Efficient Trip Planning for Maximizing User Satisfaction

Chenghao Zhu[1], Jiajie Xu[1,3](✉), Chengfei Liu[2], Pengpeng Zhao[1,3], An Liu[1,3], and Lei Zhao[1,3]

[1] School of Computer Science and Technology, Soochow University, Suzhou, China
{xujj,ppzhao,anliu,zhaol}@suda.edu.cn
[2] Faculty of ICT, Swinburne University of Technology, Melbourne, Australia
cliu@swin.edu.au
[3] Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, China

Abstract. Trip planning is a useful technique that can find various applications in Location-Based Service systems. Though a lot of trip planning methods have been proposed, few of them have considered the possible constraints of POI sites in required types to be covered for user intended activities. In this paper, we study the problem of multiple-criterion-based trip search on categorical POI sites, to return users the trip that can maximize user satisfaction score within a given distance or travel time threshold. To address this problem, we propose a spatial sketch-based approximate algorithm, which extracts useful global information based on spatial clusters to guide effective trip search. The efficiency of query processing can be fully guaranteed because of the superior pruning effect on larger granularity. Experimental results on real dataset demonstrate the effectiveness of the proposed methods.

1 Introduction

Location-Based Services (LBS) are one of the most frequently used tools in our life nowadays, and most of them rely on the trip planning techniques [1–4] to provide traffic related services. People can easily find the routes they need by accessing Google Map or Microsoft MapPoint through their personal computers or smart phones. Some systems are capable of suggesting users the trips that can cover their intended activities. Obviously, the trip planning algorithms play a key role to improve the quality and efficiency of our travel, and a lot of work [3,5–11] has been done to return users the rational trips and routes.

Given that people usually seek to find trips according to their intended activities, this paper investigates the problem of multiple-criterion-based trip search on categorical POI sites. Consider the example in Figure 1 where all POI sites are distributed on a road network, and each of them belongs to a type (e.g., a restaurant or a cafe) and has a satisfaction score marked by previous guests. A tourist issues a query to find a route from location B to location E via a

M. Renz et al. (Eds.): DASFAA 2015, Part I, LNCS 9049, pp. 260–276, 2015.
DOI: 10.1007/978-3-319-18120-2_16

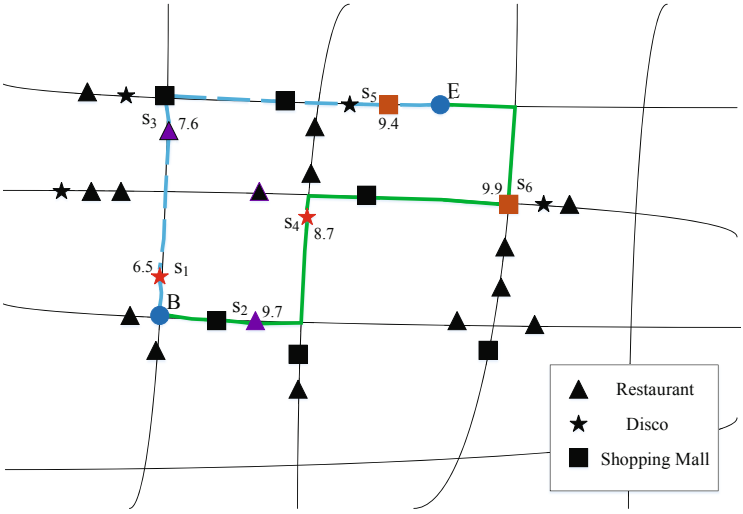


Fig. 1. Trip Search on Categorical POI Sites

cinema, a shopping mall and a restaurant within a distance threshold. Here, most of the route search methods [5,6] cannot be applied because they require a POI site in each type to be selected before the trip search; The categorical trip search methods proposed in [7,9] are able to return a shortest trip (i.e., Tr_1 in the dashed line), but it only supports single-criterion-based trip search, while in reality users would prefer to choose such a trip (i.e., Tr_2 in the full line) that can maximize the satisfaction of their intended activities within the given distance threshold.

Motivated by the above example, this paper studies the problem of multiple-criterion-based categorical trip search, which has high computational overhead for two main reasons: firstly, it requires to select some POI sites from database to meet the activity requirements from users, and the combinations between categorical POI sites are obviously high; secondly, the trip search requires to compare the scalable candidate trips that pass all selected POI sites, and the validation of each candidate trip often relies on a large number of shortest path queries on road network. As far as we know, no existing work can well address all above challenges to find the optimal trip in Figure 1 efficiently.

In this paper, we present a spatial sketch-based approximate algorithm to cope with the challenges mentioned above. It utilizes the spatial clustering to manage the scalable POI sites in larger granularity, then explores the sketched information in a global point of view to understand the spatial relations between clusters, as well as the overall execution plan to avoid unnecessary shortest path queries based on the cost analysis. Finally the extracted information is used to guide effective trip search. Based on a set of theoretical bounds and search space reduction techniques, trip search can be processed efficiently because of the much superior pruning effects over the required dimensions. In summary, the main contributions of this paper can be summarized as follows:

- We define a problem called efficient trip planning for maximizing user satisfaction, which could find applications in various LBS systems.
- We propose the satisfaction-score-first trip planning algorithm as the baseline algorithm to return the exact result.
- We propose the spatial sketch-based approximate trip planning algorithm to achieve a high performance with a little loss of accuracy.
- We implement the proposed algorithms and conduct experiments on real datasets to evaluate the performances of our proposed solutions.

The remainder of this paper is organized as follows: We firstly introduce the related work in Section 2 and then define the problem in Section 3. Afterwards, an exact solution and an approximate solution are presented in Section 4 and Section 5 respectively. We carry out experimental analysis to evaluate and compare the proposed methods in Section 6. Finally, we conclude this paper in Section 7.

2 Related Work

Trip planning is a classical research problem that has been investigated for many decades, and existing efforts can be generally classified into two main categories: the point-based trip search and the categorical trip search.

The point-based trip search aims to find a good path that can pass some user specified locations over the spatial network, and the evaluation of the trips can be measured by a single factor [5,6] or multiple factors [12,13], e.g., distance, expense or time. Some indexing structures such as arterial hierarchy [14], contraction hierarchies [15], path oracle [16], transit-node routing [17], detour based [18] as well as some trajectory based methods [19] are developed to speed up the searching process by the shortcuts embedded in the indexing structure. However users tend to have some intended activities in the trip (e.g., to have dinner) in some cases, the trip returned by those methods cannot be ensured to cover the POI sites in required types (e.g., restaurant).

To address the above problem, some effort [3,7–9] has been made in recent years toward the categorical trip search, i.e., to find the good paths with all required categorical POI sites to be covered. Specifically, Li et al. proposed a trip planning query (TPQ) in [3,7] to find the shortest path through at least one POI site of each required type. Later, Sharifzadeh et al. [9] defined and solved a variant problem of TPQ that additionally ensures the POI sites appear in a correct order consistent with user specification. However, those methods mainly take single criterion into account, while the users tend to have multiple requirements (e.g., have distance threshold and prefer total satisfaction score to be high). More recently, some work [20–23] was done to recommend trips to users under different criteria based on the historical trajectory data. An assumption was made that there exist one or more trajectories in database that can meet the requirement of each user query. Unfortunately, this assumption is not realistic in many cases especially we need to control the size of trajectory data for querying efficiency concerns.

In this paper, we investigate the problem of multiple-criterion-based trip search over categorical POI sites. Compared to TPQ and its variants, our problem requires to balance different factors and thus incurs much greater complexity in query processing. Therefore, our methods need to adopt the spatial clustering techniques and explore useful knowledge in a global view to prune the search space in greater granularity.

3 Problem Definition

In this section, we formalize the problem of this paper based on some notions defined as follows.

Definition 1. *(Road Network) A spatial network is modeled as a connected and undirected graph $G = (V, E)$, where V is a set of vertices and E is a set of edges. A vertex $v_i \in V$ indicates a road intersection or an end of a road. An edge $e \in E$ is defined as a pair of vertices and represents a segment connecting the two constituent vertices.*

For example, (v_i, v_j) represents a road segment that enables travel between vertices v_i and v_j. We use $|v_i, v_j|$ to denote the shortest path between vertices v_i and v_j, and their network distance $||v_i, v_j||$ is the accumulated length of all edges on $|v_i, v_j|$, i.e., $||v_i, v_j|| = \Sigma_{e_k \in |v_i, v_j|}\ length(e_k)$.

Definition 2. *(Site) A site is a POI where users can take some activities, and it is stored in the form of $(loc, sc, type)$. Given a site s, $s.loc$ denotes the geographical location of the site; $s.sc$ is the satisfaction score of site s, which is computed as the average of all scores marked by historical guests, and thus implies the overall satisfaction level of guest experience; $s.type$ specifies the type of site s, e.g., $'shopping\ mall'$ and $'restaurant'$, based on a pre-defined type vocabulary.*

The location of each site can be mapped onto a given road network by map-matching algorithms. In this paper, we assume all sites have already been assigned to a vertex on the road network (i.e., $s.loc \in V$) for the sake of clear description. Also, we assume that each site belongs to only one type.

Obviously, users desire to obtain a trip (ordered) as the result returned by the system rather than a collection of sites (unordered), so a concept of our trip is necessary to be defined.

Definition 3. *(Trip) We use $Tr = \{B, S, E\}$ to denote our trip, which is from beginning point B to ending point E via a sequence of sites S (Duplicate site type is not allowed in S). Between any two adjacent sites $s_i, s_j \in S$, the trip always follows the shortest path connecting s_i and s_j. Therefore, the distance of a trip Tr can be calculated as*

$$||Tr|| = ||B, f_s|| + \sum_{s_i, s_j \in S} ||s_i, s_j|| + ||l_s, E|| \tag{1}$$

where f_s and l_s denote the first and the last sites in sequence S respectively. Regarding to user experience, we evaluate satisfaction score of a trip Tr as the sum of satisfaction scores of all sites in its site sequence S such that

$$score(Tr) = \sum_{s_i \in Tr.S} s_i.sc. \tag{2}$$

Definition 4. *(Query) Given a spatial network G, a collection of sites and a type vocabulary, we can define our query as $Q = \{B, E, Max_d, T\}$, where B and E specify the beginning point and the ending point respectively; Max_d is the threshold of the trip distance; and T is the subset of type vocabulary, which contains the activity types that the user desires to access.*

Definition 5. *(Candidate Trip) Given a query Q, a trip Tr is said to be a candidate trip if it satisfies the following conditions:*

(1) It starts at beginning point $Q.B$ and ends up at ending point $Q.E$;

(2) Every type in $Q.T$ corresponds to only one site in $Tr.S$, i.e., $Q.T = \{s_i.type | s_i \in Tr.S\}$;

(3) The distance of the trip is within the given distance threshold $Q.Max_d$, i.e., $||Tr|| \leq Q.Max_d$.

Problem Definition. Given a query Q, the road network G and a set of sites, from the set of all candidate trips $\mathbb{S}_{Tr}$, this paper intends to find and return the candidate trip $Tr \in \mathbb{S}_{Tr}$ that has the maximum value of satisfaction score, such that $\forall Tr' \in \mathbb{S}_{Tr}$, $score(Tr') \leq score(Tr)$.

4 Satisfaction-Score-First Trip Planning Algorithm

In this section, we propose a baseline algorithm, the satisfaction-score-first trip planning algorithm (S^2F) to process the user query. It adopts the best first method and returns the exact result. The main idea of S^2F is to verify the candidate sites (covering all required types) by the descending order of total satisfaction score. For each *candidate sites* as a group, we compute the candidate trips that pass all those sites, and a trip is returned if its distance is less than or equal to the given threshold. For the convenience of description, we define the notion candidate sites ($\mathbb{C}_s$) first.

Definition 6. *(Candidate Sites) Given a query Q, a set of sites $\mathbb{C}_s$ is called candidate sites if they can satisfy the following two conditions: (1) It covers all the required types in $Q.T$, i.e., $Q.T \subseteqq \{s_i.type | s_i \in \mathbb{C}_s\}$; (2) If any site is removed from $\mathbb{C}_s$, the above condition would fail. The satisfaction score of $\mathbb{C}_s$ is measured as the total score of all sites in $\mathbb{C}_s$ such that $score(\mathbb{C}_s) = \sum_{s_i \in \mathbb{C}_s} s_i.sc.$*

We maintain some necessary data structures to support query processing. For each type in $Q.T$ (e.g., $'restaurant'$), we utilize a list in the form of $< siteId, score >$ to store all sites in this type, sorted by their value of score. Then a bi-dimensional list STL is used to cover all types, e.g., $STL[i][0]$ indicates the

Algorithm 1. Satisfaction-Score-First Trip Planning Algorithm

Input: a query Q, the inverted list STL
Output: the optimal planned trip Tr

```
1: Tr ← ∅;
2: PQ_cs ← {STL[i][0] | t_i ∈ Q.T};
3: while true do
4:     candidate sites C_s ← PQ_cs.poll();
5:     candidate trips S_Tr ← deriveCandidateTrips(C_s);
6:     for each candidate trip Tr' ∈ S_Tr do
7:         if ||Tr'|| ≤ Q.Max_d then
8:             Tr ← Tr';
9:             break;
10:        else
11:            PQ_cs = UpdatePriorityQueue(PQ_cs, STL);
12:        end if
13:    end for
14: end while
15: return Tr;
```

first record in the list belonged to type t_i (i.e., the site that has the highest satisfaction score among all sites in type t_i). Based on STL, a priority queue PQ_{cs} is used in implementation to dynamically maintain a list of $\mathbb{C}_s$ for access in the descending order of their satisfaction scores.

Algorithm 1 shows how the S^2F algorithm works. At first, the $\mathbb{C}_s$ that has maximum satisfaction score in each required type is inserted to the priority queue PQ_{cs} (Line 2). Then we keep verifying all the $\mathbb{C}_s$ (Lines 3-14): for each round, we fetch the top record from PQ_{cs} to get the candidate sites $\mathbb{C}_s$ with highest satisfaction score (Line 4), and then verify it regarding to the candidate trips (Lines 5-13). If a valid trip is found, then we simply return this trip to the user (Lines 7-9); otherwise, we update the priority queue PQ_{cs} to guarantee the next best candidate sites appear in PQ_{cs} (Line 11). This procedure repeats until the optimal result can be found.

Given a candidate sites $\mathbb{C}_s$, it corresponds to a number of $|\mathbb{C}_s|!$ candidate trips (via sites in different sequences), so it may require to verify all of the candidate trips unless a valid trip can be found. By $Q.E$ is the intermediate destination in the way back to $Q.B$, it can be easily reduced to the famous Traveling Salesman Problem in NP Complete complexity. Fortunately, in the real applications, the number of required types tends to be relatively small, such that we can normally process it in an acceptable time. However, for those extreme cases with a large value of $|\mathbb{C}_s|$, we adopt a greedy method to iteratively choose the closest unvisited site in $\mathbb{C}_s$ to check the promising candidate trips with more priority, so as to stop early and thus accelerate the query processing.

In overall, the best first methods of the satisfaction-score-first trip planning algorithm contribute to reach the optimal trip earlier than the search by random. But obviously, it is potentially engaged in traversing all candidate trips as

well. In such cases, the tremendous combinations would cause the algorithm to be extremely slow especially when the number of sites goes up.

5 Spatial Sketch-Based Approximate Trip Planning Algorithm

In this section, we further introduce a spatial sketch-based approximate trip planning algorithm (S^2A in short) to support efficient trip search. We first overview the structure of S^2A and then describe the work mechanism in detail.

5.1 Overview

In the S^2A algorithm, we process the trip search by three main steps as shown in Figure 2. The first step is the preprocessing step, in which the sites are clustered to spatial clusters, and the useful inter and intra cluster information are extracted as well. Figure 2 (a) illustrates the preprocessing step.

Afterwards, the second step is to search on the granularity of clusters, without touching at the site level. As shown in Figure 2 (b), we first discuss the priority of the candidate clusters to be checked. Then we consider the issue of sequence of the candidate clusters to go through (sketched trips in Figure 2 (b)), so that the hopeless and unpromising ones can be filtered out. Furthermore, we analyze the contents of the sketched trips to derive the execution plan based on the cost analysis of the required shortest path search, which contributes to avoid vast unnecessary computational cost in trip search.

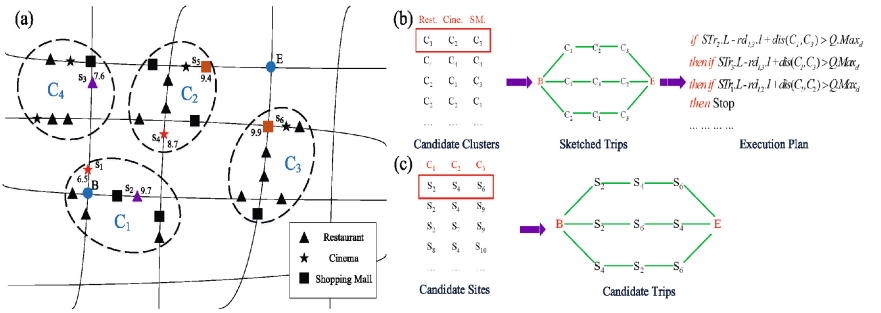


Fig. 2. An example of S^2A

At last, the trip search goes to the site level as Figure 2 (c) shows. Given a candidate clusters, we scan the corresponding candidate sites based on the execution plan derived from the previous step, and some bounds are used to guarantee the effectiveness and efficiency of trip search in S^2A.

5.2 Preprocessing

As the amount of sites is large, it is not realistic to process the trip search on all sites directly. In the S^2A algorithm, we carry out the spatial clustering on sites first, so that it is possible to maintain a global picture to prune on larger granularity and to guide smarter trip search.

Towards the spatial clustering, classical algorithms can be generally classified into two categories, namely density-based method and partitioning-based method. As illustrated in Figure 3, for algorithms in density-based method like DBSCAN, they tend to find clusters with high spatial coherence (circles in the full line), but nevertheless, they do not guarantee all sites to be included into the clusters because of its density requirements, possibly leading good sites unable to be represented in clusters. In contrast, the partition-based approaches like k-means or k-medoids are capable of ensuring every site to appear in a cluster, while the clusters (circles in the dashed line) tend to be not compact enough.

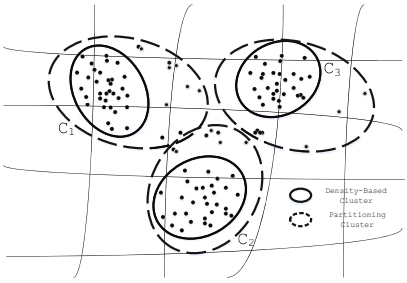


Fig. 3. Clustering By Classic Algorithms

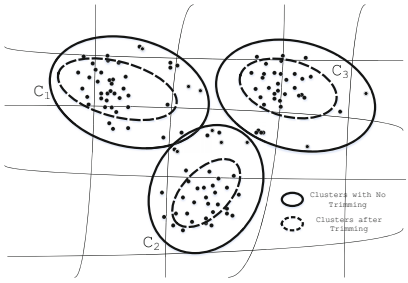


Fig. 4. Clustering By the PTC

To address above problem, we present the partition-trimming clustering algorithm (PTC) to return the spatially coherent clusters like those in Figure 4. Specifically, the sites are clustered by the partitioning-based approach first, so all sites are covered by the clusters (circles in the full line). But the initial clusters are possibly too sparse, so we further trim the clusters in the second step: for each cluster C, we continuously remove the sites with satisfaction scores lower than that of an interior site (that has less distance to center of C) in the same type; other sites (circles in the dashed line) are reserved because they are likely to appear in the trip required by users. In this way, it achieves a good balance between the spatial coherence and good site coverage.

Based on the spatial clusters, we extract and store some necessary information to assist the pruning in trip search. Firstly, we compute and store the spatial relations of clusters, e.g., the minimum and the maximum possible distances between any two clusters, and the maximum distance between sites within a cluster, etc. Two variables $Cmin_{i,j}$ and $Cmax_{i,j}$ are used to represent the minimum and the

maximum distance from cluster C_i to C_j respectively. Secondly, we extract the categorical information of each cluster, e.g., the types in the cluster, so we can have an overview of the cluster to help cluster selection.

5.3 Sketched Trip Search on Clusters

Based on the spatial clusters, we analyze the information extracted not only to prune the search space in a greater granularity, but also to find out guidelines of how to carry out trip search on scalable sites. Given a query Q, we commence with the *candidate clusters*, e.g., in Figure 2 (b) (C_1, C_2, C_3) if the user desires to get a trip via a restaurant, a cinema and a shopping mall. We formally give the definition of candidate clusters ($\mathbb{C}_c$) as follows:

Definition 7. *(Candidate Clusters) Given a query Q, a collection of clusters $\mathbb{C}_c$ is called candidate clusters if it can satisfy the following two conditions: (1) Each type in $Q.T$ corresponds to a cluster in $\mathbb{C}_c$, i.e., $\forall t_i \in Q.T$, $\exists C_j \in \mathbb{C}_c$, and $\exists s_k \in C_j$, such that $s_k.type = t_i$; (2) It's a one-to-one correspondence, i.e., each type in $Q.T$ is only bound with one cluster in $\mathbb{C}_c$. The satisfaction score of $\mathbb{C}_c$ is computed as:*

$$Score(\mathbb{C}_c) = \sum_{C_i \in \mathbb{C}_c} Max(s_j.sc | s_j \in C_i, s_j.type = C_i.type); \tag{3}$$

where $C_i.type$ implies that C_i has to provide a site in $C_i.type$.

As our goal is to maximize the satisfaction score of the trip, those $\mathbb{C}_c$ with higher satisfaction scores have greater priority because they cover better trip candidates. Similar to S^2F, we use a priority queue to retrieve $\mathbb{C}_c$ in the descending order of satisfaction scores. For example, in Figure 2 (b), (C_1, C_2, C_3) has the greatest score value, so we fetch it to verify its feasibility first.

Given a $\mathbb{C}_c$ to verify, we devote to return a group of sketched trips from beginning point $Q.B$ to ending point $Q.E$ via the clusters in some sequences, e.g., in Figure 2 (b) $< B, C_1, C_2, C_3, E >$, $< B, C_1, C_3, C_2, E >$ and $< B, C_2, C_1, C_3, E >$. Defined as follows, a sketched trip in cluster level corresponds to a number of candidate trips in site level, through which we can implement the pruning on a larger granularity perspective and markedly improve the search efficiency.

Definition 8. *(Sketched Trip) A sketched trip STr is modeled as $STr = \{B, S_c, E\}$, where B and E are the beginning point and the ending point respectively; the S_c denotes a set of clusters with a sequence that appears in the trip (i.e., that has temporal order to visit), for example $S_c = < C_1, C_2, C_3 >$ in Figure 2 (b).*

In any two concessive clusters C_i and C_j in a sketched trip STr, the actual travel distance between the two clusters is in the range $[Cmin_{i,j}, Cmax_{i,j}]$ (can be derived from the matrix extracted in pre-processing step), and the distance of a sketched trip STr is thus in the range $[STr.L, STr.U]$ such that

$$STr.L = Cmin_{B,f_c} + \sum_{C_i, C_j \in S_c} Cmin_{i,j} + Cmin_{l_c,E} \tag{4}$$

$$STr.U = Cmax_{B,f_c} + \sum_{C_i,C_j \in S_c} Cmax_{i,j} + Cmax_{l_c,E} \tag{5}$$

where f_c and l_c denote the first and the last clusters in the cluster sequence S_c respectively.

As a $\mathbb{C}_c$ can generate a large number ($|\mathbb{C}_c|!$) of sketched trips (via clusters in different sequences), it would be time-consuming if we scan all of them when the value $|\mathbb{C}_c|$ is high. Therefore it is not realistic to consider all sketched trips, and we seek to select out the top-k ones most likely to fulfill the distance requirement of the query for consideration. The problem is how to measure the possibility $P(STr)$ of an arbitrary sketched trip STr for satisfying the distance constraint. For a given distance threshold $Q.Max_d$, there are three **situations** of a sketched trip STr: for **situation 1** where $Q.Max_d < STr.L$, obviously STr is hopeless to meet the distance threshold ($P(STr) = 0$), so it is filtered; while in **situation 2** where $Q.Max_d \geq STr.U$, it can be guaranteed to satisfy the distance constraint ($P(STr) = 1$), and all trips covered by STr are valid trips; for the **situation 3** where $STr.L \leq Q.Max_d < STr.U$ holds, STr is said to be potentially valid, so we measure the valid possibility to be:

$$P(STr) = \frac{Q.Max_d - STr.L}{STr.U - STr.L} \tag{6}$$

where $STr.U - STr.L$ confines the range of possible distances of a trip covered by STr, and among the whole distance range, $Q.Max_d - STr.L$ confines the part of the range that is valid in terms of the distance constraint of query Q. Their ratio can thus represent the possibility of a trip (that can be sketched to STr) to be valid. Therefore given a $\mathbb{C}_c$, we only consider a sketched trip in situation 2 if it exists, or consider the top-k sketched trips regarding to their valid possibilities.

Based on the sketched trips to be considered, we further find out effective execution plan that requires less shortest path querying overhead based on our cost analysis model. For each sketched trip, we need to use a number of $|STr.S_c| + 1$ times of shortest path search (to calculate the distance between two sites in adjacent clusters, including the beginning/ending point), and we need to process all sketched trips of the candidate clusters in the worst case. Different cluster pairs should have varying priorities because of the difference of their capability on pruning for two main factors:

- *Frequency.* If a cluster pair appears in more sketched trips, e.g., (C_1, C_3) in Figure 2 (b), thus a single shortest path search may help us to prune multiple trips (e.g., on $< B, C_1, C_3, C_2, E >$ and $< B, C_2, C_1, C_3, E >$);
- *Impact.* Different cluster pairs have varying impacts on the pruning effect of each sketched trip it belongs to.

Therefore in this paper, we measure the priority $\omega_{i,j}$ of a given cluster pair $< C_i, C_j >$ as Equation 7:

$$\omega_{i,j} = \sum_{STr \in \tau} \frac{Cmax_{i,j} - Cmin_{i,j}}{STr.U - STr.L} \tag{7}$$

where τ is the set of relevant sketched trips (top-k sketched trips in valid possibility), and $\frac{Cmax_{i,j}-Cmin_{i,j}}{STr.U-STr.L}$ denotes the pruning effect of the cluster pair on a sketched trip STr. As a result, $\omega_{i,j}$ denotes the total pruning effect on all sketched trips the cluster pair $< C_i, C_j >$ belongs to. Once we have these computing priorities, we can use them to organize the verification of sketched trips and to guide the trip search on sites.

The execution plan implements the verification of sketched trips by some rules. The rules are made by the computing priorities and some global variables kept to maintain the current minimum distances of sketched trips. For example, in Figure 2 (b), for $STr_2 = < B, C_1, C_3, C_2, E >$, at the beginning its current minimum distance is initialized to its minimum distance $STr_2.L$, if the priority of $< C_1, C_3 >$ is the highest, we update the current minimum distance by subtracting $Cmin_{1,3}$ and adding the actual distance between a site in C_1 to a site in C_3. If it exceeds the distance threshold $Q.Max_d$ already, we prune it directly.

To sum up, sketched trip search on clusters implements a larger granularity search and markedly improve the search efficiency. And the execution plan avoids some unnecessary shortest path queries in trip search on sites.

5.4 Trip Search On Sites

If the sketched trip search on clusters cannot prune all the sketched trips, we must go further to the trip search on sites. In this subsection, we present the trip search on sites in detail. Algorithm 2 shows how the trip search is executed.

Given a query Q, we generate a list of $\mathbb{C}_c$ based on all requested types in $Q.T$ first. A priority queue PQ_{cc} is used to organize all the $\mathbb{C}_c$ in the descending order of their satisfaction scores (Line 4). Then we scan PQ_{cc}, each time we fetch a $\mathbb{C}_c$ to do the sketched trip search on clusters (Lines 9-11). If no valid sketched trip is found, we go on processing the next $\mathbb{C}_c$. Otherwise, we carry out the trip search on sites on basis of the sketched trips (Lines 12-24).

Given a group of sketched trips, the first step towards the trip search on sites is that we generate all the $\mathbb{C}_s$ derived from that group of sketched trips. Similar to before, we also adopt a priority queue PQ_{cs} to organize all the $\mathbb{C}_s$ obtained in the descending order by their satisfaction scores (Line 13). For example, In Figure 2 (c), $< s_2, s_4, s_6 >$ is the champion in satisfaction score.

The advantage that the execution plan brings to this stage is that we can check multiple trips with a single shortest path query. For example, in Figure 2 (c), we would launch a shortest path query from s_2 to s_6 if the priority of $< C_1, C_3 >$ is the highest. If the result distance exceeds the threshold that the execution plan gave, we can prune the two trips ($< B, s_2, s_6, s_4, E >$ and $< B, s_4, s_2, s_6, E >$) by only one shortest path query.

Once we find a feasible trip, we cannot directly return it as the result, because there may exist other solutions from the unprocessed $\mathbb{C}_c$ that have higher satisfaction scores. As such, a global bound is defined, S_{LB}, to guarantee the correctness of

Algorithm 2. Spatial Sketch-Based Approximate Trip Planning Algorithm

Input: a query Q
Output: a solution trip Tr

```
1: Tr ← ∅;
2: S_LB ← 0;
3: Tr' ← ∅;
4: PQ_cc ← generateAllCandidateCluster();
5: while (C_c ← PQ_cc.poll()) != null do
6:     if Score(C_c) ≤ S_LB then
7:         break;
8:     end if
9:     STrList ← generateAllSTr(C_c);
10:    STrList.reduce();
11:    STrList.executionPlan();
12:    if STrList.isEmpty() == false then
13:        PQ_cs ← generateAllCandidateSite(STrList);
14:        while (C_s ← PQ_cs.poll()) != null do
15:            if Score(C_s) < S_LB then
16:                break;
17:            end if
18:            Tr' ← tripSearch(C_s, STrList);
19:            if Tr' != null then
20:                S_LB ← Score(Tr');
21:                Tr ← Tr';
22:                break;
23:            end if
24:        end while
25:    end if
26: end while
27: return Tr;
```

the trip search on sites. S_{LB} is designed to keep track of the best feasible solution's satisfaction score, which is computed as

$$S_{LB} = max_{Tr_i \in T_s}\{score(Tr_i)\} \tag{8}$$

where T_s is the set of feasible trips we have found. Obviously, S_{LB} is dynamically updated every time we find a new feasible trip, and it is the threshold in trip search processing: if the score of a trip falls below S_{LB} already, we need not continue to search as the remaining trips are hopeless to provide a higher score than S_{LB} (Lines 15-17). Since all the $\mathbb{C}_c$ are ranked in a descending order according to their satisfaction scores, **the entire search can be stopped** if a $\mathbb{C}_c$ can be found such that $Score(\mathbb{C}_c) \leq S_{LB}$ (Lines 6-8), because all of the trips derived from that $\mathbb{C}_c$ are hopeless to provide a higher score than S_{LB}.

6 Experimental Evaluation

In this section, extensive experiments are carried out on real spatial datasets to demonstrate the performance of the proposed algorithms. We use the real road network of Beijing, which contains 165,991 vertices and 225,999 edges respectively. Also, the POI dataset includes 100,000 Beijing POI sites that are classified into 11 categories, and each POI site is affiliated with a satisfaction score and mapped onto the road network. The number of POI sites in a categorical type varies between 5,835 to 13,639. Both algorithms are implemented in JAVA and tested on a DELL 7010MT (i5-3740) computer with 4-core CPUs at 3.2GHz and a 8GB RAM running Windows 7 (64 bits) operating system.

In the evaluations, we generate 100 representative sample queries based on the parameters shown in Table 1, and all results are derived by the average of the sample queries. Note that α is the parameter used to vary the distance threshold in a query, which is α times of the road network distance from beginning point to ending point via shortest path. We first compare the two proposed algorithms in the varying size of the POI sites, and then evaluate the effects of the varying parameters to the performance of the S^2A algorithm. Given that the algorithms are time-consuming for large dataset, we use a parameter ξ to evaluate the intermediate result at the ξ's iteration, so as to know the performance in the early iterations for scenarios requesting immediate response.

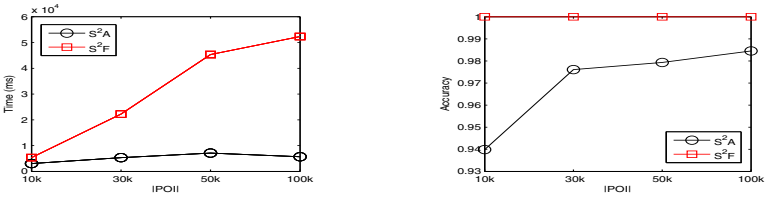
Table 1. Default Parameter Values

Parameter	Value	Description
k	4	Sketched trip reduction factor
t	4	Number of query site types
α	2	Shortest path distance multiplier
ξ	unlimited	Iteration threshold

Comparison of Proposed Algorithms. Figure 5 (a) and (b) show the efficiency and accuracy of the S^2F and S^2A algorithms in the varying size of the POI sites. From Figure 5 (a), it can be easily observed that the efficiency of the two algorithms are similar when the POI dataset is small. But when the POI dataset becomes large, the S^2A algorithm is much more efficient than S^2F, and this phenomenon can be explained by the much more superior pruning effect of S^2A (in the cluster granularity). In addition, the experimental results in Figure 5 (b) show that S^2A has fairly good accuracy, and particularly, the accuracy tends to increase when the size of POI sites goes up. The accuracy in small dataset cases is a lot worse than others, and this is is partly because of the top-k trips tend to be more skewed in the cases running on small POI dataset. Therefore the S^2A is a near optimal algorithm and much more efficient than S^2F.

Then we evaluate the impacts from parameters k, t and α to the performance of the S^2A algorithm. In each experiment, we compare the intermediate results (ξ = 10k, 30k, 50k candidate clusters to be checked) and the final result.

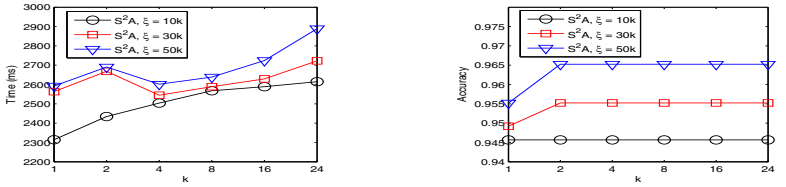
Effect of k. Figure 6 (a) and (b) show the efficiency and accuracy of the S^2A algorithm in the settings of k between 1 and 24 (as the default 4 types has



(a)Efficiency (b)Accuracy

Fig. 5. Performances of S^2F and S^2A

24 trip sequences at most). When the value of k is greater than 4, the S^2A algorithm tends to be efficient (also stable) according to Figure 6 (a) and to be very accurate according to Figure 6 (b). In contrast when the value of k is less than 4, the accuracy tends to drop and it becomes a little more time-consuming (Especially, $\xi = 30k, 50k$). The reason is that, it would be likely to miss the valid trip if only very few top-k sequences are considered as candidate trip, leading us to scan much more candidate sites. Therefore we should avoid the value of k to be very small. The figures also indicate the intermediate result tend to have acceptable accuracy rate if users request the result in an immediate fashion.

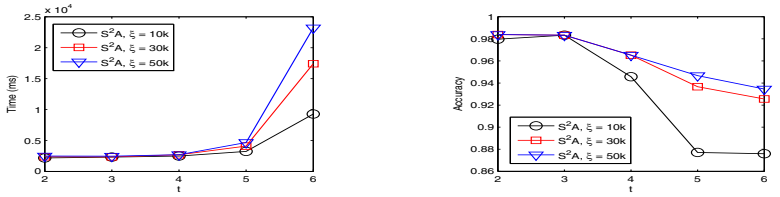


(a)Efficiency (b)Accuracy

Fig. 6. Performances via k

Effect of t. Figure 7 (a) and (b) show the efficiency and accuracy comparisons in different number of required types in query Q. Given that the time complexity is in exponential time to t, we limit the size of t to be in a reasonable range of the applications (otherwise the results cannot be obtained in an acceptable time). As shown in Figure 7 (a), the efficiency tends to be stable when the number of t is relatively small, but it climbs up sharply when the number of t becomes larger because of the huge search space accordingly. Besides, Figure 7 (b) shows that the accuracy drops dramatically when the number of t increases due to insufficient iterations.

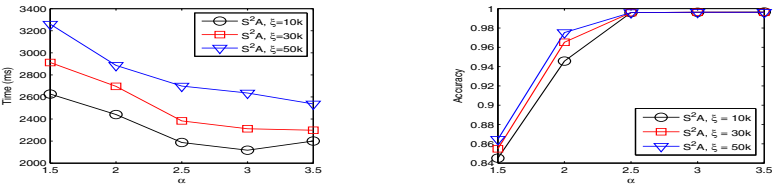
Effect of α. Figure 8 (a) and (b) show the efficiency and accuracy of the S^2A algorithm in varying distance thresholds based on the parameter α (the distance threshold is α times of the distance from beginning point to ending point of the query). We can easily observe from Figure 8 (a) that the efficiency is very sensitive to α, and the processing time significantly reduces when the distance



(a)Efficiency (b)Accuracy

Fig. 7. Performances via t

threshold becomes large. Similarly, the accuracy of the algorithm is also in the direct proportion to the parameter α according to Figure 8 (b). Above phenomena can be well explained by the fact that it is easier to find a good result if the distance threshold is relatively loose.



(a)Efficiency (b)Accuracy

Fig. 8. Performances via α

To sum up, the experimental results imply that the S^{2}A algorithm can support us to find the trip with high frequency efficiently in different parameter settings, which are related to the algorithm logic or to query itself.

7 Conclusion

In this paper, we have investigated the problem of trip search on categorical POI sites, to return users the trip that can maximize user satisfaction score within a given distance threshold. A spatial clustering-based approximate algorithm S^{2}A has been proposed to support efficient trip search, in which the sketched information in a global point of view is extracted to prune the search space in greater granularity. Extensive experimental results based on real datasets have demonstrated the effectiveness of the proposed method.

In the future, we would like to improve the algorithm by applying hierarchial clustering techniques to further improve the accuracy of trip search.

Acknowledgments. This work was partially supported by Chinese NSFC project under grant numbers 61402312, 61232006, 61303019, 61440053, Australian ARC project under grant number DP140103499, and Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

1. Roy, S.B., Das, G., Amer-Yahia, S., Yu, C.: Interactive itinerary planning. In: Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, Hannover, Germany, 11–16 April, pp. 15–26 (2011)
2. Kanoulas, E., Du, Y., Xia, T., Zhang, D.: Finding fastest paths on A road network with speed patterns. In: Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, Atlanta, GA, USA, 3–8 April, p. 10 (2006)
3. Li, F., Hadjieleftheriou, M., Kollios, G., Cheng, D., Teng, S.: Trip planning queries in road network databases. In: Encyclopedia of GIS, pp. 1176–1181 (2008)
4. Yang, B., Guo, C., Jensen, C.S., Kaul, M., Shang, S.: Stochastic skyline route planning under time-varying uncertainty. In: IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, 31 March–4 April, pp. 136–147 (2014)
5. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering route planning algorithms. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS, vol. 5515, pp. 117–139. Springer, Heidelberg (2009)
6. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM **34**(3), 596–615 (1987)
7. Li, F., Cheng, D., Hadjieleftheriou, M., Kollios, G., Teng, S.-H.: On trip planning queries in spatial databases. In: Medeiros, C.B., Egenhofer, M., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633, pp. 273–290. Springer, Heidelberg (2005)
8. Shang, S., Ding, R., Yuan, B., Xie, K., Zheng, K., Kalnis, P.: User oriented trajectory search for trip recommendation. In: Proceedings of the 15th International Conference on Extending Database Technology, EDBT 2012, Berlin, Germany, 27–30 March, pp. 156–167 (2012)
9. Sharifzadeh, M., Kolahdouzan, M.R., Shahabi, C.: The optimal sequenced route query. VLDB J. **17**(4), 765–787 (2008)
10. Yuan, J., Zheng, Y., Xie, X., Sun, G.: Driving with knowledge from the physical world. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, 21–24 August, pp. 316–324 (2011)
11. Horvitz, E., Krumm, J.: Some help on the way: opportunistic routing under uncertainty. In: The 2012 ACM Conference on Ubiquitous Computing, Ubicomp 2012, Pittsburgh, PA, USA, 5–8 September, pp. 371–380 (2012)
12. Tian, Y., Lee, K.C.K., Lee, W.: Finding skyline paths in road networks. In: Proceedings of the 17th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2009, Seattle, Washington, USA, 4–6 November, pp. 444–447 (2009)
13. Safar, M., El-Amin, D., Taniar, D.: Optimized skyline queries on road networks using nearest neighbors. Personal and Ubiquitous Computing **15**(8), 845–856 (2011)
14. Zhu, A.D., Ma, H., Xiao, X., Luo, S., Tang, Y., Zhou, S.: Shortest path and distance queries on road networks: towards bridging theory and practice. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, 22–27 June, pp. 857–868 (2013)
15. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction hierarchies: faster and simpler hierarchical routing in road networks. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 319–333. Springer, Heidelberg (2008)

16. Sankaranarayanan, J., Samet, H., Alborzi, H.: Path oracles for spatial networks. PVLDB **2**(1), 1210–1221 (2009)
17. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In transit to constant time shortest-path queries in road networks. In: Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, 6 January (2007)
18. Shang, S., Deng, K., Xie, K.: Best point detour query in road networks. In: Proceedings of the 18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2010, San Jose, CA, USA, 3–5 November, pp. 71–80 (2010)
19. Shang, S., Ding, R., Zheng, K., Jensen, C.S., Kalnis, P., Zhou, X.: Personalized trajectory matching in spatial networks. VLDB J. **23**(3), 449–468 (2014)
20. Zheng, K., Shang, S., Yuan, N.J., Yang, Y.: Towards efficient search for activity trajectories. In: 29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, 8–12 April, pp. 230–241 (2013)
21. Chen, Z., Shen, H.T., Zhou, X., Zheng, Y., Xie, X.: Searching trajectories by locations: an efficiency study. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, 6–10 June, pp. 255–266 (2010)
22. Chen, Z., Shen, H.T., Zhou, X.: Discovering popular routes from trajectories. In: Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, Hannover, Germany, 11–16 April, pp. 900–911 (2011)
23. Zheng, K., Trajcevski, G., Zhou, X., Scheuermann, P.: Probabilistic range queries for uncertain trajectories on road networks. In: Proceedings of the EDBT 2011, 14th International Conference on Extending Database Technology, Uppsala, Sweden, 21–24 March, pp. 283–294 (2011)