World Wide Web (2016) 19:679–705
DOI 10.1007/s11280-015-0352-2

On personalized and sequenced route planning

Jian Dai[1,2] · Chengfei Liu[3] · Jiajie Xu[4] · Zhiming Ding[5]

Received: 25 September 2014 / Revised: 24 February 2015 /
Accepted: 1 May 2015 / Published online: 11July 2015

Abstract Online trip planning is a popular service that has facilitated a lot of people greatly. However, little attention has been paid to personalized trip planning which is even more useful. In this paper, we define a highly expressive personalized route planning query-the *Personalized and Sequenced Route* (*PSR*) Query which considers both personalization and sequenced constraint, and propose a novel framework to deal with the query. The framework consists of three phases: *guessing*, *crossover* and *refinement*. The *guessing* phase strives to obtain one high quality route as the baseline to bound the search space into a circular region. The *crossover* phase heuristically improve the quality of multiple guessed routes via a modified genetic algorithm, which further narrows the radius of the search space. The *refinement* phase backwardly examines each candidate point and partial route to rule out impossible ones. The combination of these phases can efficiently and effectively narrow our search space via a few iterations. In the experiment part, we firstly show our evaluation results of each phase separately, proving the effectiveness of each phase. Then, we present the evaluation results of the combination of them, which offers insight into the merits of the proposed framework.

Keywords Spatial databases · Online route planning

✉ Jian Dai
daijiancn@126.com

[1] Institute of Software, Chinese Academy of Sciences, Beijing 100190, China

[2] University of Chinese Academy of Sciences, Beijing 100049, China

[3] Department of Computer Science and Software Engineering, School of Software and Electrical Engineering, Faculty of Science, Engineering and Technology, Swinburne University of Technology, Melbourne, VIC3122, Australia

[4] School of Computer Science and Technology, Soochow University, Suzhou, China

[5] School of Computer Science, Beijing University of Technology, Beijing 100124, China

1 Introduction

Location based services (LBS) are used in a variety of personal life nowadays, and the usefulness of route planning techniques in the LBS systems have been demonstrated by [1, 6, 8, 14]. With the widespread use of the mobile devises, people can easily access route planning services provided by Google, Bing, Baidu, etc. through Internet, to find routes. Through route planning, an end-user can be suggested to a proper route based on some selected places, in which all of their intended activities can be carried out. Therefore, effective route planning algorithms contribute to improve the quality of our entertainment and work, as well as travel efficiency.

Current route planning services can be classified into two categories. The first service is to provide a travel route that passes through the user specified points and the route is only measured by one factor [3, 5] (e.g. travel distance or travel time). Normally, such kind of services is fully supported online.[1,2,3] However, in many cases, where people arrive in unfamiliar cities, they tend to plan their routes based on categorical constraints. To cater for this, the second category focuses on computing a route that passes by those points according to user specified point of interest (POI) categories. In [7], the trip planning query (denoted by *TPQ*) is proposed, to find the shortest path from the starting point to destination through some computed POIs that can cover the categorical requirement of a user. In [13], the optimal sequenced route (*OSR*) query is defined, which strives to find a route of minimum length starting from a given source location and passing through a number of typed locations in a particular order imposed on the types of the locations. Both of them cannot support category based planning with multiple factors. As a matter of fact, a tourist may concern multiple factors rather than a single factor only. For example, the tourist may care about both travel distance and the popularity of POIs. Furthermore, the tourist may assign different importance to different categories. Therefore, we need define a novel route planning query query to model the tourist's need.

Consider the example in Figure 1 where a tourist plans to visit *clothes store*, *restaurant* and *electronics store* in a sequence from the source location q_1. And the tourist may place different weights on the categories according to his intention of the trip. For example, if he needs buy a personal computer, he may assign a relatively high weight to the *electronics store*. In contrast, if he mainly wants to buy a piece of clothes and just passes by an *electronics store* to check the new arrivals, then he probably gives a higher weight to the *clothes store*.

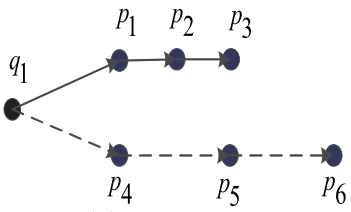
Form this illustrative example, we can see that different people may own different purposes or emphases for their routes. Hence, different weights should be associated with the different categories to reflect them accordingly. Furthermore, another aspect that will influence the choice of specific points from the category is the distance between the source location and the location of the point. Therefore, we should consider these two aspects together and balance them.

The current route planning approaches cannot support the variety of the weights assigned to sequentially travelled POI categories. The shortest path finding approaches only aim to find a shortest path from one place to another using distance or time as the weight. As shown

[1] http://map.google.com

[2] http://map.bing.com

[3] http://map.baidu.com



(a) Two Routes

Category	Point of Interest	Rating
Clothes store	p_1	1
Restaurant	p_2	1
Electronics store	p_3	1
Clothes store	p_4	5
Restaurant	p_5	5
Electronics store	p_6	5

(b) Demo POIs

Figure 1 An illustrative example

in Figure 1a and b, if we incrementally apply the shortest pathfinding algorithms to obtain the route sequentially passing *clothes store*, *restaurant* and *electronics store*, then the route $r_1(q_1 \rightarrow p_1 \rightarrow p_2 \rightarrow p_3)$ should be returned. Because the travel distance of r_1 is shorter than the other route $q_1 \rightarrow p_4 \rightarrow p_5 \rightarrow p_6$. Meanwhile, by applying conventional keyword based route planning algorithms to find the route, we probably obtain the same route (i.e., r_1) for it both covers the required keywords and has the shortest distance. However, another route $r_2(q_1 \rightarrow p_4 \rightarrow p_5 \rightarrow p_6)$ could be the optimal one if a tourist mainly concern about the rating factor. It's clear that if different weight combinations are applied, then different routes may become the optimal ones.

Therefore, to improve the flexibility of route query, we devise a novel route planning query, called **p**ersonalized and **s**equenced **r**oute planning(*PSR*) query which not only considers multiple factors of a route but also associates different weights with each category. To answer a *PSR* query, we mainly face two challenges.

1. **A *PSR* query includes multi-folded factors**. Clearly, distance is always an important concern for every traveller. If a place is far away from current location, it may become less attractive. Additionally, the rating of a place can play a determinative role in travelling as well. If the distances of two places are the same, people normally will choose the higher rating one to visit. Thus, we should take multiple concerns into account, which significantly enlarges the search space. A factor is a concern that a tourist uses it to evaluate the quality of his travel route. In particular, we consider the distance and the rating factors in this paper. Nevertheless, our approach can be easily extended to support multiple aspects.
2. **Different POI categories own different weights**. Different from previous route planning approaches, the POI category weights are associated with each query in order to flexibly reflect the route planning personalization. In other words, each trip has its own weights combination which is offered by an end-user. This flexibility also bring the computational complexity to us. An optimal route for a *PSR* may become suboptimal or even inferior one for another *PSR*, making it difficult to pre-compute some intermediate results.

To effectively and efficiently tackle with the *PSR* query, we propose a framework that not only guarantees corresponding bounds but also is capable of discovering the optimal route for a *PSR* query. We dub the framework as **GCR**, including three phases *guessing*, *crossover* and *refinement*.

The *guessing* strives to discover high score routes in a nondeterministic manner. And three different strategies are accordingly proposed for *guessing*. One of them is highly efficient and scalable while the other two are more effective. The *crossover* aims to improve the

scores of the guessed routes via a modified genetic algorithm framework. In this phase, we firstly try to heuristically locate the bottleneck that mainly causes the poor quality; then estimate which route is more likely to promote current route; finally a crossover is performed and multiple generations are generated. The *refinement* utilizes the lower bound of the score to prune points and partial routes which in turn makes the *guessing* easier to discover better routes.

Our contributions can be summarized as follows.

- We formally define a novel *personalized and sequenced route* query, which can provide great convenience in our daily lives.
- We propose a novel index to manage spatial keywords which have value attributes associated and a framework(*GCR*) which contains three phases *guessing*, *crossover* and *refinement* for processing the query.
- We conduct extensive experiments which prove the effectiveness, the efficiency and the scalability and also evaluate the effect of different parameters.

The remainder of this paper is organized as follows. We first formally define the *PSR* query and related terms in Section 2. In Section 3, we firstly present our novel framework that is used to deal with the problem of *PSR* query and then gives the detailed algorithms for its three phases *guessing*, *crossover* and *refinement*. Our experiments are reported in Section 4. The related work is listed and compared in Section 5. At last, we offer the conclusions in Section 6.

2 Problem statement

In this section, we formally define *PSR* query and related terms. Meanwhile, some examples are employed to illustrate the definitions.

Definition 1 Point of Interest (POI). Each *POI* p in this paper consists of five attributes: *pid* is the id of the POI, p_x and p_y define the location, p_c describes its category and *rating* is the rating of it. And every *POI* is contained in the search space $\mathbb{S}$ (i.e. $p \in \mathbb{S}$).

Definition 2 Formally, we define that the preference consists of two parts.

1. w_d is a distance weight.
2. W is a group of weights, where each weight specifies how important a kind of place is.

where $0 \leq w_d \leq 1$ and $\forall w \in W$, we have $0 \leq w \leq 1$.

Definition 3 The personalized and sequenced route planning query (**PSR** query) q is a query that consists of three parts.

1. q_l is the query starting point $q_l = \{loc_x, loc_y\}$, which gives the starting location of the query.
2. q_S is a category sequence $q_S = (q_{c_1}, \cdots, q_{c_m})$ which specifies the order of the categories of the *POI*s that each result route should pass through. Each $q_{c_i} (1 \leq i \leq m)$ denotes a category.

3. q_P gives the trip planning preference, which includes w_d and W. Since the input parameter of a *PSR* query only tells which categories should be visited instead of the specific places, we use trip planning preference to depict different category importance.

A *PSR* query aims to find a route that sequentially passes through q_S from q_l and ends at the point which belongs to category q_{c_m} (m denotes the number of queried categories). Meanwhile, the route should have a high score computed by (3) defined in Definition 7 using q_P and the route as input.

For example, as shown in Figure 1, a *PSR* query can be $q = \langle q_l = q_1, q_S = (\text{clothes store, restaurant, electronics store}), w_d = 0.1, W = \{0.2, 0.2, 0.5\}\rangle$, suggesting the tourist may not care too much about the travel distance but eagerly want to visit an awesome electronics store.

Definition 4 Feasible route. Given a *PSR* qurey q, $r = (q_l, p_1, \cdots, p_m)$ is a feasible route for q if and only if $p_1 \in q_{c_1}, \cdots, p_m \in q_{c_m}$ and $p_i \in \mathbb{S}$ for each $1 \leq i \leq m$. And we define $F(q)$ as the set that includes all the feasible routes of the corresponding *PSR* query.

Definition 5 Cost function. Given a PSR query q, this function defines the cost for a feasible route r. In this paper, without loss of generality, we use the distance between consecutive points as their cost and the total cost is the sum of all distances and multiples the weight given by the preference. Please notice that the point contained in a route cannot repeat itself but the category can repeat.

$$C(r, q) = w_d \times (\|q, p_1\| + \sum_{i=1}^{m-1} \|p_i, p_{i+1}\|) \tag{1}$$

where r is a feasible route, q denotes the corresponding *PSR* query, w_d is the distance weight and $\|q, p_1\| + \sum_{i=1}^{m-1} \|p_i, p_{i+1}\|$ is the total length of r.

Definition 6 Value function. This function defines the value for a feasible route. In this paper, we use the rating of a *POI* to depict the value of it. And the value function value generally represents the quality of the feasible route.

$$V(r, q) = \sum_{i=1}^{m} w_i \times Q(p_i) \tag{2}$$

where r is a feasible route, q denotes the corresponding *PSR* query, w_i specifies how important the place is, and $Q(p_i)$ represents the rating of the place.

Definition 7 Score function. The score of a route for a query is

$$score(r, q) = \frac{V(r, q)}{C(r, q)} \tag{3}$$

we use the above score function and try to maximize *score(r,q)* because normally we want achieve more value with less cost.

For example, continue using the aforementioned *PSR* q which corresponds to Figure 1, we compare the scores of two feasible routes $r_1 = (q_1 \rightarrow p_1 \rightarrow p_2 \rightarrow p_3)$ and $r_2 = (q_1 \rightarrow p_4 \rightarrow p_5 \rightarrow p_6)$. $score(r_1, q) = \frac{1\times0.2+1\times0.2+1\times0.5}{0.1\times(1.41+1+1)} \approx 2.639$ and $score(r_2, q) = \frac{5\times0.2+5\times0.2+5\times0.5}{0.1\times(1.41+2+2)} \approx 8.318$. Apparently, the tourist would choose r_2 according to his query.

Definition 8 Optimal route. The optimal route is a feasible route which maximizes the corresponding score function.

$$r_{opt} = arg\, max_{r\in\digamma(q)} score(r, q) \tag{4}$$

We can prove that answering a *PSR* query is NP-hard.

Theorem 1 *The problem of answering PSR query is NP-hard.*

Proof We show that Traveling Salesman Problem (TSP) is polynomial time reducible to *PSR* query. Given G=(V,E), suppose we start from v_0, let $q_l = v_0$, $m = n$, $q_s = \{C_1, \cdots, C_1\}$, $|q_s| = n$, then every tour of G is a feasible route and vice versa (because we ask the point cannot be repeated while the category can). We set $w_d = 0$, $w_i = 1$ and $Q(p_i) = 1$, then $V(r, q) = \sum_{i=1}^{n} w_i \times Q(p_i) = n$. $score(r, q) = \frac{n}{C(r,q)} = \frac{n}{\text{the length of a tour}}$ where a tour is a feasible route for q. It is easy to see that the score function reaches the maximal value if and only if $C(r, q)$ reaches the minimal. □

The problem of answering a *PSR* query is NP-Hard, mainly because we need to know the whole route (suggested by (1)) before we calculate the score of the route.

3 Query processing

In this paper, to effectively and efficiently generate a satisfactory answer for a *PSR* query, we propose a novel framework *GCR*. It has three phases: *guessing*, *crossover* and *refinement*.

The framwork overview is provided in Figure 2. It consists of three components. In the first first component, a set of reference routes are promptly generated from the search space via **Guessing**. These reference routes may have low score values but the cardinality of the reference route set can be large since we devise some highly efficient strategies to generate them. Based on the large number of reference routes, we further generate higher score routes by performing **Crossover**s on them. In crossover, we assign higher chosen probability to good quality routes and select them as parents to breed next generation routes. After a few of generations, we can expect some higher score routes are generated. Based on these routes, we pick up the best one out of them and use its score as the lower bound of the undiscovered optimal route score. As a result, some points which cannot contribute sufficient score are pruned and hence new search space is emerged. We name the pruning process as **Refinement**. In turn, the new search space which is often significantly smaller than the previous one is used as the search space for the next iteration.

The intuition of this framework is that some easily generated routes are used to compute the lower bound of the undiscovered optimal one and iteratively narrow the search space until it is possible to conduct exhaustive search.

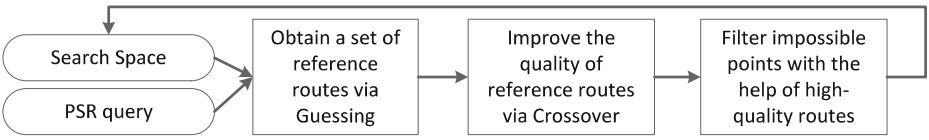


Figure 2 Framework overview

3.1 Narrowing the search space via guessing

To perform our smart guessing, we first design an Ranged-R-Tree (denoted by RR-Tree) to help efficiently retrieving the *POI*s, then with the help of the RR-Tree, we propose three ways to effectively guess the feasible routes.

An RR-tree not only records the summarized spatial information (i.e., rectangles) but also contains the ranges of the included *POI* values. Similar to R-Tree (and IR^2-Tree), an RR-Tree is a height-balanced tree data structure, where each leaf node has entries of the form (*ObjPtr*, *A*, *S*, *V*). *ObjPtr*, *A* and *S* are defined as in the IR^2-Tree while *V* includes the maximum value and the minimum value of the points inside the node. Since each node corresponds an MBR, the maximum value and the minimum value are equivalent to the maximum value and the minimum value for all the points in its subtrees.

Nondeterministic algorithms are very efficient in giving an answer to a difficult problem [4]. Here, we exploit nondeterministic algorithm to heuristically guess some feasible routes and use them to bound the search space to a circular region.

We generate a feasible route by combining a sequence of guessed points where each point is obtained by a random walk on the corresponding RR-Tree. A walk consists of a group of steps and each step starts from a node of an RR-tree and ends at a selected child of it. Consequently, a walk on a tree is a path from the root to a leaf and returns a point contained in the leaf node as an output. And the generated points are combined according to the category order of the query.

The whole process is illustrated in Figure 3 which also uses the dataset shown in Figure 1. As shown in Figure 3, two RR-trees, one for category C_1 (i.e., *clothes store* denoted by $\triangle$) and one for category C_2 (i.e., *restaurant* denoted by $\square$), are leveraged to seepup the *guessing*s. In order to generate a *POI* from $\triangle$ category (denoted by P_{C_1}), a search starts from the root of RR-tree for $\triangle$ and ends at P_4 where the intermediate nodes RR1 and RR4 are visited. Subsequently, P_4 is used to guess the next *POI* since they are correlated by the spatial locations. More concretely, after the selection of P_4, it should be more likely to select P_5 rather than P_2 since the distance between P_4 and P_5 is smaller than that of P_4 and P_2 and the rating of P_5 is greater than P_2. Similarity, the other *POI*s belonging to the queried categories can be fetched.

Abstractly, the guessing process can be formalized as (5).

$$\begin{aligned} r_g(q) \;=\; & q.l \oplus guess(C_1, q) \oplus \\ & \oplus\, guess(C_2, p_{C_1}) \oplus \\ & \oplus \cdots guess(C_m, p_{C_{m-1}}) \end{aligned} \tag{5}$$

where $p_{C_m} = \text{guess}(C_m, p_{C_{m-1}})$, $p_{C_1} = \text{guess}(C_1, q)$ and the operator means combining.

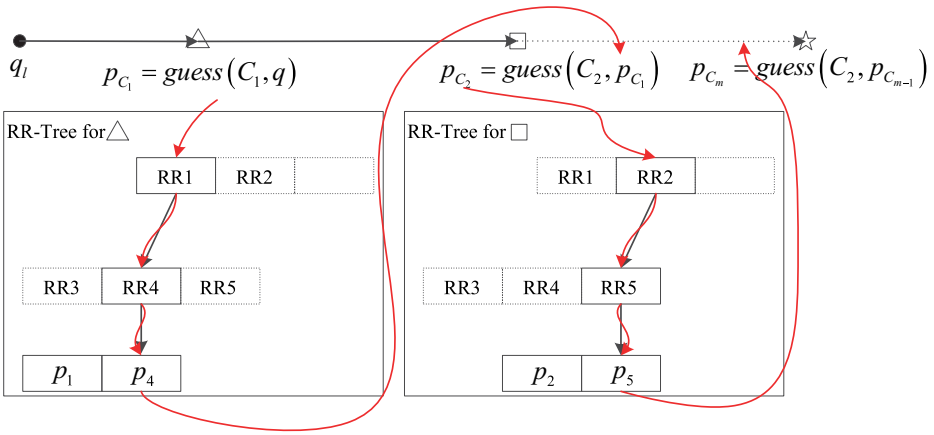


Figure 3 Guessing process overview

Figure 4 shows two MBRs both from level $i+1$ of an RR-Tree. The *HR(MBR)* represents the highest value contained in the MBR and the *LR(MBR)* represents the lowest value contained in the MBR. And p denotes the query location.

Intuitively, the distance between the query location and the MBRs and the highest (or lowest) value contained in the MBRs both affect the decision on which MBR to go next. Therefore, we define a probability of selection $Pr_s(MBR,p)$ to evaluate which MBR is more likely to be chosen where the query location is p. Please notice that the query location changes when we walk from one level to another level. And we design three strategies to compute $Pr_s(MBR,p)$.

1. **Random Choice Probability**. Using this strategy, each MBR is equally treated. And the chosen probabilities from current node to next level $(i+1)$ child nodes are the same, which is directly depended on the fanout.

$$R\text{-}Pr_s(MBR_j, p) = \frac{1}{|fanout|}. \tag{6}$$

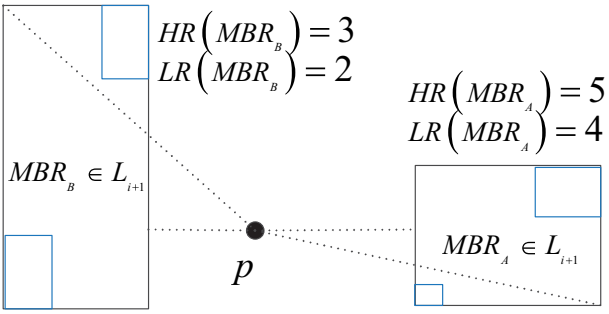


Figure 4 Three guessing strategies

2. **Pessimistic Choice Probability**. Using this strategy, the walk selects an MBR from next level $(i + 1)$ based on its possible lowest value and the MAXDIST between the point and the MBR. And the selection probability is normalized as follows.

$$PES\text{-}Pr_s(MBR_j, p) = \frac{\frac{LR(MBR_j)}{\text{MAXDIST}(MBR_j,p)}}{\sum_{MBR_i \in L_{i+1}} \frac{LR(MBR_i)}{\text{MAXDIST}(MBR_i,p)}}. \tag{7}$$

3. **Optimistic Choice Probability**. Using this strategy, the walk selects an MBR from next level $(i + 1)$ based on its possible highest value and the MINDIST between the point and the MBR. And the selection probability is normalized as follows.

$$OPT\text{-}Pr_s(MBR_j, p) = \frac{\frac{HR(MBR_j)}{\text{MINDIST}(MBR_j,p)}}{\sum_{MBR_i \in L_{i+1}} \frac{HR(MBR_i)}{\text{MINDIST}(MBR_i,p)}}. \tag{8}$$

After a feasible route is obtained, it actually indicates a circular region that the optimal route should be enclosed. By definition, $\forall score(r_{guess}, q) \leq score(r_{opt}, q)$. And $score(r_{opt}) = \frac{\sum_{i=1}^{m} w_i \times Q(p_i)}{w_d \times dist}$. Let R_{guess} denote all the guessed routes, and $r^{\star}_{guess} = arg\,max_{r \in R_{guess}} score(r, q)$, then we have a lower bound of the optimal score, $LB_{score} = \text{score}(r^{\star}_{guess}, q) \leq score(r_{opt}, q)$ Assume $\forall_{p_j \in C_i} Q(p_j) \leq MAX(C_i)$, i.e. each category has its own highest rating bound, we can compute the radius r_g of the circular region as follows.

Theorem 2

$$r_g \leq \frac{\sum_{i=1}^{m} w_i \times MAX_Q(c_i)}{w_d \times LB_{score}} \tag{9}$$

Proof

$$\begin{aligned} LB_{score} &\leq score(r_{opt}, q) \\ &= \frac{\sum_{i=1}^{m} w_i \times Q(p_i)}{w_d \times dist} \\ &\leq \frac{\sum_{i=1}^{m} w_i \times MAX_Q(c_i)}{w_d \times dist} \end{aligned}$$

Hence

$$r_g = dist \leq \frac{\sum_{i=1}^{m} w_i \times MAX_Q(c_i)}{w_d \times LB_{score}}$$

□

The guessing is highly effective to discover the optimal points (p is an optimal point when $(p \in r_{opt})$). We have a point collision between two routes r_1 and r_2, if $r_1 \cap r_2 \neq \emptyset$. When a route r has a point collision with the optimal route, $r \cap r_{opt} \neq \emptyset$. Here, we prove even the random guess can yield a high collision probability $Pr_c(r_{guess}, r_{opt})$ with the optimal route.

Theorem 3

$$Pr_c(r_{guess}, r_{opt}) = \sum_{i=1}^{m} \prod_{j=1}^{k} (1 - \frac{j}{|C_i|}) \tag{10}$$

Proof

$$\begin{aligned} Pr_c(r_{guess}, r_o pt) = \; & Pr_c(r_{guess} \cap r_{opt} \in C_1) + \\ & + Pr_c(r_{guess} \cap r_{opt} \in C_2) + \cdots + \\ & + Pr_c(r_{guess} \cap r_{opt} \in C_m) \end{aligned}$$

Meanwhile,

$$Pr_c((r_{guess} \cap r_o pt = p) \in C_i) = \prod_{j=1}^{k} (1 - \frac{j}{|C_i|})$$

Hence,

$$Pr_c(r_{guess}, r_o pt) = \sum_{i=1}^{m} \prod_{j=1}^{k} (1 - \frac{j}{|C_i|})$$

□

According to the birthday paradox principle, the collision probability grows quickly even when k is small. Hence, we can easily obtain some optimal points hidden inside the guessed routes. Furthermore, considering there may exist multiple optimal points for a category, the collision probability is even larger.

The implementation Algorithm 1 describes the details of *guessing*, where the main loop iteratively generates routes and finally returns them. Specifically, each iteration generates a feasible route r according to the specified policy (lines 4-10) and adds it into the R_{guess} (line 11).

Algorithm 1 Guessing

Input: a PSTPQ q, the search space $\mathbb{S}$, the policy used *policy*, the number of guessed routes K_{guess}
Output: the guessed routes R_{guess}

```
1  k ← 1;
2  R_guess ← ∅;
3  while k < K_guess do
4      switch policy do
5          case Optimistic
6              r ←generate a route via the equation 8;
7          case Pessimistic
8              r ←generate a route via the equation 7;
9          case Random
10             r ←generate a route via the equation 6;
11     R_guess ← R_guess ∪ r;
12     k ← k + 1;
13 return R_guess;
```

Lemma 1 *Let K_{guess} be the number of routes guessed. Given the search space $\mathbb{S}$ and m RR-Trees of $C_1, C_2, \cdots, C_m$, where the Random policy is used, Algorithm 1 returns R in $O(K_{guess} \sum_{i=1}^{m} log_M^{|C_i|})$ I/O's.*

Proof Since we totally need K_{guess} routes, a point can be obtained via a specific RR-Tree requires $\log_M^{|C_i|}$ I/Os. Therefore, Algorithm 1 returns R in $O(K_{guess} \sum_{i=1}^{m} \log_M^{|C_i|})$ I/O's. □

3.2 Breaking bottlenecks via crossovers

To utilize the guessed optimal points, our observation is that when they are put together with proper points, their performance is high; on the contrary, when unbefitting points constitute a route, they may behave ordinarily or even become the bottleneck of the current route. In other words, the scores of guessed points can be significantly improved once they are placed in suitable routes.

In order to place the right points into the right routes, we adopt the genetic algorithm. Besides the conventional steps asked by the genetic algorithm, there are two additional steps in this phase to improve the results obtained from *guessing*. First, we need identify the bottlenecks of the guessed routes and preferably use them as the crossover points. Second, we need explore the suitable routes to conduct crossovers, which can potentially lead to the score improvement.

Identifying potential bottlenecks Intuitively, a bottleneck point p_{bn} of a route r is the one which averagely contributes high score but fails to provide enough score in current route (i.e. r). Hence, the whole k guessed routes should be taken into account to find a p_{bn}. For example, in Figure 5a, the point denoted by a rectangle in r_2 is a bottleneck.

Formally, for a point $p \in C_k$ in a route r_j, we employ $\overline{score(C_i)}$ to denote the average score of the category C_i from the k guessed routes, and $dev(r_j, C_i)$ to represent the deviation from the $\overline{score(C_i)}$, i.e. $dev(r_j, C_i) = score(r_j, C_i) - \overline{score(C_i)}$. And hence employ (11) to evaluate the probability of being a bottleneck.

$$P_{bn}(p \in C_k, r_j) = \frac{index - dev(r_j, C_k)}{\sum_{i=1}^{|R_{guess}|} i} \tag{11}$$

The function *index-dev*(r_j, C_k) returns the index of r_j when we sort the set $\{dev(r_i, C_k) | r_i \in R_{guess}, 1 \le k \le m\}$ in a descending order. For example, we have three guessed routes $\{r_1, r_2, r_3\}$ and their $dev(r, C_1)$ are $\{1, -1, 2\}$ respectively, then *index-dev*(r_1, C_1)=2, *index-dev*(r_2, C_1)=3, *index-dev*(r_3, C_1)=1. Accordingly, $P_{bn}(p \in C_1, r_1) = 2/6$, $P_{bn}(p \in C_1, r_2) = 3/6$ and $P_{bn}(p \in C_1, r_3) = 1/6$. We avoid to use $\sum_{i=1}^{m} dev$ because we need to distinguish the positive deviations and the negative ones.

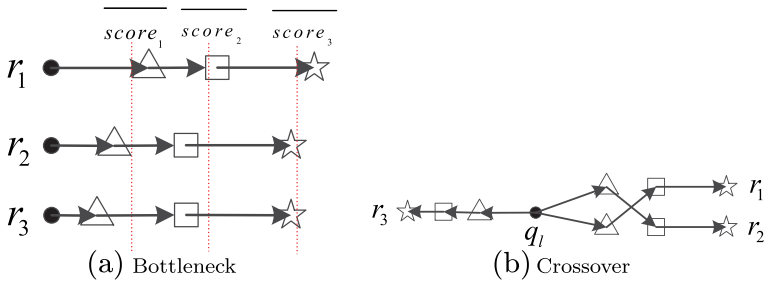


Figure 5 Bottleneck & crossover

Crossover routes selection Another observation is that a crossover between two near routes can usually improve score more than a crossover between two far away routes. For example, in the Figure 5b, it's more beneficial for r_1 to crossover with r_2 instead of r_3. To crossover more effectively, once a route r is determined to perform crossover which is controlled by the crossover probability P_c, the other route r' for crossovering with it is chosen based on a probability $P'_c(r')$.

$$P'_c(r') = \frac{\frac{score(r')}{dist(r,r')}}{\sum_{r'' \in G_{pre}} \frac{score(r'')}{dist(r,r'')}} \tag{12}$$

In (12), two factors are considered to select the other route. First, the distance factor which guarantees the near routes are easier to be chosen. Specifically, $dist(r, r')= \sum_{i=1}^{m} \| p_i - p'_i \|$, where $p_i \in r$ and $p'_i \in r'$. Second, the fitness(score) function which guarantees the good quality routes are more likely to be chosen. Here, the G_{pre} represents the previous generation of the routes.

The implementation *Initial population*. As shown in Algorithm 2, the initial population directly comes from the *guessing* phase (line 2). Therefore, there are totally K_{guess} routes.

Selection For each successive generation, a proportion of the existing population is selected in accordance with their fitness to breed a new generation (line 11). Here, we define the fitness equal to the score computed by our score function and adopt a fitness-based roulette process to select the elite.

In a roulette selection, the probability that a route is selected is computed with the following equation.

$$P(choice = i) = \frac{score(r_i)}{\sum_{j=1}^{K_{guess}} score(r_j)} \tag{13}$$

Crossover A pair of routes are selected for breeding two new routes (lines 4-10). The first route is sequentially chosen (line 5) and its bottleneck probabilities are computed according to (11) (line 7). Then, the other route is chosen according to (12) (line 6). Finally, a random number is generated(line 8), if the number is greater than the crossover probability P_c, new routes are born by exchanging the four segments of the two routes determined by the bottleneck(lines 9-10). Otherwise, if the number is less than P_c, new routes are directly cloned from their parents. The process continues until a new population of routes of the same size as the previous generation is produced, i.e. all the guessed routes are sequentially visited.

Since the *selection* operation tries to keep routes with good fitness(score) and the *crossover* operation tries to jump out of those local optima via heuristically breeding new routes, the genetic algorithm ultimately results in better routes than the initial population.

Termination This generational process ends when the predefined generation iteration number K_{GA} has been reached.

Algorithm 2 Crossover

Input: the PSTPQ q, the search space $\mathbb{S}$, the guessed routes R_{guess}, the crossover probability P_c, the generation iteration number K_{GA}
Output: the improved routes R_{cs}

```
1  k ← 0;
2  R_cs ← R_guess;                                   ▷ initial population
3  while k < K_GA do
4      for i ← 0, i < |R_cs|, i ← i + 1 do
5          r_first ← R_cs[i];
6          r_second ←obtain a route from R_cs by equation 12;
7          compute the crossover location by equation 11;
8          rand←generate a random number;
9          if rand> P_c then
10             perform crossover and generate two new routes;
11     select routes to form a new generation by equation 13;
12     k ← k + 1;
13 return R_cs;
```

3.3 Filtering with four pruning rules

After the genetic algorithm, better routes are generated. Consequently, r_g is further narrowed according to (9). It seems the optimal route can eventually emerge after multiple iterations of guessing-crossovers. However, holding with better routes, we can eliminate some impossible points before guessing which helps guess more effectively. In this subsection, four pruning rules are described to substantially reduce the enclosed points and partial routes in the narrowed circular region.

A feasible route r for query q can be viewed as a sequence of consecutive points. In this manner, we have $score(r) = \sum_{i=1}^{m} score'(p_i)$. And for each point, we have

$$score'(p_i) = \frac{w_i \times Q(p_i)}{w_d \times dist(p_i, p_{i-1})} \tag{14}$$

where $p_0 = q.l$.

Obviously, given a point p_i, the $score'(p_i)$ reaches its maximum value when $dist(p_i, p_{i-1})$ is the minimum distance.

$$maxscore(p_i) = \frac{w_i \times Q(p_i)}{w_d \times MINDIST\|p_i, p_{i-1}\|} \tag{15}$$

On the contrary, the $score'(p_i)$ reaches its minimum value when $dist(p_i, p_{i-1})$ is the maximum distance.

$$minscore(p_i) = \frac{w_i \times Q(p_i)}{w_d \times MAXDIST\|p_i, p_{i-1}\|} \tag{16}$$

Based on these two bounds, a backwardly pruning mechanism can be established to eliminate points inside the circular region which are impossible to be included in the optimal route.

Pruning rule 1. For a point p belonging to the last category ($p \in C_m$), if its largest score contribution is less than any corresponding guessing one ($p_{guess} \in C_m$), then it can be safely pruned.

$$maxscore(p) < min(score'(p_{guess})) \tag{17}$$

Here, we use $min(score'(\cdot))$ instead of $minscore$, because the guessed routes are known, we can simply use an iteration within the guessed routes to obtain the minimum score.

Figure 6a illustrates the idea of pruning rule 1 (and rule 2 also). Suppose p'_3 is found by *guessing*, if we have $maxscore(p_3) < min(score'(p'_3))$, then p_3 can be ruled out according to pruning rule 1. Meanwhile, if $minscore(p'_3) < minscore(\forall p_{guess} \in R_{guess})$, which means p'_3 owns a tighter lower bound than the guessing result, we can memorize it and use it for future pruning.

Pruning rule 2. For a point p belongs to the last category ($p \in C_m$), if its largest score contribution is less than the existing score of another point p' from the last category, then it can be safely pruned.

$$maxscore(p) < minscore(p') \tag{18}$$

Note that we cannot prune a point p which belongs to a category rather than the last category using rules 1 and 2, because the combinational score (i.e. route score) of it may be large.

Pruning rules 1 and 2 tell that a point p can be pruned, when either the minimum score of another point p' or the guessed score is greater than the maximum score of it. This rule is straightforward because when a point p' can provide higher score than p and they both belong to the last category, we should always replace p by p', which will not affect the other points and surely improve the overall score.

After the points belonging to the last category are filtered, we proceed to examine the rest points.

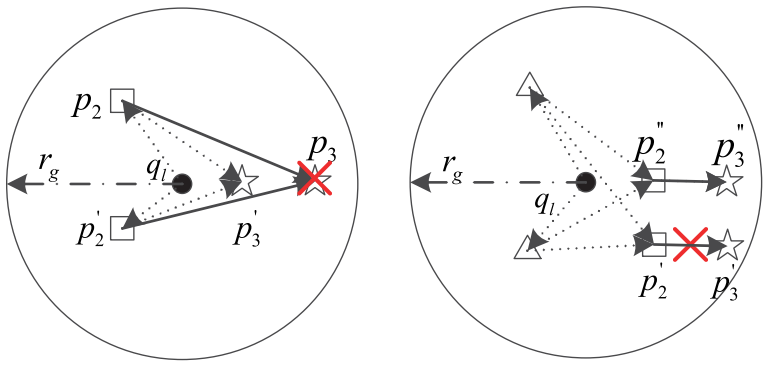


Figure 6 Pruning rules

Backwardly, if we treat the last i points as a super point, then we naturally have similar pruning rules. For simplicity, we demonstrate how the rules work when the last two points p_{m-1} and p_m are considered as a super point denoted by $\Phi(p_{m-1}, p_m)$.

$$score'(\Phi(p_{m-1}, p_m)) = score'(p_{m-1}) + score'(p_m) \tag{19}$$

Note that (19) can be easily extended to the whole route when we incrementally treat the partial route as a super point.

Given a point p_m, the maximum and the minimum score of the super point (i.e. partial route) $\Phi(p_{m-1}, p_m)$ is:

$$maxscore(\Phi(p_{m-1}, p_m)) = score'(p_m) + maxscore(p_{m-1}) \tag{20}$$

$$minscore(\Phi(p_{m-1}, p_m)) = score'(p_m) + minscore(p_{m-1}) \tag{21}$$

Similar to pruning rule 1 and pruning rule 2. We have two new rules.

Pruning rule 3. The super point can be safely pruned (please note pruning a super point is equivalent to pruning the partial route instead of discarding the two points, i.e. discard a combination of included points) when its maximum score is less than the corresponding guessed score.

$$maxscore(\Phi(p_{m-1}, p_m)) < min(score'(\Phi_{guess}(p_{m-1}, p_m))) \tag{22}$$

Pruning rule 4. The super point $\Phi(p_{m-1}, p_m)$ can be safely pruned when its maximum score is less than either the minimum guessed score or the score of another super point $\Phi(p'_{m-1}, p'_m)$, where $\{p_{m-1}, p'_{m-1}\} \in C_{m-1}$ and $\{p_m, p'_m\} \in C_m$.

$$maxscore(\Phi(p_{m-1}, p_m)) < minscore(\Phi(p'_{m-1}, p'_m)) \tag{23}$$

Note that the pruning rules 3 and 4 actually consist of a collection of rules, since we can extend the (19) to the whole route.

The (22) and (23) are formalized in a backward manner such that the maximum score of the super points can be caclulated, because the previous points have already been examined. For example, when computing $maxscore(\Phi(p_{m-1}, p_m))$, we have already examined the category C_m.

The implementation In order to reduce the number of comparisons [11], we translate pruning rule 1 into a spatial distance join that detects intersection between objects from adjacent categories. Instead of finding all pairs of $p_{m-1} \in C_{m-1}$ and $p_m \in C_m$ such that $\|p_{m-1}, p_m\| \leq \frac{w_m \times Q(p_m)}{w_d \times min(score'(p_{guess}))}$, we increase the size of all objects from C_m by $r_e = \frac{w_m \times Q(p_m)}{w_d \times min(score'(p_{guess}))}$ and test for intersection with points of C_{m-1}. Please note that the $Q(p_m)$, $min(score'(p_{guess}))$, w_m and w_d are all constants after the *guessing* phase.

The translated distance join is equivalent to the original pruning rule 1. Since we ask $maxscore(p) \geq min(score'(p_{guess}))$ (otherwise it should be pruned by rule 1), and we have $maxscore(p_m) = \frac{w_m \times Q(p_m)}{w_d \times MINDIST\|p_m, p_{m-1}\|}$, hence,

$$MINDIST\|p_m, p_{m-1}\| \leq r_e = \frac{w_m \times Q(p_m)}{w_d \times min(score'(p_{guess}))} \tag{24}$$

And the goal of our distance join is to discover the pairs who own the distance less than r_e. Therefore, we can safely convert the pruning process into the join process.

Meanwhile, pruning rule 2 can be implemented via an estimation. As a matter of fact, pruning rule 2 implicitly asks for the $MAXDIST\|p_m, p_{m-1}\|$ to further compute the corresponding *minscore*. However, it is usually time consuming to directly obtain the precise *MAXDIST*. Fortunately, all the points here are inside a circular region which is centered at $q.l$ and uses r_g as its radius (according to the equation 9). Hence, we have $MAXDIST\|p_m, p_{m-1}\| \leq \|q.l, p_m\| + r_g$, and $q.l, r_g$ are constants after the *guessing* phase. Consequently,

$$minscore(p_m) \geq \frac{w_m \times Q(p_m)}{w_d \times (\|q.l, p_m\| + r_{guess})}$$

Obviously, if there exists a point p'_m and

$$\begin{aligned} maxscore(p'_m) &< \mathrm{LB}_{score'(C_m)} \\ &= max_{\forall p_m \in C_m} \frac{w_m \times Q(p_m)}{w_d \times (\|q.l, p_m\| + r_{guess})} \end{aligned} \tag{25}$$

then p_m should be discarded. Here, we denote the right part in (25) as the $\mathrm{LB}_{score'(C_m)}$ which can be directly computed after the *guessing* and *crossover* phases.

Similarly, pruning rules 3 and 4 can be implemented via corresponding distance join and $\mathrm{LB}_{score'(C_i)}$. The following algorithm gives the details.

Algorithm 3 Refinement

Input: the PSTPQ q, the improved routes after crossover R_{cs}
Output: the filtered search space $\mathbb{S}'$

1 $i \leftarrow m-1$ $\mathbb{S}' \leftarrow \mathbb{S}$;
2 load the points into memory via a region query on RR-trees according to the equation 9;
3 **while** $i > 0$ **do**
4 **if** $i < m-1$ **then**
5 $\mathrm{LB}_{score'} \leftarrow \sum_i^{m-1} \mathrm{LB}_{score'(C_i)}$;
6 **else**
7 $\mathrm{LB}_{score'} \leftarrow \mathrm{LB}_{score'(C_i)} \leftarrow max_{\forall p_m \in C_m} \frac{w_m \times Q(p_m)}{w_d \times (\|q.l, p_m\| + r_g)}$;
8 enlarge each point $p \in C_i$ by radius $r_e \leftarrow \frac{w_{i+1} \times Q(p_{i+1})}{w_d \times min(score'(p_{guess}))}$;
9 build the corresponding RR-tree T_i according to the enlarged circles;
10 **for** *each point* $p' \in C_{i+1}$ **do**
11 try to assign p' to an MBR hierarchically from the root of T_i to a leaf according to its spatial attributes;
12 $MBR_{p'} \leftarrow$the eventual MBR that contains p' but none of its children contains p';
13 **if** *$MBR_{p'}$ is null or $MBR_{p'}$ isn't a leaf node or p' doesn't locate inside any enlarged p* **then**
14 $\mathbb{S}' \leftarrow \mathbb{S}' - p'$; ▷ discard p'
15 **else**
16 find the nearest neighbor based on the subtree of T_i using $MBR_{p'}$ as the root and p' as the query point;
17 $p_{NN} \leftarrow$the nearest neighbor of p';
18 compute the maxscore(p');
19 **if** *$maxscore(p') < LB_{score'}$* **then**
20 $\mathbb{S}' \leftarrow \mathbb{S}' - p'$; ▷ discard p'
21 $i \leftarrow i-1$;
22 **return** $\mathbb{S}'$;

In Algorithm 3, we start from the category $i = m-1$ (line 1) and backwardly examine each point of the category $i+1$ in a main loop (lines 3-21). First of all, the corresponding $\mathrm{LB}_{score'}$ can be computed according to the (25). Please note that when i equals $m-1$,

we can compute the $\mathrm{LB}_{score'}$ directly for there is only one category m to be considered (lines 6-7). Otherwise, we need compute the $\mathrm{LB}_{score'}$ incrementally for the super point represents a partial route from category i to m(lines 4-5). Then, the points belonging to the category i are physically enlarged by the radius r_d according to the (24)(line 8) and new RR-Tree T_i for the enlarged points is built(line 9). After that, we examine each point by our pruning rules in a loop (lines 10-20). Specifically, we try to assign each point ($p' \in C_{i+1}$) to the MBRs of T_i from the root to a leaf (lines 11-12). If the point p' locates inside an MBR, then we further test if it still locates inside a child MBR of the MBR. The process ends when we eventually find an MBR (denoted by $MBR_{p'}$) which contains p' but none of its child contains p'. Surely, if the $MBR_{p'}$ is null or isn't a leaf or doesn't locate inside any enlarged p, then we can safely prune it based on prune rule 1 (for point) or 3 (for partial route) (lines 13-14). Furthermore, we testify the point p' with pruning rules 2 and 4 (lines 15-20). Based on the found $MBR_{p'}$ (i.e. leaf node) on the T_i, we can perform a nearest neighbor search to obtain the nearest neighbor of p' (lines 16-17). Compute the maxscore(p') and compare it with the pre-computed $\mathrm{LB}_{score'}$ (line 18). If maxscore(p') is less than $\mathrm{LB}_{score'}$, then we prune it in accordance with prune rules 2 and 4 (line 19). Please note that the p' here may also represent a super point (i.e. a partial route). In Figure 7a and b, we demonstrate how the pruning rules are implemented respectively.

3.4 GCR: putting them together

Through the *guessing* phase, we actually obtain a radius r_g that bounds our search space. In the *crossover* phase, we further improve the quality of the guessed routes via a genetic algorithm framework in order to narrow the bounded search space. And in the *refinement* phase, a more strict and precise examination is conducted based on the four pruning rules, which aims to eliminate the impossible points and partial routes.

However, we may not obtain a solvable search space after sequentially executing the above three algorithms. This is because r_g is estimated and may not be tight enough. An efficient way to tighten the bound is to guess more routes and select the best one. But when the search space is extreme large, sole *guessing* can only find some good routes rather than the optimal one. We find it is also highly efficient to perform multiple iterations of the combined three algorithms. This is because after several iterations, the search space often becomes extremely small enough which can quickly run these algorithms.

Algorithm 4 gives the details.

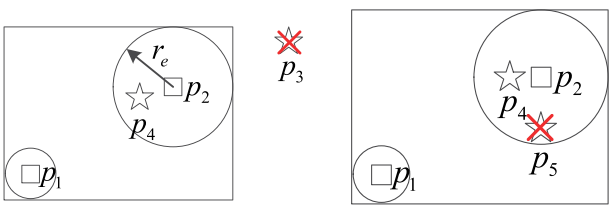


Figure 7 Implementation of pruning rules

Algorithm 4 GCR

Input: the PSTPQ q, search space $\mathbb{S}$, the guessing policy *policy*, the number of guessed routes K_{guess}, solvable number of points M, the genetic algorithm K_{GA}, the crossover probability P_c, the iteration number K_I, the iteration number for higher score routes K_g

Output: a high-score route r for q (possibly $score(r) = score(r_{opt})$)

```
1  i ← K_I, j ← K_g;
2  r_g ←an empty route;
3  while i > 0 do
4      while j > 0 do
5          guess K_guess feasible routes R_guess by algorithm 1;
6          improve the quality of guessed routes by algorithm 2;
7          j ← j − 1;
8          δ ←max(score(r ∈ R_guess))-score(r_g);
9          if j = 0 and δ ≤ 0 then
10             return r_g;
11         else if j > 0 and δ ≤ 0 then
12             K_guess ← K_guess × 2;
13             continue;
14         else
15             r_g = arg max_{1≤i≤K_guess} score(r_i ∈ R_guess);
16             break;
17     S' ←filter in-memory points by the algorithm 3;
18     i ← i − 1;
19     if |S'| > M and i > 0 then
20         S ← S';
21     else if |S'| > M and i = 0 then
22         r = arg max_{1≤i≤K_guess} score(r_i ∈ R_guess);
23         return r;
24     else
25         r ←exhaustive search to obtain the optimal route;
26         return r;
```

In Algorithm 4, we first assign the iteration number to a local variable *i* and the *guessing-crossover* iteration number for higher score route to *j*(line 1). And use a local variable r_g to store the best route found so far. Then in the main loop (lines 2-26), we iteratively perform the *guessing*, *crossover* and *refinement* according to the algorithms 1, 2 and 3, respectively. Please notice that normally after the search space is narrowed, it's easier to obtain the higher score routes via *guessing*. However, occasionally we find the score hasn't been improved even after the *crossover* phase. There are two reasons caused this phenomenon. First, we may already obtain the optimal route. Second, the search space is extremely large that K_{guess} guesses are not sufficiently large to discover high score routes. As for the second case, we should increase the K_{guess} and strive to find some high score routes. To avoid endless loop, we break the *guessing-crossover* loop after K_g iterations. Hence, after one *guessing* and one *crossover* (lines 5-6), we examine the j and δ. If $j = 0$ and $\delta < 0$, we return the best route so far(lines 9-10). If $j > 0$ and $\delta < 0$, we set $K_{guess} \leftarrow K_{guess} \times 2$ and continue the loop (lines 11-13). Otherwise (i.e. we find a better route via *guessing-crossover*), we can update the best route and jump out of the loop(lines 14-16). Subsequently, we use the best route to perform *refinement*(line 17). After the three phases, we reduce the value of i by 1, then examine the number of the points enclosed in the refined search space ($\mathbb{S}'$) and the i. If $i > 0$ and the enclosed point is greater than predefined M (which is a small integer, indicating the problem can be solved via an exhaustive search), then we need replace the previous search search $\mathbb{S}$ by the narrowed $\mathbb{S}'$ and continue the loop (lines 19-20). If $i = 0$ and the enclosed point is greater than predefined M, we have run out

of the iterations and need return the best route found so far (lines 21-23). Otherwise, we can directly perform an exhaustive search to find the optimal route (lines 24-26).

Fortunately, lines 9-10 are rarely executed in our experimental evaluations. Nevertheless, to ensure not wasting the computation time, we design the shortcut which directly returns the best route found so far when we find the highest score hasn't been improved after the *crossover*.

In Algorithm 4, the new guessed routes R_{guess} will be accepted unless $max(score(r \in R_{guess})) > score(r_g)$ after each *guessing* and *crossover*. Consequently, according to (9) the radius will be reduced. As the radius keeps decreasing, the number of candidate points also significantly decline. Meanwhile, the *refinement* phase also tries to eliminate the impossible points and partial routes. Hence, for each iteration, we have

Lemma 2 *Let $\mathbb{S}$ denote the search space before an iteration and $\mathbb{S}'$ denote the search space after the iteration, $|\mathbb{S}'| \leq |\mathbb{S}|$.*

Proof During each iteration, all the three phases don't introduce more points. Therefore, $|\mathbb{S}'| \leq |\mathbb{S}|$. □

Lemma 3 *Let r' denote the best route found before an iteration and r denote the best route after the iteration, $score(r') > score(r)$*

Proof Before the termination of Algorithm 4, it passes the best route to the next iteration. And both *guessing* and *crossover* try to discover better routes. Therefore, $score(r') > score(r)$. □

According to Lemmas 2 and 3, it's possible that the optimal route is eventually discovered (lines 25-26) when K_I and K_g is large enough.

4 Experimental results

In this section, we report an extensive experimental evaluation of the proposed algorithms: *guessing*, *crossover*, *refinement* and the combination of them (i.e. *GCR*). Varying paraments and dataset sizes are used to test the effectiveness of algorithms, as well as efficiency and scalability.

We totally used five datasets in our experiments and these datasets are all indexed by the proposed RR-Trees (page size=1K bytes, node capacity=16). First, we used a real-world dataset. It contains 68,840 *POI*s collected via the public APIs provided by the website map.baidu.com. Each *POI* is associated with a set of spatial, keyword and rating attributes, including latitude, longitude, type and overall_rating. All of these attributes are parsed from the returned *JSON* strings. We used questionnaires to survey the preferences of 24 students. And different *PSR* querys are generated based the collected preferences and randomly generated query locations. Please notice that we generated far more than 24 queries because different query locations are used.

In order to compare the optimal route with the route found by our algorithm, we also used the synthetic datasets. Since the optimal route is difficult to discover directly, given multiple *PSR* querys, we generate synthetic datasets via two steps: first, optimal routes are explicitly constructed for different queries; then, the remaining *POI*s are randomly generated far away from the query locations and the optimal routes, which guarantees

the *POI*s will not become a point of an optimal route. More concretely, from the (15), we are able to compute the maximum score of a point. Further, according to the (25), we know that a point should not be considered in the optimal route finding if its maximum score is less than the current lower bound of the category. Therefore, the distance is computed by combining the equation 15 with (25), which means computing the lower bound with the corresponding point included in the optimal route and letting the maximum score of newly generated points less than it. Different sizes of query sets and corresponding datasets are generated in this way for both effectiveness experiments and scalability experiments. Because based on the query set and corresponding synthetic dataset we know the optimal route, we can examine the effectiveness of our proposed algorithms. Meanwhile, the sizes of the synthetic datassets are 5000, 10000, 15000 and 20000, respectively.

All the algorithms were implemented in Java 1.7.0 and run on a Windows 8 with Intel i5 CPU 2.5GHz, 4GB RAM, and a 1TB hard disk.

4.1 Effect of guessing

Figure 8 shows the experimental results for varying number of categories(m) and size of data size($|\mathbb{S}|$), where *Rand* represents the random choice probability described in Section 3.1, *PES* represents the pessimistic choice probability and *OPT* represents the optimistic choice probability.

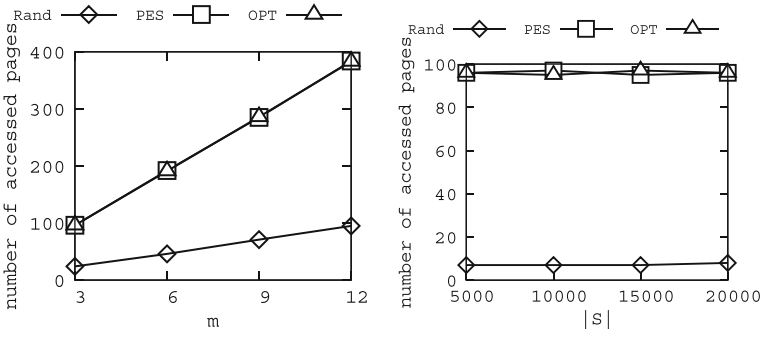
The first group of experiments (shown in Figure 8a) are conducted on the real-world dataset to evaluate the average number of accessed pages to obtain a guessed route. When the number of queried category gets larger, the number of accessed pages increases slowly for three strategies. The optimistic strategy and the pessimistic one have nearly the same number of accessed pages, which can also be clearly concluded from (7) and (8). And the random strategy always has smaller number of accessed pages which indicates it is much more efficient.

The second group of experiments (shown in Figure 8b) which are conducted on the synthetic datasets aim to evaluate the scalability of our *guessing* algorithm (described in Algorithm 1). We can see that when the size of the dataset gets larger, the average number of accessed pages only increase slightly which can be negligible. The results clearly prove the excellent scalability of the *guessing*. As a matter of fact, we can conclude this from Figure 4 because when the size of data set gets larger, the RR-Tree increases in a logarithmic manner which guarantees the *guessing* is highly scalable.

The running time curves exhibit similar trends with Figure 8 since the number of accessed pages directly determines the running time. We can see from Figure 9 that since the *Rand* strategy only considers the fanout, it has the most efficient. In contrast, the other two strategies have to examine the total number of nodes which are the children of the current node, leading to more running time.

The random strategy is not only efficient and scalable, but very effective as well. Figure 10 gives the experimental results which proves the effectiveness of *guessing*.

Figure 10a shows the point accuracy of *guessing* when K_{guess} varies (default $m = 3$). For a collection of guessed routes, we define the point accuracy as *(the number of points shared by the optimal route and the guessed routes)/(the total number of points in a route)*. This group of experiments are conducted on a synthetic dataset ($|\mathbb{S}| = 2 \times 10^4$). Since we exactly know the optimal routes, the point accuracy can be easily computed. When the number of guessed routes increases, the point accuracy firstly grows sharply and then gently. All of the three strategies can reach above 0.9 point accuracy when we guess 80 routes from the



(a) I/O vs m (b) I/O vs $|\mathbb{S}|$

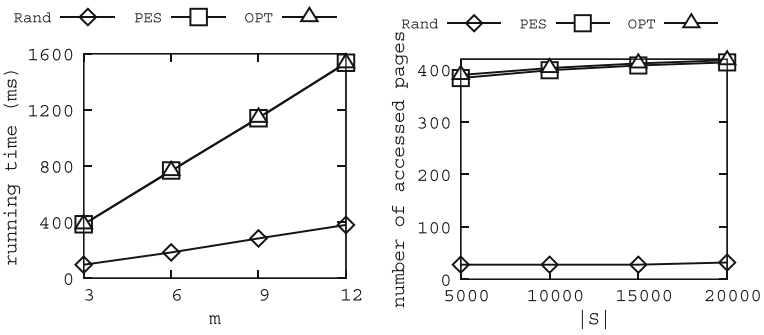
Figure 8 Performance of different *guessing* strategies

dataset, which proves *guessing* can successfully discover the points contained in the optimal routes.

Figure 10b shows the score accuracy of *guessing* when K_{guess} varies (default $m = 3$). For a collection of guessed routes, we define the score accuracy as *the highest score of the guessed routes/the optimal score*. This group of experiments are conducted on the synthetic dataset($|\mathbb{S}| = 2 \times 10^4$) as well. After the multiple *guessing*s, the optimistic and the pessimistic strategies can reach around 0.8 score accuracy and the random strategy can reach above 0.65 score accuracy, indicating we can narrow the search space into a relatively small circular region according to the (9).

4.2 Effect of crossover

The *guessing* phase brings us the optimal points. However, from Figure 10a and b, we can naturally conclude that these points are often distributed in wrong routes. Otherwise, the score accuracy should be as high as the point accuracy. In this subsection, we give our the experimental results on the *crossover* phase to show how much the scores can be improved via *crossover*.



(a) varying m (b) varying $|\mathbb{S}|$

Figure 9 Running time of *guessing*

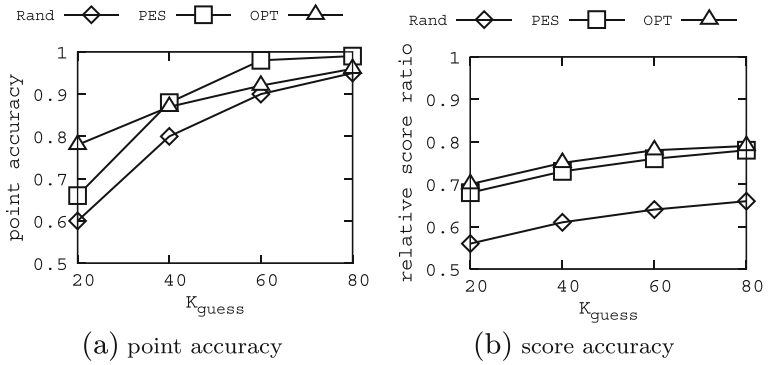


Figure 10 Accuracy of different *guessing* strategies

Figure 11a shows the improved score ratio(i.e. (*max score after the crossover-max score before the crossover*)/*the optimal score*) on a synthetic dataset($|\mathbb{S}| = 2 \times 10^4$) when we vary the number of generations(i.e. K_{GA}). For each number, we report the average improved score ratio over five runs and we use the default $P_c = 0.8$. We can see that the scores of the guessed routes increase when the K_{GA} grows. The routes from the optimistic and pessimistic strategies increase much slower than the randomly guessed routes. This is because the routes obtained by the optimistic and pessimistic strategies already have relatively high scores and the routes randomly guessed tend to have chaos points distribution. Through the *crossover*, we try to identify the bottlenecks and break them via exchanging the segments with selected routes. After the *crossover*, the guessed routes (including all routes obtained from the three strategies) can reach as high as 0.85 score accuracy. Another parameter is the crossover probability(P_c). Figure 11b shows the improved score ratio on the synthetic dataset($|\mathbb{S}| = 2 \times 10^4$) when we vary P_c. For each number, we report the average improved score ratio over five runs and we use the default $K_{GA} = 20$. The experimental results are similar to the ones shown in Figure 11a, which both prove the effectiveness of the *crossover*. According to our experimental results, it appears that increasing K_{GA} is more effective to improve the score than increasing P_c. However, another fact is increasing K_{GA} is more time costly than increasing P_c.

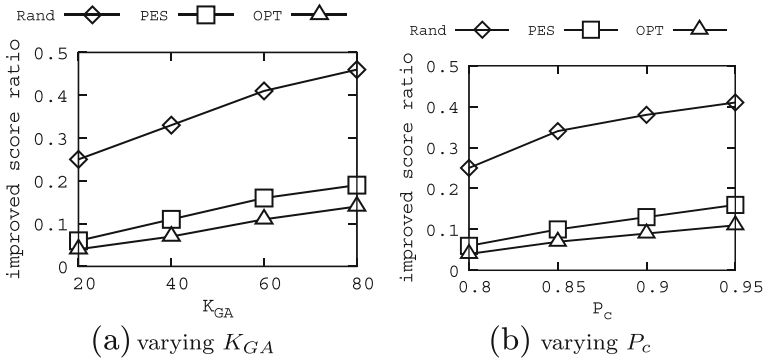


Figure 11 Crossover

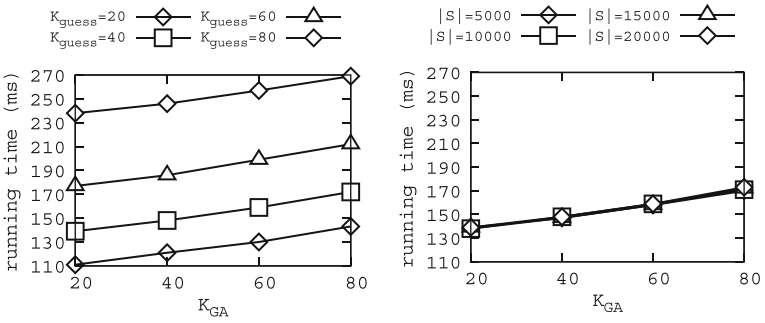
Figure 12 displays the running times of *Crossover* under different parameter settings, where Figure 12a shows the average running times of *Crossover* with varying K_{guass}s and Figure 12b shows the running times when we use different dataset sizes. The first group of experiments are conducted on real world dataset (shown in Figure 12a). As the number of involved routes increases, the running time also grows gently. And when $K_{GA} = 80$ and $K_{guess} = 80$, the running time is still below 270ms, indicating the *Crossover* phase scales well. Additionally, according to Figure 12b, the size of dataset cannot influence the running time of this phase. This is because the *Crossover* only considers the routes generated by *Guessing*.

4.3 Effect of refinement

Using the improved routes, we can further narrow the search space according to (9). Subsequently, ruling out some impossible points and partial routes can in turn facilitate the *guessing*, making it easier to hit more optimal points (even the the optimal route). Eventually, making it possible to solve the original *PSR* query via an exhaustive search. The running time and the pruning power are two aspects we mainly focus in the *refinement* phase.

Figure 13a shows the experimental results for varying the queried sequence length (m). This group of experiments are conducted on the real-world dataset. There are two curves, one shows the pruning ratio when we vary m, the other shows the running time when we vary m. When the m increases, we can see that the pruning ratio gently decreases but still greater than 0.8 when $m = 12$; meanwhile, the running time increases but still less than 1900ms when $m = 12$. These are as expected. Because the bounds used in our estimation are not the tight bounds, when m is getting larger, we have more candidate points to filter which leads to the increasing running times and the decreasing pruning ratios. Nevertheless, in most cases, after many iterations of *GCF* we can convert the *PSR* query into a solvable problem.

Figure 13b shows the similar content as Figure 13a except that the datasets used here are the synthetic ones. And our goal is to investigate the scalability of our Algorithm 3. When we vary the size of the dataset, we can see that the pruning ratio only change slightly and the running time increase sublinearly, which indicates the superb scalability of Algorithm 3.



(a) varying K_{GA} (b) varying $|\mathbb{S}|$

Figure 12 Performance of *Crossover*

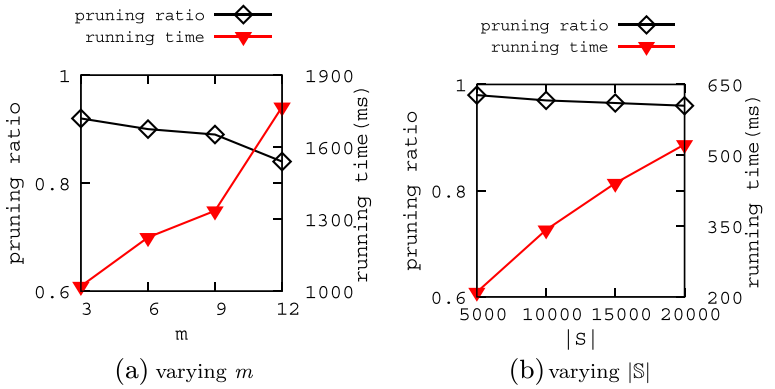


Figure 13 Refinement

4.4 Effect of GCR

The combination of the *guessing*, *crossover* and *refinement* produces a surprising result. The experimental results suggest that in most cases we can eventually narrow the search space into a solvable degree after multiple iterations, even though the initial real-world search space contains as many as 68,840 *POI*s.

Figure 14a shows the experimental results when we vary the iteration numbers(K_Is). As expected, the running time is becoming shorter and shorter and the improved score ratio is smaller and smaller when the K_I is increasing. This is because the search space is narrowed after each iteration (including *guessing*, *crossover* and *refinement*, where we used the random guessing strategy). Another reason leading to the results is that we actually load the points into the main memory in the first iteration, which saves the loading time for the subsequent iterations. Hence, it's acceptable and reasonable to run multiple iterations until the search space has been squeezed into a small region. This group of experiments are conducted in the real-world dataset. Fortunately, we observe that in most cases (over 80 %) the search space can be narrowed into a solvable degree (including only about one hundred *POI*s).

Figure 14b shows the experimental results when we vary the dataset size $|\mathbb{S}|$. The running time grows when the dataset size increases (we used the random guessing strategy). Please notice that another aspect we evaluated is the relative score ratio (defined as *max score obtained by GCR*)/*the optimal score*) rather than the improved score ratio. Because we used the synthetic datasets in this group of experiments. And the relative score ratio decreases when the $|\mathbb{S}|$ increases. However, the relative score ratio can maintain at a high value and the running time can maintain at an acceptable (less than 1s) value even when the $|\mathbb{S}|$ is very large.

Summary The evaluation results offer insight into the merits of the proposed *GCR*. The *GCR* framework benefits our problem in two ways. First, since the algorithms in *guessing* and the *crossover* are highly scalable and can provide good quality routes, it is efficient and can find routes of high quality. Moreover, the *refinement* further narrows the search space based on the concluded pruning rules, which makes the subsequent iterations even more efficient and more likely to discover the optimal route.

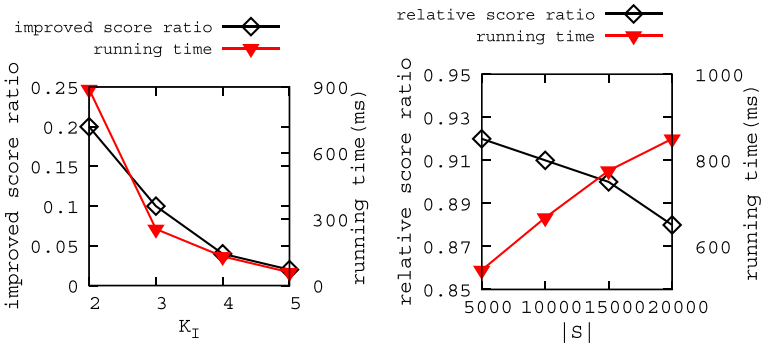


Figure 14 GCR

5 Related work

Due to its practical significance, the travel route search (or trip recommendation) problem and has attracted a lot of attentions. This section introduces some representative works related to this problem and compares them with our work.

Li et al. [8] formally propose a query named Trip Planning Query (*TPQ*). A *TPQ* consists of three input parameters, namely a start location s, an end location e and a set of categories C. The corresponding output of *TPQ* is the shortest path that starts at s, passes through at least one spatial object from each category in C and ends at e. Sharifzadeh et al. [13] defines a variant problem of *TPQ* and dubs it as the Optimal Sequenced Route query (*OSR*) in both vector and metric spaces. An *OSR* strives to find a route of minimum length starting from a given source location s and passing through a number of typed locations in a particular order imposed on the types of the locations. Exact algorithms are proposed to answer the query. Shang et al. [12] considers the User Oriented Trajectory Search (*UOTS*) for trip recommendation. Such kind of query input contains a set of intended places given by a traveler and a set of textual attributes describing the traveler??s preference.

Different from the above queries, the *PSR* query proposed in this paper considers the sequence of the categories in C and the users' preferences are also considered in the score function, which is a more general query with respect the users' preferences.

There exist many ways to answer a travel route search (or trip recommendation). Many of them focus on how to mine or learn certain patterns based on historical trajectories and/or check-in behaviors. On one hand, these techniques make our assumption that different preferences can be offline obtained reasonable and feasible. On the other hand, the *PSR* query is actually an online query which can flexibly capture and reflect the preferences of end-users which haven't been investigated before.

Lu et al. [9] build travel routes via utilizing the geo-tagged photos collected from the Flickr. The locations and trips all are defined with scores and the recommendation is performed according to the users' travel requirements. Chen et al. [2] define the problem of finding the most popular route between two locations based on the collected travel historical trajectories of many users. Malviya et al. [10] deal with the problem of answering continuous route planning queries in a road network environment. The route planning takes the delay of updates into account and aims to obtain the shortest path. Roy et al. [1] consider the interactive trip planning problem, where the users can give feedbacks for the suggested

points of interests and the routes are built in an iterative way based on some constraints. Yao et al. [15] propose a query by specifying a starting and an ending location and a set of (keyword, threshold) values pairs and name the query as the multi-approximate-keyword routing (MARK) query. Yoon et al. [16] propose a smart trip recommendation which leverages the historical GPS trajectories to efficiently generate itineraries. However, none of these proposals take into account the preferences as we do in this paper.

6 Conclusions and future work

This paper has presented a widely used query *PSR* query which hasn't been formally researched. We first formalize it and analyze its complexity. Then we propose a flexible and efficient framework which includes three phases. The *guessing* phase strives to obtain some good quality routes as the baselines to bound the search space into a circular region. The *crossover* phase heuristically improve the quality of the routes via a modified genetic algorithm framework, which further narrow the radius of search space. The *refinement* phase carefully and backwardly examines each candidate point and partial route to rule out impossible ones, which can eliminate the majority of the points and partial routes. Combining them together, it's surprising that we can often make the original *PSR* query solvable in an acceptable time, which eventually makes the online *PSR* querying possible.

As future work, it is of interest to automatically mine the personalized weight settings from historical trajectories to facilitate the personalized route computation. It is also of interest to take the availability of *POI*s into account to support more accurate route recommendation.

Acknowledgments This work was supported by the National High-tech Research and Development Program (863 Program) of China under Grant No. 2013AA01A603, the Pilot Project of Chinese Academy of Sciences under Grant No. XDA06010600 and the National Natural Science Foundation of China (Grant No. 61402312).

References

1. Basu Roy, S., Das, G., Amer-Yahia, S., Yu, C.: Interactive itinerary planning. In: 2011 IEEE 27th International Conference on Data Engineering (ICDE), pp. 15–26. IEEE (2011)
2. Chen, Z., Shen, H.T., Zhou, X.: Discovering popular routes from trajectories. In: Abiteboul, S., Böhm, K., Koch, C., Tan K.L., (eds.) ICDE, pp. 900–911. IEEE Computer Society (2011)
3. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering route planning algorithms. In: Algorithmics of Large and Complex Networks, pp. 117–139. Springer (2009)
4. Du, D., Ko, K.I., Hu, X.: Design and analysis of approximation algorithms, vol. 62. Springer (2012)
5. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM (JACM) **34**(3), 596–615 (1987)
6. 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, 2006. ICDE'06, pp. 10–10. IEEE (2006)
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.J., Bertino E., (eds.) SSTD, Lecture Notes in Computer Science, vol. 3633, pp. 273–290. Springer (2005)
8. Li, F., Hadjieleftheriou, M., Kollios, G., Cheng, D., Teng, S.H.: Trip planning queries in road network databases. In: Encyclopedia of GIS, pp. 1176–1181. Springer (2008)
9. Lu, X., Wang, C., Yang, J.M., Pang, Y., Zhang, L.: Photo2trip: generating travel routes from geo-tagged photos for trip planning. In: Bimbo, A.D., Chang, S.F., Smeulders, A.W.M. (eds.) ACM Multimedia, pp. 143–152. ACM (2010)

10. Malviya, N., Madden, S., Bhattacharya, A.: A continuous query system for dynamic route planning. In: 2011 IEEE 27th International Conference on Data Engineering (ICDE), pp. 792–803. IEEE (2011)
11. Nobari, S., Tauheed, F., Heinis, T., Karras, P., Bressan, S., Ailamaki, A.: Touch: in-memory spatial join by hierarchical data-oriented partitioning. In: Ross, K.A., Srivastava, D., Papadias, D. (eds.) SIGMOD Conference, pp. 701–712. ACM (2013)
12. Shang, S., Ding, R., Yuan, B., Xie, K., Zheng, K., Kalnis, P.: User oriented trajectory search for trip recommendation. In: Rundensteiner, E.A., Markl, V., Manolescu, I., Amer-Yahia, S., Naumann, F., Ari I. (eds.) EDBT, pp. 156–167. ACM (2012)
13. Sharifzadeh, M., Kolahdouzan, M., Shahabi, C.: The optimal sequenced route query. VLDB J. **17**(4), 765–787 (2008)
14. Xu, J., Guo, L., Ding, Z., Sun, X., Liu, C.: Traffic aware route planning in dynamic road networks. In: Database Systems for Advanced Applications, pp. 576–591. Springer (2012)
15. Yao, B., Tang, M., Li, F.: Multi-approximate-keyword routing in gis data. In: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 201–210. ACM (2011)
16. Yoon, H., Zheng, Y., Xie, X., Woo, W.: Smart itinerary recommendation based on user-generated gps trajectories. In: Ubiquitous Intelligence and Computing, pp. 19–34. Springer (2010)