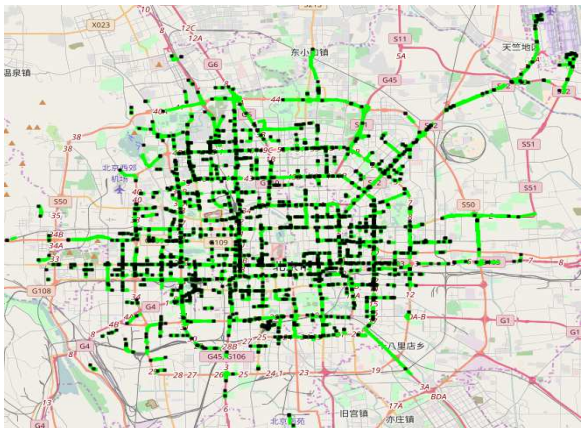
Table 3: The graph size on different τ ($k = 5,000$)

τ	$\|E\|$
10	3,006,271
50	665,287
100	243,695
150	165,587

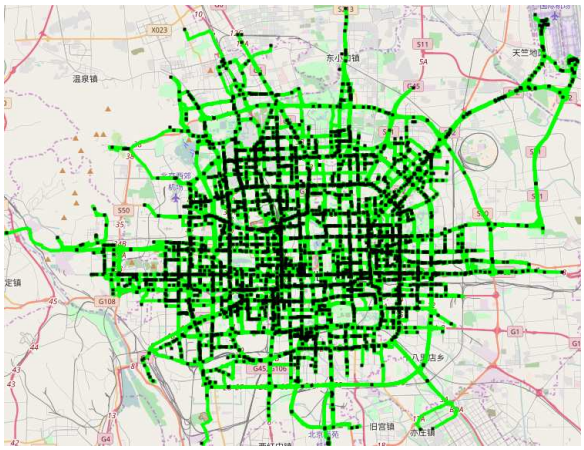
visualizing). Generally, the graphs we build match the road networks of Beijing city and the graph with $k = 5,000$ well covers Beijing city compared to the graph with $k = 2,000$, and its distribution follows our commonsense knowledge. Table 3 shows the size of popular traverse graph on different τ with $k = 5,000$. As τ increases, the number of trivial popular routes on the graph sharply decreases, because more and more trajectories are needed to form a trivial popular route. In the following experiments, we use the graph with $k = 5,000$ and $\tau = 100$ as the default popular traverse graph unless stated otherwise. The default popular traverse graph contains 243,695 different trivial popular routes, in which 210,338 trivial popular routes have size greater than 2, they account for the overwhelming majority of the trivial popular routes. That is, the dependencies between roads generally exist. Figure 7 shows the distribution of the length and support of popular routes on default graph. As we can see, the number of popular routes decreases as length and support rise, and the length of trivial popular route can be as long as 40km, which shows the long dependencies between roads.

7.3 Evaluating routing algorithm

We show the performance of our routing algorithm according to its effectiveness and efficiency.



(a) $k = 2,000$



(b) $k = 5,000$

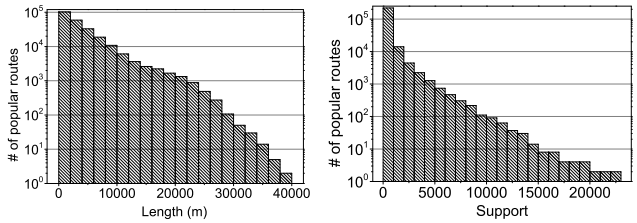
Fig. 6: Visualized popular traverse graph ($\tau = 100$)

7.3.1 Effectiveness

We use mean absolute error (MAE) and mean relative error (MRE) to measure the effectiveness of our method. Equation (4) describes the common definitions, where y_i is an estimate, $\widehat{y_i}$ is the ground truth and n is the number of samples. In our experiments, each sample is a query mentioned above.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \widehat{y_i}|}{n}, MRE = \frac{\sum_{i=1}^{n} |y_i - \widehat{y_i}|}{\sum_{i=1}^{n} \widehat{y_i}} \quad (4)$$

Effectiveness of MDL We ran MDL+Dijkstra and T-drive on the popular traverse graph to investigate the effectiveness of MDL and VE-Clustering [2]. Recall that T-drive needs two parameters, one for V-Clustering and the other for E-Clustering, we try different parameter combinations to reflect its effectiveness. Table 4 compares MAE and MRE between MDL+Dijkstra and T-drive based on aforementioned 30,000 queries. We consider 12 different parameter settings for T-drive. Clearly, different parameter settings lead to different



(a) Length distribution (b) Support distribution

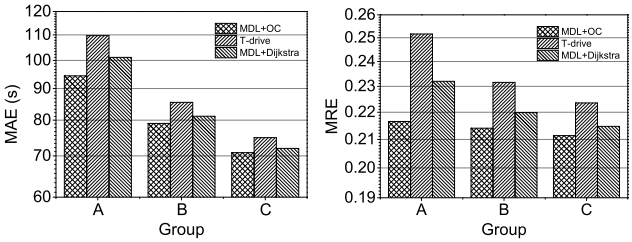
Fig. 7: Distribution of the popular routes on PTG

results and in the best case, MAE = 74.9 seconds and MRE = 0.224. Note that MDL+Dijkstra has MAE of 71.8 seconds and MRE of 0.214. Obviously, MDL+Dijkstra outperforms T-drive. Since the MDL method we proposed find an approximate optimal trade-off between homogeneity and conciseness on every popular route, which guarantees a better result on each popular route. While the global parameters of T-drive cannot generate good partitions for all popular routes due to different travel cost distributions of different popular routes. In the following experiments, we use the best parameter setting which is (50,30) for T-drive.

Overall Effectiveness We sort the queries mentioned above by their lengths in descending order and then choose the top-k queries to generate three original groups of queries with k= 10,000, 20,000, 30,000 respectively. Table 5 shows the statistics of each group. We can see that group A with the top 10,000 queries has maximum average travel time and maximum average length which are 435 seconds and 6,226 meters, respectively. Figure 8 illustrates the performance of three methods for these groups. Clearly, both MAE and MRE of MDL+OC and MDL+Dijkstra are smaller than T-drive on all groups, which means that MDL+OC and MDL+Dijkstra behave better than T-drive, because MDL principle is more suitable for travel cost modeling. For example, the MAEs of MDL+OC, MDL+Dijkstra and T-drive on all 30,000 queries are 70.86, 71.99, 74.94 seconds respectively, and the MREs are 0.211, 0.214, and 0.224. Moreover, MDL+OC outperforms MDL+Dijkstra due to the dependencies between roads are considered in MDL+OC. We note that the performance gap between MDL+OC and MDL+Dijkstra, T-drive rises as the query distance increases. For example, on group C, the MRE differences between MDL+OC and MDL+Dijkstra, T-drive are 0.003 and 0.013 respectively, while on group A, their differences become 0.016 and 0.036. The reason is that MDL+OC considers the optimal concatenation while routing and the longer the distance is, the more concatenations of the route contains, as a result, the more optimal route concatenation will be found, which leads to more accurate travel cost 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.

Table 5: Statistics of three original 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



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

Fig. 8: Comparison of different methods on original query groups

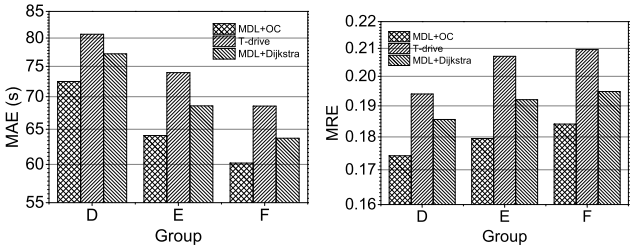
An interesting finding is that around 60-70% queries follow the same route as we found in each group. Hence, we refine each group by keeping the queries that follow the same route as we returned and discarding the others. As a result, we generate three new groups. Table 6 shows the statistics of these refined groups. Clearly, most queries follow the route we recommended. For example, 20,930 out of 30,000 queries (nearly 70%) follow the same popular routes as we returned. That is to say, people prefer the popular routes when driving, and the shorter the distance is, the higher probability the query follow the same route as we found. Figure 9 shows the performance of three methods on the refined groups. Since the path of the query follows the same route as we found, both MAE and MRE decrease comparing to Figure 8. 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 9(a) illustrates the MAE of different methods. As the length decreases, the MAE decreases, too. Conversely, in Figure 9(b), the MREs of all methods increase as the length of query decreases. This is because the shorter a path is, the more unstable its travel time could be due to traffic conditions, such as turnings and traffic lights. On the other hand, a long distance means more con-

Table 4: Comparison of MDL+Dijkstra and T-drive

Methods	MDL+Dijkstra	T-drive											
		30, 10	30, 30	30, 50	30, 100	50, 10	50, 30	50, 50	50, 100	100, 10	100, 30	100, 50	100, 100
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	**0.214**	0.225	0.224	0.227	0.232	0.225	**0.224**	0.227	0.232	0.226	0.224	0.227	0.233

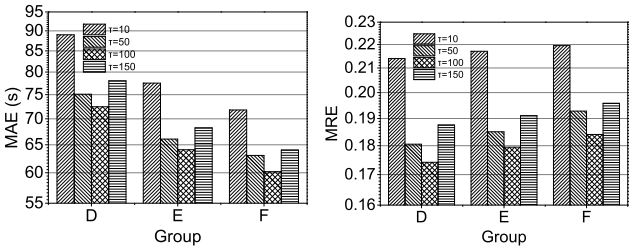
Table 6: Statistics of refined query groups

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%



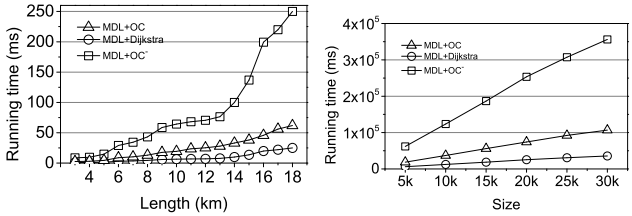
(a) MAE of different methods (b) MRE of different methods

Fig. 9: Comparison of different methods on refined query groups



(a) MAE w.r.t. τ (b) MRE w.r.t. τ

Fig. 10: Performance of MDL+OC with different τ



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

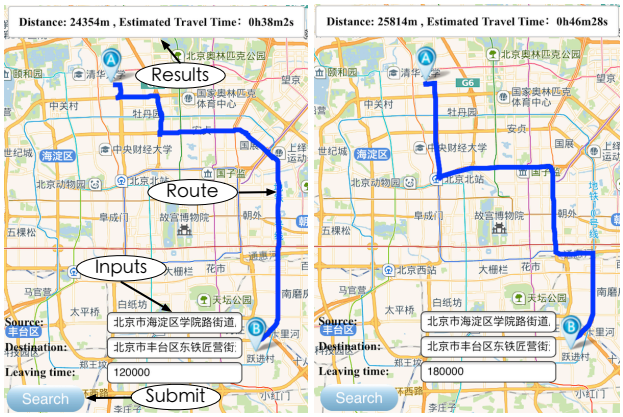
Fig. 11: Runtime of different methods

catenations, it helps to find a “better" optimal concatenation, therefore MDL+OC behaves better when the length increases.

With $k = 5,000$, we test the performance of MDL+OC on different popular traverse graph by varying τ. Figure 10 illustrates the results with different τ. Obviously, both MAE and MRE of MDL+OC are the greatest when $\tau = 10$, this is because the trivial popular routes are with fewer supports, and there are not sufficient data to partition the travel cost, which increases the error of travel cost estimation. As τ increases, both MAE and MRE of MDL+OC get smaller, because the popular routes with less support are filtered. However, when τ becomes too large, the performance of MDL+OC deteriorates. Since less popular routes can be found in this case, accordingly, the dependencies between roads decrease, especially for long distance dependencies, which results in the degradation of performance.

7.3.2 Efficiency

We test the efficiency of our routing algorithm. Figure 11 shows the runtime of MDL+OC, MDL+OC$^-$ and MDL+Dijkstra on different query distance and query size, where MDL+OC$^-$ is the routing algorithm proposed in our previous work [1], and it computes the optimal concatenation by considering all possible non-dominated concatenations. Since MDL+Dijkstra does not need to consider the optimal concatenation, it runs much faster than MDL+OC and MDL+OC$^-$. Moreover, MDL+OC outperforms MDL+OC$^-$, because MDL+OC reuses the pre-computed optimal concatenation results of other nodes, it does not need to consider many different concatenations compared to MDL+OC$^-$. Figure 11(a) shows that the runtime of MDL+OC increases as the query distance rises, since long distance leads to more concatenations to be checked while routing. However, MDL+OC can return the result very quickly, far less than 1 second. For example, when the query length is 18km, MDL+OC finds the route in 62ms on average. We then test the scalability of our algorithm by randomly selected different size of queries, Figure 11(b) illustrates that the runtime of MDL+OC linearly grows as the query size increases, which shows a good scalability of our method.



(a) Non-rush hour (b) Rush hour

Fig. 12: A case study demonstration

7.4 A case study

We implement our system by a mobile App. Figure 12 shows the interface of our App on the client. The interface only contains three inputs, including source and destination locations and the leaving time. Once users submit queries, they will receive the detailed popular route from source (location A) to destination (location B) at the leaving time displayed on the map, moreover, the corresponding travel time and distance of the route will be given. We test the results between the same source-destination pair at different time, as shown in Figure 12. Clearly, the travel time at the rush hour is longer than that at the non-rush hour and two different routes are given at this case, which follows our common sense and shows the effectiveness of our system.

8 Conclusion

In this paper, we formally define our popular route planning problem and propose a cloud-mobile based framework to find the popular route in consideration of travel cost estimation. To this end, we firstly construct a popular traverse graph from historical trajectories, where each node is a popular location and each edge is a popular route, and then we use MDL principle to model the travel cost on each popular route. Subsequently, given a query that consists of source ,destination and leaving time, we devise a more efficient routing algorithm to find the popular route with minimal travel cost in terms of optimal concatenation. Finally, we evaluate our method with extensive experiments and a mobile App is presented, showing that our method is both effective and efficient.

As for future work, it is of interest to use the real-time traffic condition to plan a route for user query, and personalized route planning by considering user profile is also an interesting and challenging work.

9 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. Liu H., Jin C., Zhou A.: Popular Route Planning with Travel Cost Estimation. In: DASFAA, pp. 403-418 (2016)
2. Yuan J., Zheng Y., Zhang C., et al: T-drive: driving directions based on taxi trajectories. In: SIGSPATIAL, pp. 99-108 (2010)
3. Wang Y.,Zheng Y., Xue Y.: Travel time estimation of a path using sparse trajectories. In: KDD, pp. 25-34 (2014)
4. Grnwald P. D., Myung I. J., Pitt M. A.: Advances in Minimum Description Length: Theory and Applications. The MIT Press, (2005)
5. Chen Z., Shen H. T., Zhou X.: Discovering popular routes from trajectories. In: ICDE, pp. 900-911 (2011)
6. Wei L. Y., Zheng Y., Peng W. C.: Constructing popular routes from uncertain trajectories. In: KDD, pp. 195-203 (2012)
7. Luo W., Tan H., Chen L., Ni L. M.: Finding time period-based most frequent path in big trajectory data. In: SIGMOD, pp. 713-724 (2013)
8. Balan R. K., Nguyen K. X., Jiang L.: Real-time trip information service for a large taxi fleet. In: MobiSys, pp. 99-112 (2011)
9. Ester M., Kriegel H. P., Sander J., Xu X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: KDD, pp. 226-231 (1996)
10. Lee J. G., Han J., Whang K. Y.: Trajectory clustering: a partition-and-group framework. In: SIGMOD, pp. 593-604 (2007)
11. Yang B., Guo C., Jensen C. S., Kaul M., Shang S.: Stochastic skyline route planning under time-varying uncertainty. In: ICDE, pp. 136-147 (2014)
12. Dijkstra E. W.: A note on two problems in connexion with graphs. Numerische mathematik, 2(1), 269-271 (2013)
13. Fredman M. L., Tarjan R. E.: Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. Annual Symposium on Foundations of Computer Science, 338-346 (1984)
14. Zheng K., Zheng Y., Xie X., Zhou X.: Reducing Uncertainty of Low-Sampling-Rate Trajectories. In: ICDE, pp. 1144-1155 (2012)
15. Cooke K. L., Halsey E.: The shortest route through a network with time-dependent internodal transit times. Journal of mathematical analysis and applications, 14(3), 493-498 (1966)

16. Kanoulas E., Du Y., Xia T., Zhang D.: Finding Fastest Paths on A Road Network with Speed Patterns. In: ICDE, pp. 10-19 (2006)
17. Ding B., Yu J. X., Qin L.: Finding time-dependent shortest paths over large graphs. In: EDBT, pp. 205-216 (2008)
18. Yuan J., Zheng Y., Xie X., Sun G.: Driving with knowledge from the physical world. In: KDD, pp. 316-324 (2011)
19. Ranu S., Deepak P., Telang A. D., Deshpande P.: Indexing and matching trajectories under inconsistent sampling rates. In: ICDE, pp. 999-1010 (2015)
20. Dai J., Yang B., Guo C., Jensen C. S., Hu J.: Path Cost Distribution Estimation Using Trajectory Data. In: VLDB, pp. 85-96 (2016)
21. Yuan J.,Zheng Y., Xie X., Sun G.: T-Drive: Enhancing Driving Directions with Taxi Drivers. TKDE, 25(1), 220-232 (2013)
22. Hart P. E., Nilsson N. J., Raphael B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100-107 (1968)
23. Gonzalez H., Han J., Li X., Sondag J. P.: Adaptive Fastest Path Computation on a Road Network: A Traffic Mining Approach. In: VLDB, pp. 794-805 (2007)
24. Zheng J., Ni L. M.: Time-dependent trajectory regression on road networks via multi-task learning. In: AAAI, pp. 1048-1055 (2013)
25. Yang B., Kaul M., Jensen C. S.: Using Incomplete Information for Complete Weight Annotation of Road Networks. TKDE, 26(5), 1267-1279 (2013)
26. Guo C., Yang B., Andersen O., Jensen C. S., Torp K.: EcoSky: Reducing vehicular environmental impact through eco-routing. In: ICDE, pp. 1412-1415 (2015)
27. Dai J., Yang B., Guo C., Ding Z.: Personalized route recommendation using big trajectory data. In: ICDE, pp. 543-554 (2015)
28. Yang B., Guo C., Ma Y., Jensen C. S.: Toward personalized, context-aware routing. VLDB J, 24(2): 297-318 (2015)
29. Ding B., Jeffrey X., Qin L.: Finding time-dependent shortest paths over large graphs. In: EDBT, pp. 205-216 (2008)

Huiping Liu, received the BS degree in Software Engineering from East China Normal University, Shanghai, China, in 2013. Currently, he is a PhD student supervised by Prof. Cheqing Jin. His research mainly focuses on location-based services, massive data mining and processing and data quality.

Cheqing Jin, received the BS and master degrees in Computer Science from Zhejiang University, China, in 1999 and 2002, respectively, and then received PhD degree in Computer Science from Fudan University, China, in 2005. Currently, he is an associate professor of Software Engineering Institute, East China Normal University, China. His current research interests include streaming data, uncertain databases, location-based services and data quality.

Aoying Zhou, Professor and dean of the School for Data Science and Engineering at East China Normal University, Shanghai. He is the professorship appointment under Chang Jiang Scholars Program sponsored by Ministry of Education. He is now acting as the vice-director of ACM SIGMOD China and Database Technology Committee of China Computer Federation. He is serving as the associate editor-in-chief of China Journal of Computer, and member of the editorial boards of some prestigious academic journals, such as VLDB Journal, WWW Journal, and etc. His research interests include Web data management, data management for data-intensive computing, management of uncertain data, data mining and data streams, distributed storage and P2P computing.