Distributed and Parallel Databases (2022) 40:109–133
https://doi.org/10.1007/s10619-021-07366-7

A framework for discovering popular paths using transactional modeling and pattern mining

P. Revanth Rathan[1] · P. Krishna Reddy[1] · Anirban Mondal[2]

Accepted: 9 September 2021 / Published online: 20 September 2021

Abstract
While the problems of finding the shortest path and *k*-shortest paths have been extensively researched, the research community has been shifting its focus towards discovering and identifying paths based on user preferences. Since users naturally follow some of the paths more than other paths, the popularity of a given path often reflects such user preferences. Given a set of user traversals in a road network and a set of paths between a given source and destination pair, we address the problem of performing top-*k* ranking of the paths in that set based on path popularity. In this paper, we introduce a new model for computing the popularity scores of paths. Our main contributions are threefold. First, we propose a framework for modeling user traversals in a road network as transactions. Second, we present an approach for *efficiently* computing the popularity score of any path based on the itemsets extracted from the transactions using pattern mining techniques. Third, we conducted an extensive performance evaluation with two real datasets to demonstrate the effectiveness of the proposed scheme.

Keywords Popular paths · Transactional modeling · Pattern mining · Road networks

1 Introduction

The problems of finding the shortest path [6, 7, 23] and *k*-shortest paths [12, 19, 27] have been extensively researched. Path finding and discovery has significant applications in several important and diverse domains such as city planning, transportation,

✉ P. Krishna Reddy
pkreddy@iiit.ac.in

P. Revanth Rathan
revanth.parvathaneni@research.iiit.ac.in

Anirban Mondal
anirban.mondal@ashoka.edu.in

1 Kohli Centre on Intelligent Systems, IIIT, Hyderabad, India

2 Ashoka University, Sonipat, India

vehicular navigation, disaster management, tourism and so on. Interestingly, the recent focus of the research community as well as that of most commercial route planning and navigation systems has been shifting towards discovering and identifying paths based on user preferences.

Such user preferences may include roads with relatively high thoroughfare (i.e., better for safety), smoother roads with better infrastructure (e.g., fewer potholes), roads with more facilities or points of interest nearby, lighted roads as opposed to dark and unsafe streets, roads with relatively lower crime rates, roads with better scenic beauty and so on. In practice, given that users naturally follow some of these paths significantly more as compared to other paths, the *popularity of a given path* often reflects such user preferences. *In this work, we use the notion of popularity as the overarching theme for reflecting the path preferences of users.* Besides traversal time/cost of a route, users also consider the *popularity* of a route as an important factor.

Now we explain the related terminology about road network, paths and user traversals. We model a given road network as a directed graph $G(V, E)$, where V is the set of nodes of the graph. Here, each node represents an intersection of the road segments. Thus, $V = \{v_1, v_2,..., v_n\}$, where v_i represents the ith node. E is the set of edges of the graph, where each edge represents a given road segment. Additionally, the edges are of the form v_i, v_j such that v_i, $v_j \in V$. A path is a spatial trajectory, which is represented by a sequence of edges in the graph G. A given path p_i is of the form $\{(v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n)\}$, where v_i ($i \neq 1$ and $i \neq n$) is the ith node, while v_1 and v_n denote the source node and the destination node respectively. In this work, we consider a user traversal as a path. We consider that a given *user traversal* does not involve any break exceeding a pre-defined threshold time limit t_w, which is essentially application-dependent.

Given a set of user traversals in a road network and a set χ of paths between a given source and destination pair, we address the problem of performing ranking of the paths in χ based on popularity. We refer to this problem as the *Popular Paths (PP) query*.

In the literature, the work in [3] has addressed the problem of detecting popular paths by creating a summary travel path graph based on the user trajectory data. In particular, it has used an adaptive Markov model for computing the likelihood of traveling from a given node to a specific destination. Moreover, an effort has been made in [26] for computing the top-k routes (between a given source and destination pair), which pass a given sequence of locations within a specified time. Furthermore, the work in [2] exhaustively computes all the paths between a given source and destination pair in the summary travel path graph and outputs the top-k popular routes.

Our work is fundamentally differentiated w.r.t. existing works as follows. First, unlike existing works, we model user traversals as transactions. Second, in contrast with existing works, we use the knowledge generated by employing pattern mining techniques towards *efficiently* computing the popularity scores of the paths between a given source and destination pair. Third, we define the popularity of a given path by also taking into account the user traversals, which partially cover the path. Fourth, while existing works focus on detecting the popular paths from the generated summary travel path graph, we identify the popular paths based on a real road network.

Given a source s and a destination t, the issue is to compute the popular paths between s and t. For this purpose, given a set of paths between s and t, we propose a model for determining the popularity score of any given path. The popularity score of a given path is based on the popularity contribution of all user traversals, which contain at least one edge of the given path. In this model, we incorporate the effect of partially covered traversals to the popularity of the path. We also present a formula for computing the popularity contribution of a given user traversal to a given path.

Given a road network G and a set of user traversals, a straightforward brute-force approach to compute the popularity score of a given path between s and t is as follows. We need to compute the sum of the popularity contributions of all the corresponding user traversals. Such an approach would be prohibitively expensive because the number of corresponding user traversals would explode as the number of edges in the path increases. Hence, we propose an *efficient* approach by using transaction modeling and pattern mining. Pattern mining is interesting as it extracts interesting frequent itemsets from the given set of transactions. By observing that each user traversal can be modeled as the transactions and the fact that comprehensive algorithms are already available in the literature to extract frequent patterns from the large transactional datasets, extending pattern mining approach to compute the popularity score of the path will be a valuable contribution. By considering edges of user traversals as items, we convert all user traversals into a corresponding set of transactions. Then we extract all possible patterns (i.e., combinations of edges) with a corresponding value of support (i.e., the percentage of transactions in which a given pattern appears) in an *offline mode* by exploiting the apriori pruning property of pattern mining algorithms. Given a path between any s and t, we compute the popularity score of the path based on the knowledge of patterns and the corresponding support values.

The main contributions of this work are threefold:

1. We propose a framework for modeling user traversals in a road network as transactions.
2. We present an approach for efficiently computing the popularity score of any path based on the itemsets extracted from the transactions using pattern mining techniques.
3. We conduct an extensive performance evaluation with two real datasets to demonstrate the effectiveness of the proposed scheme.

In particular, the results of our performance evaluation on the real datasets indicate that by traversing a relatively small additional distance, it is possible to *quickly* extract more number of popular paths, thereby providing more flexibility towards path recommendations for users.

In [21], we have made an preliminary effort to present the popularity model. In this paper, we have extended the paper by refining the model, algorithm, and carried out extensive performance experiments by considering the additional dataset.

The remainder of the paper is organized as follows. In Sect. 2, we discuss related work and background. In Sect. 3, we present the modeling of the popularity score of a given path. In Sect. 4, we discuss our proposed scheme. In Sect. 5, we report the performance evaluation. Finally, in Sect. 6, we conclude the paper.

2 Related work and background

In this section, we discuss existing works and background.

2.1 Related work

The problem of finding and efficiently computing the shortest path has been researched in [6, 7, 20, 23, 24]. The problem of finding top-k shortest paths have been addressed in [12, 19, 27]. In particular, Yen's algorithm [27] finds loop-less top-k shortest paths. Yen's algorithm exploits the idea that the kth shortest path may share edges and sub-paths (path from source to any other intermediate nodes within the path) from the $(k-1)$th shortest path. Yen's algorithm first finds the shortest paths using Dijkstra algorithm [7]. It then takes every node in the current shortest path, except for the destination and computes the shortest paths from each selected node to the destination. The problem of finding top-k diverse shortest paths has been addressed in [4, 5, 17]. The work in [5] proposes exact and approximate algorithms to discover the top-k diverse shortest paths based on the concept of *limited overlap* by using Yen's algorithm.

Research efforts are being made to find popular paths based on the spatial trajectories of users [2, 3, 26]. The work in [3] uses an adaptive Markov model for computing the popularity of a given sequence of edges in order to compute the most popular route. This model is used to deduce the transfer probability for each edge in the path. Here, the transfer probability refers to the probability of traversing from that edge to the destination. The work in [2] constructs the familiar road network from the user trajectory dataset and extracts all paths from the source to a specific destination. Moreover, it ranks the paths based on the popularity, which is computed using the count of user traversals and the length of the path. Notably, the approaches proposed in [2, 3] do not consider the effect of partial traversals in determining the popularity score of any given path.

The proposal in [26] constructs a route inference model and a routable graph construction from uncertain trajectories for determining the top-k popular paths, which comprise only trajectory points. A popularity-related weight is assigned to each and every edge of the path based on the inference model for computing the popularity of the path. The working of prototype Trip [15] presents a route based on the trajectories of the users. In particular, Trip improves the routing by considering the real historical data of users and learning individual preferences, which can be applied to future route planning scenarios.

The work in [13] proposed a two-step routable graph construction method for determining popular paths using a set of historical check-in data and a set of query locations specified by a user. The work in [10] proposed a collaborative method for discovering popular routes using taxi drivers' experience and preferences. Traffic density based methods have also been proposed to find *hot routes* in road networks. The work in [16] proposed a density-based algorithm to group roads based on their shared common traffic and proposed a framework to find popular routes. The work in [18] proposed a fuel-efficient route plan method based on game theory. The proposal in [30] used an approach to mine popular travelling sequences based on the users' travel experience and location interests.

The work in [25] uses a route score function that strikes a balance between user preference degrees and the length of the route as a measure of the popularity. The approach proposed in [11] recommends the appropriate path with multi-preferences for a user, while the user selects preferred attributes and provides their respective weights. Furthermore, the work in [14] introduced two novel concepts to trajectory indexing and discussed algorithms for various types of spatio-temporal queries that involve routing in road networks such as finding all paths and vehicles in a road network. A detailed survey of trajectory data mining techniques can be found in [8].

The work in [22] studies the problem of finding the most preferred path between the source and destination in the network. They defined the notion of preferred zones that users are more acquainted to drive through and the objective is to minimize the time spent outside of the preferred network. In contrast, in our work, the main goal is to maximize a collective popularity score of the path instead of reducing the travel time in a non-preferred edge.

Frequent pattern (FP) mining is one of the important concepts in data mining. The process of mining FPs extracts the interesting information about the associations among the items in patterns in a transactional database based on a user-specified *support (frequency)* threshold. Mining FPs has been extensively studied in the literature [1].

Notably, our work differs from existing shortest path approaches and diverse path computation approaches in that we address the problem of extracting the top-k popular paths for a given source and destination pair. As discussed previously in Sect. 1, our work fundamentally differs from existing approaches in that we model user traversals as transactions and use pattern mining techniques for *efficiently* computing the popularity scores of the paths between a given source and destination pair. Furthermore, unlike existing works, we identify popular paths based on a real road network as opposed to a generated summary travel path graph. Additionally, we define the popularity of a given path based on the *effective* number of user traversals that cover the path. Table 1 describes the differences with the work in the literature.

Table 1 Differences with the work in the literature

References	Computation of popularity	Computation model for computing popular path
[2]	Popularity of the path is computed using route scoring function, which reflects both personalized preferences and also the length of the path	Based on a summary graph formed
[3]	Popularity of the path is computed using Adaptive Markov model, which helps in deducing the transfer probability of an edge	Based on a summary graph formed
[13]	Popular paths are determined using a set of historical check-in data and a set of query locations specified by a user	Based on a summary graph formed
[26]	Popular paths from uncertain trajectories are constructed using inference framework based on collective knowledge	Based on a summary graph formed
Our proposed model	Popularity score of the path is computed using the combined effect of partially or fully covered user traversals	Based on frequent patterns

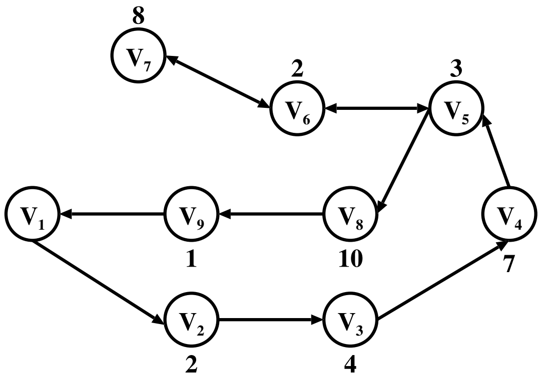


Fig. 1 User traversal of the delivery person

3 Model of popularity score of a path

In this section, we first discuss the terminology related to this paper. Later we develop a new model for computing the popularity score of a given path and subsequently, we present the equation for computing the popularity score of the given path. We defer the discussion on the proposed approach for the efficient computation of top-*k* popular paths from a set of paths to Sect. 4.

3.1 Definitions

Now we shall introduce the definitions of user traversal (*UT*) and user traversal transaction (*UTT*).

User Traversal (*UT*): *UT* is a sequence of edges traversed by the user in the given graph particularly for a long period of time (12 hours, a day, a week etc.). In a given *UT*, the user might perform different types of tasks. For example, consider an amazon delivery person, the *UT* for a particular period of time is given in Fig. 1. The number under each node represents the time spent (in min) by the user at that particular node. Since the user starts and ends at the node v_1, there was no number under node v_1. The tasks for the above user includes picking up the orders and delivering them to multiple customers and to submit the returned products at the warehouse. In Fig. 1, the user picks multiple orders from the warehouse at node v_1. The delivery location of the customers are node v_4 and node v_7. The pickup location of the returned order is node v_8. The user returns the picked up order at node v_1 (warehouse) at the end of the traversal.

This complete sequence of edges traversed by the user represents a single *UT* i.e., $\{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_5, v_6), (v_6, v_7), (v_7, v_6), (v_6, v_5),$
$(v_5, v_8), (v_8, v_9), (v_9, v_1)\}$.

User Traversal Transaction (*UTT*): *UTT* is a sequence of edges extracted from the *UT* such that each sequence has a specific task attached to it. In the above example, the main tasks for the amazon delivery person includes delivering, picking up the order and returning the order at the warehouse. We need to develop the methods such that each *UTT* should be assigned with a specific task. We can convert the *UT* to *UTTs* using the following methods. As a part of first method, the conversion of a given *UT* to *UTTs* is carried through a timeout mechanism. If the user stops at any particular location for a reasonable duration t_w, we consider that the user has started a new task i.e., new *UTT*. Here, t_w is a threshold value. Hence, the parts of the *UT* after that location are considered to be a new *UT*. As a part of second method, for a given *UT*, whenever a user backtracks or visits one of the already visited nodes, the edges visited so far are considered as one *UTT*; the traversal process is then continued from the preceding node. For both methods, the same process is repeated recursively till the last node to identify further *UTTs*.

Let us consider the duration t_w is 5 min. The *UT* in Fig. 1 can be converted to the following *UTTs* using the afore-mentioned duration method.

UTT_1—$\{(v_1, v_2), (v_2, v_3), (v_3, v_4)\}$; UTT_2—$\{(v_4, v_5), (v_5, v_6), (v_6, v_7)\}$;

UTT_3—$\{(v_7, v_6), (v_6, v_5), (v_5, v_8)\}$; UTT_4—$\{(v_8, v_9), (v_9, v_1)\}$

By using the backtracking method, the *UT* in Fig. 1 can be converted to the following *UTTs*:

UTT_1—$\{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_5, v_6), (v_6, v_7)\}$;

UTT_2—$\{(v_7, v_6), (v_6, v_5), (v_5, v_8), (v_8, v_9), (v_9, v_1)\}$

3.2 Model of computing popularity score

Now we shall introduce the notion of *popularity score* of a given path p_i. Let n_i denote the number of edges of a given path p_i. We shall henceforth designate the popularity score of p_i as $\omega(p_i)$. Intuitively, the value of $\omega(p_i)$ depends upon the number of user traversal transactions (*UTTs*) that traverse p_i. In practice, all *UTTs* need not necessarily traverse all n_i edges in p_i. We can divide the whole traversals of all users into a set of traversals, which cover only 1 edge, only 2 edges and so on up to the set of traversals that cover n_i edges of p_i. In this model, we incorporate the effect of partially covered traversals to the popularity of the path. To compute the effect of such partially covered traversals on popularity, we shall now introduce the notion of the *popularity contribution* $PC(q_i, p_i)$ of a *UTT* q_i w.r.t. a given path p_i. Intuitively, $PC(q_i, p_i)$ indicates the extent to which a *UTT* q_i contributes to the popularity of p_i.

Consider a path p_i between a source s and a destination t with n_i edges. Let $Q(p_i)$ represent the number of *UTTs*, which cover *at least* one edge of p_i. The *popularity score* $\omega(p_i)$ of p_i equals the sum of the individual popularity contributions of $Q(p_i)$ *UTTs*. We define $\omega(p_i)$ as follows:

$$\omega(p_i) = \sum_{q_i=1}^{|Q(p_i)|} PC(q_i, p_i) \tag{1}$$

Here, $0 \leq PC(q_i, p_i) \leq 1$. When all edges of p_i are covered by q_i, $PC(q_i, p_i) = 1$. Conversely, when none of the edges of p_i is traversed by q_i, $PC(q_i, p_i) = 0$.

Equation 1 captures *PC* of all kinds of *UTTs*. The issue is to compute the value of $\omega(p_i)$ by considering all kinds of *UTTs* if every edge in a given path p_i satisfies the threshold condition. First, for simplicity, we present a method for estimating $\omega(p_i)$ by considering only those *UTTs*, which cover all of the edges of the path. (This hypothetical case may not hold good in practice.) Second, we discuss a method for estimating $\omega(p_i)$ by considering the *UTTs*, which pertain to the traversal of a fixed number of edges in the path. Third, we present a method for the generalized case arising in real-world scenarios by considering all kinds of *UTTs*. Now let us discuss these three cases in detail.

Case 1 -UTTs which cover all edges of p_i: Given a path p_i with n_i edges, and a *UTT* q_i, if q_i traverses all edges of p_i, $PC(q_i, p_i) = 1$; otherwise, it is 0. Let $Q(p_i, n_i)$ represent the number of *UTTs*, which cover all n_i edges of p_i. Using Eq. 1, the popularity score of p_i is as follows:

$$\omega(p_i) = Q(p_i, n_i) \tag{2}$$

Given two paths p_1 and p_2 between source s and destination t with the number of *UTTs*, which cover all edges of p_1 and p_2, being $Q(p_1, n_1)$ and $Q(p_2, n_2)$ respectively, $\omega(p_1)$ $\omega(p_2)$ iff $Q(p_1, n_1)$ $Q(p_2, n_2)$. This matches our intuitive understanding of the notion of popularity score, which depends on the number of *UTTs* of the path.

Case 2 -UTTs which cover exactly k edges of p_i: In practice, any given user may use only one or more edges of the path and then take a detour into other edges/paths. Case 1 does not capture this real-world scenario. In Case 1, if q_i covers all n_i edges of p_i, $PC(q_i, p_i)$ is 1.

Now, if q_i covers only k edges of p_i, $PC(q_i, p_i) = W(k)$, where W is a function with k as the parameter. W can be any monotonically increasing function w.r.t. k. Thus, we can define W as follows:

$$W(k) \propto k \; and \; 0 \leq W(k) \leq 1.$$

If q_i does not cover any edge of p_i, $PC(q_i, p_i) = W(0) = 0$. (It can be noted that we consider the *UTT* with k edges even though the edges of the *UTT* are not continuous.)

Different functions can be selected for weight function W. Intuitively, given two traversals, a traversal, which covers more edges of p_i contributes more to $\omega(p_i)$. Hence, W should be selected such that given *UTTs* q_1 and q_2, which cover k_1 and k_2 edges of p_i such that $k_1 < k_2$, $W(k_1)$ should be less than $W(k_2)$.

We denote $Q(p_i, k)$ be the number of *UTTs*, which cover only k edges in p_i. The number of edge combinations of size k from the path p_i consisting of n_i edges is $\binom{n_i}{k}$. We can divide the $Q(p_i, k)$ *UTTs* into the *UTTs*, which traverses each and every combination of k edges.

$$Q(p_i, k) = \sum_{j=1}^{\binom{n_i}{k}} Q(p_i, k, j) \tag{3}$$

Here, $Q(p_i, k, j)$ represents the number of *UTTs*, which traverse the jth combination of k edges. Using Eq. 1, we obtain:

$$\omega(p_i) = \sum_{q_i=1}^{|Q(p_i,k)|} PC(q_i, p_i) = Q(p_i, k) * W(k) \tag{4}$$

Given two paths p_1 and p_2 between source s and destination t with the number of *UTTs*, which cover k edges of p_1 and p_2, being $Q(p_1, k) * W(k)$ and $Q(p_2, k) * W(k)$, $\omega(p_1)\ \omega(p_2)$ iff $\left(Q(p_1, k) * W(k)\right)\left(Q(p_2, k) * W(k)\right)$. Observe that, if k is equal to n_i, by substituting the value of k as n_i in Eq. 4, we obtain the same popularity score as in Case 1.

Case 3—All UTTs which cover p_i: We can divide $Q(p_i)$ *UTTs* into *UTTs*, which cover only 1 edge, only 2 edges and up to all n_i edges of the path. Let $Q(p_i, k)$ be the number of *UTTs*, which cover only k edges in p_i.

Thus, the value of $Q(p_i)$ is computed as follows:

$$Q(p_i) = \sum_{k=1}^{n_i} Q(p_i, k) \tag{5}$$

Each of these *UTTs* contributes to the popularity score of the path. Thus, using Eqs. 4 and 5, we can compute $\omega(p_i)$ as follows:

$$\omega(p_i) = \sum_{q_i=1}^{|Q(p_i)|} PC(q_i, p_i) = \sum_{k=1}^{n_i} Q(p_i, k) * W(k) \tag{6}$$

Intuitively, given two paths, we consider that a path, which is covered by more *UTTs* with a larger combination of edges, is more *popular* than the path, which contains a relatively lower number of *UTTs*. Observe how this intuition of popularity is reflected in Eq. 6.

3.3 Computing of popularity score

If path p_i contains an edge having zero *UTTs* or *UTTs* having count less than the threshold value traversing through it, from Eq. 6, we observe that some value of popularity score is assigned to p_i. In general, we say that a path is popular if each and every edge of the path is traversed by at least a threshold number of *UTTs*, which is denoted by the minimum frequency *minF* threshold. Let us consider the path p_i with n_i edges. Let the number of *UTTs*, which traverse through an edge e in the path p_i, be $RF(e)$. The threshold condition for a path to be considered as popular is as follows:

$$\forall e \in p_i; RF(e) \geq minF \tag{7}$$

Based on our above discussion, using Eq. 6, the equation for $\omega(p_i)$ for the generalized real-world case is as follows:

$$\omega(p_i) = \begin{cases} \sum_{k=1}^{n_i} Q(p_i, k) * W(k), & \forall e \in p_i; RF(e) \geq minF \\ 0. & \text{otherwise.} \end{cases} \tag{8}$$

The popularity score $\omega(p_i)$ of the path p_i captures the effective number of users traversing through it. In general, if all the users traverses all the edges in the path (see Case 1), the popularity of the path is maximum and it is equal to the number of users traversing the path. If the *UTTs* do not traverse through any edge of the path or the number of *UTTs* traversing through each edge of the path is less than that of the *minF* threshold, the popularity of the path is minimum and it is equal to zero.

4 Popular paths query and proposed scheme

In this section, we introduce the *Popular Paths* (PP) *query* and discuss the proposed scheme.

The *popular paths (PP)* query is as follows. Given a set of user traversals in a road network G, source s, destination t and a set χ of paths between s and t, the PP query determines the ranked list L of popular paths in χ in descending order of popularity score. (Recall the computation of popularity score in Eq. 8.) Here, any ties in path popularity scores are resolved arbitrarily.

Now we shall discuss the basic idea of the proposed scheme.

4.1 Basic idea

To process a given PP query, we need to compute the popularity scores of paths in set χ of paths (using Eq. 8). However, this requires the frequencies of potential combinations of edges of user traversals, which traversed at least one edge of p_i. Hence, given a path p_i comprising n_i edges, we need to examine the frequencies of $2^{n_i} - 1$ combinations of these edges. Notably, for every path p_i, when a PP query comes in, it would be prohibitively expensive to generate all combinations of edges and compute their frequencies in an *online* manner because each path may have a different set of edges. However, there is an opportunity here for *offline* pre-processing and extraction of the knowledge of frequency of edge combinations; we designate such knowledge as *patterns*. Thus, when a PP query comes in, we can use the extracted patterns towards *efficiently* processing the PP query *online*.

We consider this modified query as PP. Our PP query processing scheme is as follows. First, we convert the user traversals into a transactional dataset *D*. Next, we extract the knowledge of patterns from *D* by using the existing FP-Growth [9] algorithm. Then we compute the popularity score of each path of χ using Eq. 8, extract the set PP of popular paths and order the paths based on popularity score.

4.2 Proposed scheme

The proposed scheme receives the set of user traversals (*UT*), minimum frequency (*minF*) threshold and the set of paths χ as an input and the top-*k* popular paths from χ as an output.

The proposed scheme has three phases: (1) formation of user traversal transactions (*UTTs*) (2) extraction of combinations of edges and their frequencies from *UTTs* (3) computation of top-*k* popular paths from χ. Now we shall discuss each phase.

1. Formation of user traversal transactions (*UTTs*) Here, the input is a set of user traversals (*UTs*). Recall that, *UT* is a sequence of edges traversed by the user in the given graph particularly for a long period of time (12 hours, a day, a week etc.). Given a *UT*, the time spent (in min) by the user is shown at each node. We can convert each *UT* to one or more number of *UTTs* using one of the following methods. As a part of first method, the conversion of a given *UT* to *UTTs* is carried through a timeout mechanism. If the user stops at any particular location for a reasonable duration t_w, we consider that the user has started a new task i.e., new *UTT*. Here, t_w is a threshold value. Hence, the parts of the *UT* after that location are considered to be a new *UT*. As a part of second method, for a given *UT*, whenever a user backtracks or visits one of the already visited nodes, the edges visited so far are considered as one *UTT*; the traversal process is then continued from the preceding node. For both methods, the same process is repeated recursively till the last node to identify further *UTTs*.

Algorithm 1: Compute_DBEC(G, UT)

Input : G: Graph; UT: User Traversals; $minF$: minimum frequency

Output: $DBEC$: Database of edge combinations

1 Form $UTTs$ from UT

2 Extract edge combinations (and corresponding frequencies) which satisfy minimum support $minF$ from $UTTs$ using FP-growth and store in $DBEC$

2. Extraction of combinations of edges and their frequencies from UTTs: Here, the input is *UTTs* and the output is the database of edge combinations (*DBEC*). By considering *UTTs* as transactions and edges as items, all combinations of edges with support greater than or equal to the user-specified value of *minF* is extracted by the existing FP-growth frequent pattern mining algorithm [9]. Here, the entries in *DBEC* are of the form <edge combination, frequency>. Algorithm 1 depicts the creation of *DBEC*.

Algorithm 2: $PP(\chi, DBEC, \theta, W, k)$

Input : χ: set of paths; $DBEC$: list of $<e_i, f_i>$. Here, f_i represents the frequency of i^{th} edge combination e_i; W: weight function; k: required number of paths

Output : PP: popular paths

Variables: F: list of $<e_i, f_i>$

1 Initialize PP to null
2 **foreach** $p_i \in \chi$ **do**
3   Generate all edge combinations of p_i
4   Initialize F to null
5   **foreach** e_i *of* p_i **do**
6   Extract $<e_i, f_i>$ from DBEC and store in F //We use hash map
7   Initialize ps to zero
8   **foreach** $<e_i, f_i> \in F$ **do**
9         $count$=number of edges in e_i
10   ps = ps + (W(count) * f_i) // W is the given weight function
11   Insert $<p_i, ps>$ to PP
12 Sort PP w.r.t. ps in the descending order

3. Computation of top-k popular paths from χ: Here, the input is *DBEC* and the set of paths χ and the output is top-*k* popular paths. Algorithm 2 depicts the computation of top-*k* popular paths (PP). We compute the popularity score $\omega(p_i)$ of a given path p_i in χ using Eq. 8. We compute $\omega(p_i)$ by extracting (from *DBEC*) the knowledge of the respective frequencies of *UTT*s of all edge combinations, which are formed with the edges of p_i. We use the principle of inclusion and exclusion to obtain the frequencies of *UTTs*, which traverse *only* certain edge combinations from *DBEC*. In Algorithm 2, the computation of $\omega(p_i)$ is shown in Lines 3-10. Here, computing all combinations of edges in a given path p_i would be prohibitively expensive. Hence, we use a hash based indexing on *DBEC* to obtain the edge combinations, which are a subset of p_i, and their corresponding frequencies (see Line 6). After computing the popularity scores for all paths in χ, we sort the paths in descending order based on their respective popularity scores into a list PP (see Line 12).

4.3 Illustrative example

Figure 2 represents all possible paths from the part of the graph with source node as 1 and destination node as 4. There might be other edges from the nodes present in the graph, which are not leading to the destination node 4. Hence, the people, who are traversing through a certain node, may leave through any edge connecting to other nodes, which is not shown in Fig. 2.

We explain the working of our approach through an example. Consider a large number of users traversal transactions (say, about 1000) in the road network depicted in

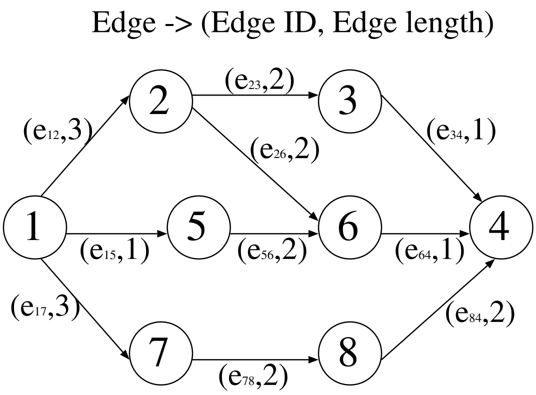


Fig. 2 Sub-graph with all paths from node 1 to node 4

Table 2 DBEC of user traversals

Edge combination	Frequency	Edge combination	Frequency
$\{e_{12}\}$	40	$\{e_{12}, e_{26}\}$	25
$\{e_{23}\}$	20	$\{e_{23}, e_{34}\}$	40
$\{e_{34}\}$	30	$\{e_{15}, e_{56}\}$	15
$\{e_{15}\}$	6	$\{e_{15}, e_{64}\}$	10
$\{e_{56}\}$	5	$\{e_{56}, e_{64}\}$	20
$\{e_{64}\}$	4	$\{e_{17}, e_{78}\}$	10
$\{e_{17}\}$	100	$\{e_{17}, e_{84}\}$	15
$\{e_{78}\}$	50	$\{e_{78}, e_{84}\}$	15
$\{e_{84}\}$	60	$\{e_{26}, e_{64}\}$	20
$\{e_{26}\}$	80	$\{e_{12}, e_{23}, e_{34}\}$	40
$\{e_{12}, e_{23}\}$	30	$\{e_{12}, e_{26}, e_{64}\}$	30
$\{e_{12}, e_{34}\}$	20	$\{e_{15}, e_{56}, e_{64}\}$	15
$\{e_{12}, e_{64}\}$	15	$\{e_{17}, e_{78}, e_{84}\}$	0

Fig. 2, which consists of four paths, namely p_1 (e_{12}, e_{23}, e_{34}), p_2 (e_{15}, e_{56}, e_{64}), p_3 (e_{17}, e_{78}, e_{84}) and p_4 (e_{12}, e_{26}, e_{64}) from the source node 1 to the destination node 4. Table 2 contains the details of edge combinations with frequencies (*DBEC*), which can be obtained from the *UTTs* by using the existing frequent pattern mining algorithm (FP-growth [9]). Given source node 1 and destination node 4, for computing the top-*3* PPs, we need to compute the popularity score (ω) of four paths using Eq. 8.

We consider the weight function $W(i)$ as $\frac{i*(i+1)}{n*(n+1)}$, where i is the number of edges in the combination and n represents the number of edges in the path. Popularity scores (ω) of the paths in Fig. 2 are computed as follows:

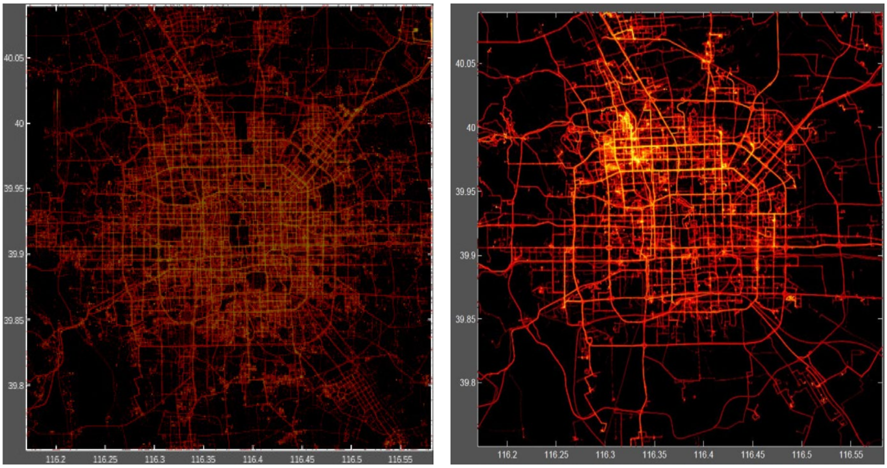
$$
\begin{aligned}
\omega(p_1) &= \sum_{k=1}^{3} W(k) * Q_{p_{1k}} \\
&= W(1) * Q_{p_{11}} + W(2) * Q_{p_{12}} + W(3) * Q_{p_{13}} \\
&= \frac{2}{12} * (40 + 20 + 30) + \frac{6}{12} * (30 + 20 + 40) \\
&\quad + \frac{12}{12} * 40 = 100
\end{aligned}
$$

Similarly, the values of $\omega(p_2)$, $\omega(p_3)$ and $\omega(p_4)$ are 40, 55 and 80.67 respectively. Paths are arranged in the descending order $p_1 > p_4 > p_3 > p_2$ based on the popularity score. Here, notice that p_1 is more popular than p_3, even though the number of users traversing through p_3 is more than that of p_1. This is because more users are traversing through the combinations of edges, which results in the increase in $\omega(p_1)$ w.r.t. $\omega(p_3)$. Even though p_2 has more users traversing through the combinations of edges than p_3, the *effective number* of users traversing through p_2 is significantly low when compared to that of p_3. As a result, p_3 is receiving more popularity score w.r.t. p_2. Hence, the final PP list is (p_1, p_4, p_3, p_2) .

4.4 Time and space complexity

DBEC is extracted from UTTs in an offline manner using Algorithm 1, which employs an existing pattern extraction algorithm such as FP-growth [9]. Performing an algorithm offline implies that we can execute the algorithm in advance and can materialize the knowledge in the repository. We can use the materialized knowledge for performing multiple queries/requests without carrying out the computation from scratch for each query. Given χ, source and destination, *PP*s are extracted using Algorithm 2 online. Performing an algorithm in an online manner means we need to execute an algorithm from the scratch whenever the query/request comes in by accessing the user traversal data from the disk. So, algorithm execution occurs for each query and is I/O intensive. Here, the size of DBEC is much smaller than the user traversal data and so the queries can be executed in the main memory without accessing disk.

Now, we present the complexity of Algorithm 2. In order to compute the popularity score of the path, we need to generate each and every combination of edges in the path whose frequency is greater than *minF*. Along with generation of combinations, we also need to compute the frequency of each and every generated combination. By using frequent pattern mining, we can get an advantage of computing the combination of edges along with their frequency in an effective manner by reusing the generated knowledge. By using hash based indexing, we can easily retrieve the edge combinations per path from the DBEC. Let *TC* denote the number of edge combinations (in DBEC) for all paths in χ. The average number C of edge combinations per path is $TC/|\chi|$. The time complexity for the different steps is as follows: (i) computing the popularity score for all paths in χ is $O(|\chi| * C)$ (ii) computing top-k *PPs* is $O(|\chi| * log(|\chi|))$. Hence, the total time complexity is $O(|\chi| * C)$ $+O(|\chi| * log(|\chi|))$. In general, $C \gg |\chi|$. Hence, the *time complexity* of Algorithm 2



(a) T-Drive dataset (b) Geolife dataset

Fig. 3 Heat maps of the user traversals

comes to $O(|\chi| * C)$. Moreover, the *space* required for Algorithm 2 is equal to the space for storing all the paths in χ and *DBEC*.

5 Performance evaluation

We conducted our performance evaluation on an Intel i7 processor with 8GB RAM running Ubuntu Linux. We used the data from OpenStreetMap[1] to obtain the complete road network data of Beijing and stored the road network data using the OSMnx[2] framework.

We used Microsoft's *T-Drive* [28] and *Geolife* [29] real user trajectory datasets. T-Drive dataset [28] contains the GPS trajectories of 10,357 taxis during the period of February 2 up to February 8, 2008 within Beijing. The total number of points in this dataset is about 15 million and the total distance of the trajectories reaches to 9 million kilometers. The average sampling interval is about 177 seconds with a distance of about 623 meters. Each file of this dataset, which is named by the taxi ID, contains the trajectories of one taxi. In this work, we considered 10,290 traversals in Beijing. Figure 3a depicts the heat map of the user traversals in the T-Drive dataset within Beijing's fifth Ring Road.

Geolife GPS trajectory dataset was collected in the Geolife project by 182 users during a period of over 5 years. The dataset contains 17,621 trajectories with a total distance of 1,292,951 kilometres and a total duration of 50,176 hours. These trajectories were recorded by different GPS loggers/phones with varied sampling rates.

[1] https://www.openstreetmap.org.

[2] https://osmnx.readthedocs.io/en/stable/.

Each trajectory is a sequence of time-stamped points, each of which contains information about latitude, longitude and altitude. In this work, we considered 14,175 traversals in Beijing. Figure 3b depicts the heat map of the user traversals in the Geolife dataset within Beijing's fifth Ring Road.

We mapped each trajectory of the user to the road network using a map-matching tool Graphhopper[3]. Then we used nominatim[4] to obtain the corresponding sequence of edge IDs; we consider this sequence as a path.

We have used the backtracking method (see Sect. 4.2) for dividing user traversals into multiple user traversal transactions (*UTTs*) i.e., whenever a user backtracks or visits one of the already visited nodes, the edges visited so far are considered as one *UTT*, and we consider the next edge in the user traversal as the starting point of the next *UTT*. We repeat the above procedure for each user traversal. We deployed the FP-Growth algorithm [9] to obtain all frequent patterns from *UTTs* with support (*minF*) greater than 300 for both datasets. We can set the *minF* parameter initially as 0.1*|*D*|, where |*D*| represents the size of the *UTT* and reduce it until the pattern explosion problem occurs (due to large number of frequent patterns). To select the candidate set of paths between s and t, we introduce the notion of distance threshold δ. Let sl be the shortest path length between s and t. We extract all paths, whose length is less than $(sl + \delta * sl)$.

The objective is to demonstrate that by traveling a little more distance than the shortest path between two nodes, we can get an adequate number of popular paths. Another objective is to ensure that our recommended set of paths have a greater popularity score as compared to the reference approaches (shortest paths and diverse shortest paths). We also shown the significant difference in execution time through visualization when we use frequent patterns for computing the popular paths.

We conducted our experiments with three representative queries, where each query is a source and destination pair. To generate these queries, we randomly selected 25 queries each from areas of low, medium and high spatial densities such that the distance between each of the source and destination pairs is less than 1 km. In the context of this paper, spatial density implies the number of roads in a given area. Then, for each type of region (i.e., regions having low, medium and high spatial densities), we randomly selected five queries from among these 25 queries and grouped them into a single query. We shall henceforth refer to the queries from regions of high, medium and low spatial densities as Q1, Q2 and Q3 respectively.

Our performance metrics include average length AL of a set of paths, average popularity score APS of a set of paths, number of popular paths (NPP), number of diverse popular paths (NDP) and execution time ET.

As reference, we used the implementation of our model for computing the popularity scores of paths (see Sect. 3) without using frequent pattern information. Given a transactional database with user traversal transactions, we traverse all of the transactions online to obtain the frequencies of each and every edge combination of each

[3] https://graphhopper.com/api/1/docs/map-matching/.

[4] https://nominatim.openstreetmap.org/.

Table 3 Performance study parameters

Parameter	Default	Variations
Distance threshold (δ)	0.2	0.1, 0.3, 0.4, 0.5
Diversity threshold (θ)	0.6	0.2, 0.4, 0.8, 1

path. In this approach, we construct *DBEC* for a specific path through online traversal. We shall henceforth refer to this reference scheme as the *online approach*.

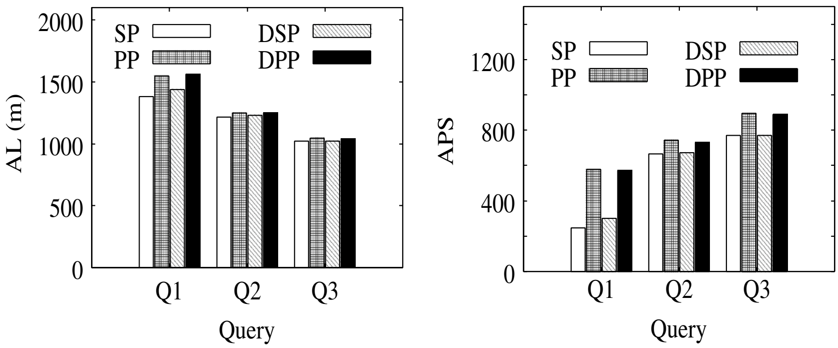
Additionally, as reference, we have used three different approaches. First, we used the Yen's algorithm implemented in OSMnx framework for computing the top-k shortest paths. Yen's algorithm first finds the shortest paths using Dijkstra algorithm [7]. It then takes every node in the current shortest path, except for the destination, and computes the shortest paths from each selected node to the destination. Henceforth, we designate this approach as the top-k *S*hortest *P*aths approach, and we abbreviate it as *SP*.

Furthermore, we also introduce the notion of diversity for the sake of comparison. We used the notion of diversity mentioned in [4] to compute top-k diverse shortest paths and top-k diverse popular paths from the output paths of the SP and PP scheme respectively. We abbreviate the top-k *D*iverse *S*hortest *P*aths and top-k *D*iverse *P*opular *P*aths approaches as *DSP* and *DPP* respectively. In [4], diversity value (DV) between any two given paths is computed as $DV(p_i,\ p_j) = 1 - \frac{\sum_{e \in p_i \cap p_j} l(e)}{\sum_{e \in p_j} l(e)}$, where $l(e)$ represents the length of the edge e. We deem the *diversity threshold criterion* to be satisfied for any two paths p_i and p_j if $DV(p_i, p_j) \geq \theta$, where θ is the diversity threshold. To compute the set of diverse shortest paths, we first add the shortest path to the output set in DSP approach. Then we add the next shortest path to the output set, if the diversity threshold criterion is satisfied for each path in the output set of DSP. We follow the above procedure until we obtain k paths or all of the paths in output of SP have been exhausted. Similarly, we compute the output of DPP from PP.

Table 3 summarizes the parameters of our performance study. Given source and destination, for Q1, Q2, and Q3, we compare our proposed PP scheme with the reference approaches, namely SP, DSP and DPP. We also compare the performance of PP with the online approach.

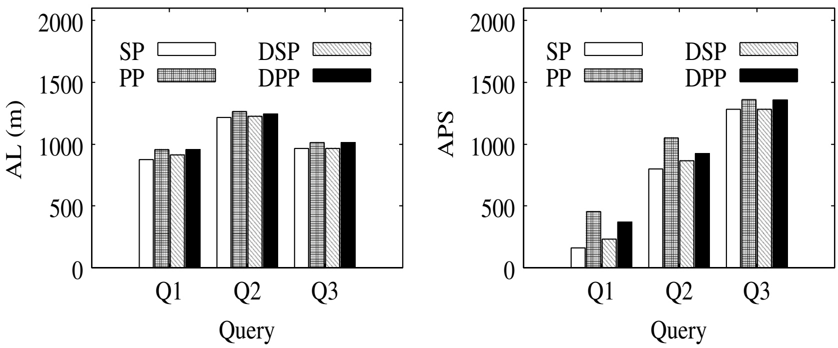
5.1 Results on average length (AL) and average popularity score (APS)

Figure 4 depicts the results on average length (AL), and average Popularity Score (APS) of the comparison approaches for the T-Drive dataset. From Fig. 4a, we can observe that AL of the top-k shortest paths (SP) is not more than AL of the top-k popular paths (PP), top-k diverse shortest paths (DSP), and top-k diverse popular paths (DPP) and AL of DSP is not more than AL of DPP. This occurs because the length of the popular path, diverse popular path, and diverse shortest path are greater than or equal to the length of the shortest path, and the length of the diverse popular path is greater than or equal to the length of the diverse shortest path. However, the



(a) Average Length (b) Average Popularity Score

Fig. 4 Results on queries in regions of varying spatial densities in T-Drive dataset

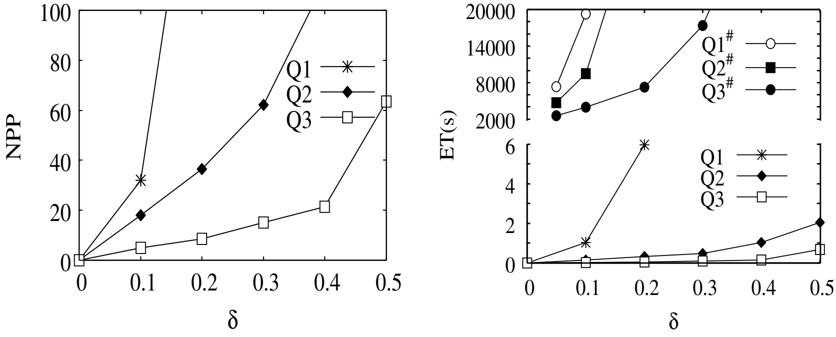


(a) Average Length (b) Average Popularity Score

Fig. 5 Results on queries in regions of varying spatial densities in Geolife dataset

additional distance is not significant. The results show that it is possible to obtain popular paths for a given source and destination by traveling a small additional distance.

For Q1, Q2, and Q3, Fig. 4b shows the results of APS of top-k SP, PP, DSP and DPP. From Fig. 4b, we can observe that APS of SP, DSP, and DPP is less than APS of PP and APS of DSP is less than APS of DPP. This occurs because the popularity scores of the shortest path, diverse shortest path, and the diverse popular paths are not greater than the popularity score of the popular path, and the popularity score of the diverse popular path is greater than or equal to the popularity score of the diverse shortest path. The results also show that independent of spatial density, it is possible to obtain popular paths for Q1, Q2, and Q3 as there are a reasonable number of user traversals in the order of thousands for each query. The popularity score is independent of spatial density w.r.t. roads because the proposed notion of popularity considers user traversals.



(a) NPP (b) ET

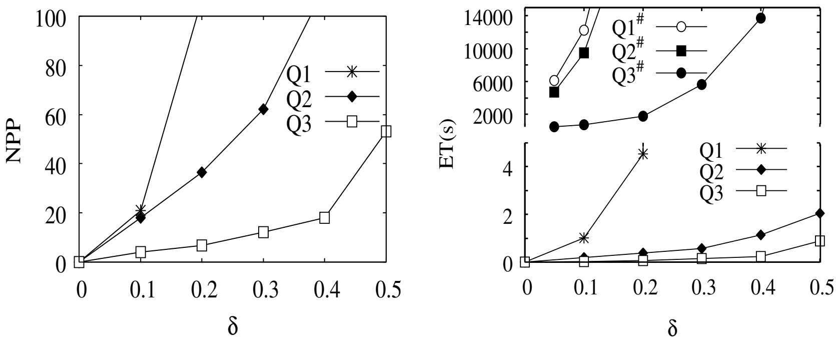
Fig. 6 Effect of variations in the distance threshold in T-Drive dataset

We observe a similar trend in the results for the Geolife dataset, as depicted in Fig. 5. The values differ due to the difference in dataset sizes and sampled queries in each region.

5.2 Effect of varying the distance threshold

Figure 6a depicts the results of varying the distance threshold (δ) for the T-Drive dataset. Here, δ denotes the additional percentage distance w.r.t. the shortest path distance. It can be observed that as δ increases, the number of popular paths (NPP) increases for all the queries. This occurs because with an increase in δ, more paths are available in the candidate set χ, and there is a chance for a path to be popular. Hence, NPP also increases. Since the spatial density of Q1's region is more than Q2's region and Q3's region, more paths are generated in Q1 than in other two queries. Hence, NPP value for Q1 is significantly high as compared to NPP values of Q2 and Q3. Observe that for Q1 (which is a query in a dense region), 36 popular paths could be obtained even by adding a small δ (say 10%) in the T-Drive dataset. Similarly, for Q2 and Q3, it is possible to obtain 22 and 6 popular paths, respectively, with a small δ of 10% in the T-Drive dataset. *The results are very encouraging because they indicate that users need not travel significant additional distance to obtain popular paths.* We believe that traversing a small additional distance is a small price to realize the several potential benefits provided by popular paths. We observe the similar results in Fig. 7a for Geolife dataset. The values differ due to the difference in dataset sizes and sampled queries.

We have conducted experiments by employing both proposed and online approaches by computing popular paths for Q1, Q2 and Q3. In online approach, we do not employ frequent patterns for computing the popularity score. The performance of the proposed approach for queries Q1, Q2 and Q3 is depicted with the notations Q1, Q2 and Q3 and for the same queries, the corresponding performance with the online approach is depicted with the notations $Q1^{\#}$, $Q2^{\#}$, and $Q3^{\#}$. It can be observed that as δ increases, the results in Fig. 6b indicate that the ET increases both for the proposed approach and the online approach. This occurs because with



(a) NPP (b) ET

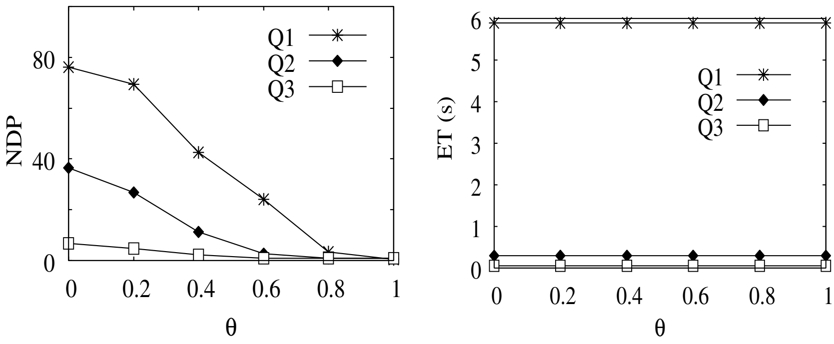
Fig. 7 Effect of variations in the distance threshold in Geolife dataset

an increase in δ, more paths are generated, leading to more ET. For the proposed approach, as δ increases, the execution time (ET) to compute NPP also varies among Q1, Q2, and Q3.

The ET value to extract NPP for Q1 increases exponentially with δ due to a significant increase in the number of paths as the query is in dense region. For Q2 and Q3, as there are queries in sparse regions, ET increases gradually compared to Q1 due to a lower number of popular paths. A similar trend is observed in $Q1^{\#}$, $Q2^{\#}$, and $Q3^{\#}$ as the number of paths computed remains the same. It can be observed that the proposed approach improves the performance significantly over the online approach as we extract frequencies of each combination of edges in an offline manner using pattern mining techniques. Furthermore, we employ a hashmap to *efficiently* search the patterns. On the other hand, in the online approach, the database of *UTT*s needs to be scanned to extract the frequency of combinations of each path in an online manner, which is computationally prohibitively intensive. As a result, the proposed approach exhibits significant performance improvement and returns the popular paths within only a few seconds. Thus, our proposed approach can be used as a foundation for building near real-time path discovery services. Notably, the online reference approach's time complexity is in the order of the transaction size and the number of edges in the path. We observe a similar trend in the results for the Geolife dataset, as depicted in Fig. 9b. The values differ due to the difference in dataset sizes and sampled queries.

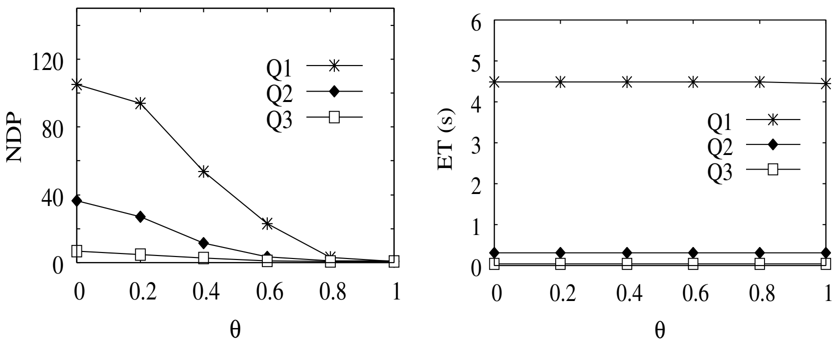
5.3 Effect of varying the diversity threshold

Figure 8 depicts the results of varying the diversity threshold for the T-Drive dataset. $\theta = 0$ implies that all the paths, irrespective of their diversity constraint, can be a part of the DPP result list. $\theta = 1$ implies that only those paths, which are entirely different (i.e., diverse), can contribute to the DPP result list. As θ increases, the results in Fig. 8a indicates that the number of diverse popular paths (NDP) decreases for all the queries. This occurs because with an increase in θ, fewer are included in the diverse popular list, and it reaches a saturation point (i.e., the point at which only



(a) NDP (b) ET

Fig. 8 Effect of variations in the diversity threshold in T-Drive dataset



(a) NDP (b) ET

Fig. 9 Effect of variations in the diversity threshold in Geolife dataset

one path in the candidate set is in the diverse set). For some of the queries, the saturation point reaches earlier because all of the extracted paths are not diverse w.r.t. each other. More paths are generated in the dense region; due to this, the list DPP also increases for queries with high spatial densities.

Figure 8b shows that the execution time (ET) for computing the top-k DPP for Q1 is high as compared to Q2 and Q3. This is because more popular paths are generated in Q1 i.e., more paths to be processed to get the DPP list. As θ increases, the ET remains comparable for a query because most of the time in computing DPP goes into computing the set of paths; this depends on δ, which is constant in this scenario.

We observe a similar trend in the results for the Geolife dataset, as depicted in Fig. 9. The values differ due to the difference in dataset sizes and sampled queries.

5.4 Comparison of shortest path rank and popularity rank

Tables 4 and 5 depict the comparison of shortest path rank and popularity rank and vice versa of all three queries for the top-10 results. Here, we randomly select

Table 4 Illustrative comparison of shortest path rank and popularity rank for Geolife Dataset

Q1				Q2				Q3			
T10-SP	PR	T10-PP	SPR	T10-SP	PR	T10-PP	SPR	T10-SP	PR	T10-PP	SPR
1	3	1	5	1	6	1	9	1	8	1	5
2	2	2	4	2	8	2	10	2	9	2	2
3	7	3	3	3	1	3	4	3	5	3	8
4	8	4	2	4	4	4	2	4	1	4	10
5	6	5	10	5	9	5	1	5	7	5	7
6	1	6	1	6	10	6	8	6	2	6	1
7	4	7	6	7	5	7	5	7	4	7	4
8	5	8	8	8	2	8	6	8	10	8	6
9	10	9	9	9	7	9	3	9	3	9	3
10	9	10	7	10	3	10	7	10	6	10	9

T1O-SP: Rank of 10 paths as per the distance (shortest path receives the first rank); *PR*: Popularity Rank; *T1O-PP*: Rank of 10 paths as per the popularity score (the path with highest popularity receives the first rank); *SPR*: Shortest Path's Rank

Table 5 Illustrative comparison of shortest path rank and popularity rank for T-Drive Dataset

Q1				Q2				Q3			
T10-SP	PR	T10-PP	SPR	T10-SP	PR	T10-PP	SPR	T10-SP	PR	T10-PP	SPR
1	7	1	6	1	7	1	6	1	3	1	6
2	3	2	7	2	5	2	2	2	2	2	2
3	10	3	2	3	4	3	1	3	5	3	1
4	6	4	3	4	3	4	4	4	8	4	4
5	9	5	5	5	2	5	3	5	4	5	7
6	2	6	1	6	1	6	10	6	9	6	10
7	5	7	10	7	8	7	9	7	6	7	3
8	1	8	9	8	10	8	7	8	1	8	8
9	8	9	8	9	6	9	5	9	10	9	9
10	4	10	4	10	9	10	8	10	7	10	5

T1O-SP: Rank of 10 paths as per the distance (shortest path receives the first rank); *PR*: Popularity Rank; *T1O-PP*: Rank of 10 paths as per the popularity score (the path with highest popularity receives the first rank); *SPR*: Shortest Path's Rank

one query from the set of five representative queries in each region. We present the list of top-10 shortest paths and their corresponding popularity rank and the list of top-10 popular paths and their ordering based on path length for all three queries. For each query, observe that the path having the highest shortest path rank (SPR) (i.e., shortest path between source and destination) may not be the path with the highest popularity. Conversely, the paths with higher popularity score may not have the highest value of SPR. Hence, the shortest path might not be the most popular path and vice versa. Hence, the ordering of paths based on popularity score indicates a different kind of knowledge than the ordering based on distance.

6 Conclusion

In this paper, we have addressed the problem of ranking the input paths between a given source and destination pair based on the popularity of the paths. We have proposed a model to compute the popularity score of a given path by combining the effect of the number of user traversals and the number of edges of the path covered by each user traversal. We have also presented an efficient approach for computing the popularity score of a given path by modeling user traversals as transactions and deploying pattern mining techniques. Our performance study with two real datasets demonstrates the effectiveness of the proposed scheme. In particular, the results of our performance evaluation indicate that by traversing a relatively small additional distance, we can obtain more number of popular paths.

We believe that the proposed approach would be beneficial in designing popular path discovery services as a complement to services for finding only the shortest paths. We plan to investigate the issues in the deployment of the proposed approach in designing popular path discovery services as a complement to services for finding only the shortest paths.

References

1. Aggarwal, C.C., Han, J. (eds.): Frequent Pattern Mining. Springer (2014)
2. Chang, K.P., Wei, L.Y., Yeh, M.Y., Peng, W.C.: Discovering personalized routes from trajectories. In: Proceedings of the International Workshop on Location-Based Social Networks, pp. 33–40. ACM (2011)
3. Chen, Z., Shen, H.T., Zhou, X.: Discovering popular routes from trajectories. In: Proceedings of the International Conference on Data Engineering, pp. 900–911. IEEE (2011)
4. Chondrogiannis, T., Bouros, P., Gamper, J., Leser, U.: Alternative routing: *k*-shortest paths with limited overlap. In: Proceedings of the International Conference on Advances in Geographic Information Systems, p. 68. ACM (2015)
5. Chondrogiannis, T., Bouros, P., Gamper, J., Leser, U.: Exact and approximate algorithms for finding *k*-shortest paths with limited overlap. In: Proceedings of the International Conference on Extending Database Technology, pp. 414–425 (2017)
6. Chondrogiannis, T., Gamper, J.: ParDiSP: A partition-based framework for distance and shortest path queries on road networks. In: Proceedings of the International Conference on Mobile Data Management, vol. 1, pp. 242–251. IEEE (2016)
7. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (1959)
8. Feng, Z., Zhu, Y.: A survey on trajectory data mining: techniques and applications. IEEE Access **4**, 2056–2067 (2016)
9. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: ACM SIGMOD Record, 2, pp. 1–12. ACM (2000)
10. He, Z., Chen, K., Chen, X.: A collaborative method for route discovery using taxi drivers experience and preferences. IEEE Trans. Intell. Transp. Syst. **19**(8), 2505–2514 (2017)
11. Hendawi, A.M., Rustum, A., Ahmadain, A.A., Hazel, D., Teredesai, A., Oliver, D., Ali, M., Stankovic, J.A.: Smart personalized routing for smart cities. In: Proceedings of the International Conference on Data Engineering, pp. 1295–1306. IEEE (2017)

12. Hershberger, J., Maxel, M., Suri, S.: Finding the k shortest simple paths: a new algorithm and its implementation. ACM Trans. Algorithms **3**(4), 45 (2007)
13. Hu, G., Shao, J., Ni, Z., Zhang, D.: A graph based method for constructing popular routes with check-ins. World Wide Web **21**(6), 1689–1703 (2018)
14. Koide, S., Tadokoro, Y., Yoshimura, T., Xiao, C., Ishikawa, Y.: Enhanced indexing and querying of trajectories in road networks via string algorithms. ACM Trans. Spatial Algorithms Syst. **4**(1), 1–41 (2018)
15. Letchner, J., Krumm, J., Horvitz, E.: Trip router with individualized preferences (trip): incorporating personalization into route planning. In: Proceedings of the National Conference on Artificial Intelligence and the Innovative Applications of Artificial Intelligence Conference, pp. 1795–1800. AAAI Press (2006)
16. Li, X., Han, J., Lee, J.G., Gonzalez, H.: Traffic density-based discovery of hot routes in road networks. In: Proceedings of the International Symposium on Spatial and Temporal Databases, pp. 441–459. Springer (2007)
17. Liu, H., Jin, C., Yang, B., Zhou, A.: Finding top-k shortest paths with diversity. IEEE Trans. Knowl. Data Eng. **30**(3), 488–502 (2018)
18. Lo, C.L., Chen, C.H., Hu, J.L., Lo, K.R., Cho, H.J.: A fuel-efficient route plan method based on game theory. J. Internet Technol. **20**(3), 925–932 (2019)
19. Martins, E.Q., Pascoal, M.M.: A new implementation of Yens ranking loopless paths algorithm. Q. J. Belgian French Ital. Operat. Res. Soc. **1**(2), 121–133 (2003)
20. Potamias, M., Bonchi, F., Castillo, C., Gionis, A.: Fast shortest path distance estimation in large networks. In: Proceedings of the ACM Conference on Information and Knowledge Management, pp. 867–876. ACM (2009)
21. Rathan, P.R., Reddy, P.K., Mondal, A.: Discovering diverse popular paths using transactional modeling and pattern mining. In: Proceedings of the International Conference on Database and Expert Systems Applications, pp. 327–337. Springer (2019)
22. Sacharidis, D., Bouros, P., Chondrogiannis, T.: Finding the most preferred path. In: Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 1–10 (2017)
23. Sommer, C.: Shortest-path queries in static networks. ACM Comput. Surv. **46**(4), 1–31 (2014)
24. Wang, Z., Che, O., Chen, L., Lim, A.: An efficient shortest path computation system for real road networks. In: Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, pp. 711–720. Springer (2006)
25. Wei, L.Y., Chang, K.P., Peng, W.C.: Discovering pattern-aware routes from trajectories. Distrib. Parallel Databases **33**(2), 201–226 (2015)
26. Wei, L.Y., Zheng, Y., Peng, W.C.: Constructing popular routes from uncertain trajectories. In: Proceedings of the International Conference on Knowledge Discovery and Data Mining, pp. 195–203. ACM (2012)
27. Yen, J.Y.: Finding the k shortest loopless paths in a network. Manag. Sci. **17**(11), 712–716 (1971)
28. 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 the International Conference on Advances in Geographic Information Systems, pp. 99–108. ACM (2010)
29. Zheng, Y., Xie, X., Ma, W.Y.: Geolife: a collaborative social networking service among user, location and trajectory. IEEE Data Eng. Bull. **33**(2), 32–39 (2010)
30. Zheng, Y., Zhang, L., Xie, X., Ma, W.Y.: Mining interesting locations and travel sequences from GPS trajectories. In: Proceedings of the International Conference on World Wide Web, pp. 791–800. ACM (2009)