This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

Efficient Retrieval of Bounded-Cost Informative Routes

Wengen Li, Jiannong Cao, *Fellow, IEEE,* Jihong Guan, Man Lung Yiu, and Shuigeng Zhou, *Member, IEEE,*

Abstract—The widespread location-aware applications produce a vast amount of spatio-textual data that contains both spatial and textual attributes. To make use of this enriched information for users to describe their preferences for travel routes, we propose a *Bounded-Cost Informative Route* (BCIR) query to retrieve the routes that are the most textually relevant to the user-specified query keywords subject to a travel cost constraint. BCIR query is particularly helpful for tourists and city explorers to plan their travel routes. We will show that BCIR query is an NP-hard problem. To answer BCIR query efficiently, we propose an exact solution with effective pruning techniques and two approximate solutions with performance guarantees. Extensive experiments over real data sets demonstrate that the proposed solutions achieve the expected performance.

Index Terms—Road network, route query, query keywords, informative route

1 Introduction

The constantly increasing volume of spatio-textual data brings abundant information about travel routes. Examples include comments and check-ins from Foursquare, geo-tagged photos with textual descriptions from Facebook, geo-tagged posts from twitter, and location-based advertisements from various business promotion platforms. It is appealing to take full advantage of such spatio-textual data to retrieve interesting routes.

Although there already exist some studies on route query over road networks using the spatio-textual data [1], [2], [3], [4], [5], they aim to find a route that covers a set of query keywords while minimizing or bounding the travel cost (e.g., travel distance). These queries are analogous to the Boolean keyword query in Web search engines and suitable for trip planning that aims at visiting particular types of points of interest (POIs), e.g., a route passing a restaurant and a bank. However, if users want to find a route whose text description is textually relevant to the given query keywords, keyword coverage route query cannot be applied because it does not compute a textual relevance score with respect to the query keywords for each route.

In this paper, we propose a *Bounded-Cost Informative Route (BCIR) query* which retrieves the optimal route that is the most textually relevant to the user-specified query keywords, subject to a travel cost constraint (e.g., the maximum travel distance or time). The reason for introducing a travel cost budget is that users usually have a cost budget in mind to avoid a high travel cost. Specifically, given a start location s and a destination d, query keywords $\mathcal{K}$ and a travel cost budget B, BCIR query finds the optimal route R^* from s to d such that: (i) the travel cost of R^* is bounded by the given travel cost budget B and (ii) the textual relevance (also called route score henceforth) between the keyword description of R^* and the query keywords $\mathcal{K}$ is maximized.

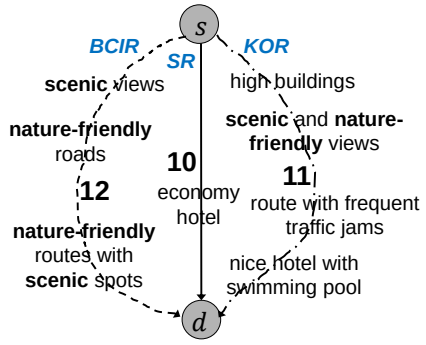


Fig. 1. The shortest route (SR), keyword-aware optimal route (KOR) and bounded-cost informative route (BCIR), where query keywords are "*scenic, nature-friendly*" and the textual description of each route can be extracted from online maps, Foursquare, social networks, etc.

For example, a tourist may issue a query with query keywords "*scenic, nature-friendly*" to find a route that is scenic and nature-friendly. Figure 1 illustrates the difference between shortest route (SR) query, keyword-aware optimal route (KOR) query [1] and BCIR query, where KOR query is the representative for the keyword coverage route queries. The number beside each route is the corresponding travel cost. The SR route is the shortest route from s to d yet contains no query keywords. The KOR route is the optimal route that passes all the query keywords once. In contrast, the BCIR route is the most relevant to "*scenic, nature-friendly*" than the other two routes.

BCIR query has many potential applications, e.g., finding interesting tourist routes, identifying emotional routes, and detecting those routes that are most relevant to complaint. Particularly, two representative applications are discussed below.

- *Wengen Li and Jihong Guan are with the Department of Computer Science and Technology, Tongji University, Shanghai, China. E-mail: {8lwengen, jhguan}@tongji.edu.cn*
- *Wengen Li, Jiannong Cao and Man Lung Yiu are with the Department of Computing, the Hong Kong Polytechnic University, Hong Kong, China. E-mail: {cswgli, csjcao, csmlyiu}@comp.polyu.edu.hk*
- *Shuigeng Zhou is with Shanghai Key Lab of Intelligent Information Processing, and the School of Computer Science, Fudan University, Shanghai, China. E-mail: sgzhou@fudan.edu.cn*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

Application scenario I: Finding interesting tourist routes
Tourists usually would like to explore interesting attractions when visiting a new city. With the BCIR query, tourists can easily specify what they are interested in by query keywords and search for the routes that are most relevant to these query keywords. As discussed in Figure 1, one tourist may issue a query to find a route relevant to "*scenic, nature-friendly*" from her current location to the hotel. More interestingly, BCIR query can retrieve routes of some specific topics. For example, a route with the topic "*shopping*" can be retrieved by specifying some query keywords relevant to shopping such as "*sale*", "*mall*" and "*shop*".

Application scenario II: Identifying emotional routes
In addition to finding efficient routes that are short or fast, users may also want to find routes that are emotionally pleasant [6]. Existing studies mainly focuses on computing the emotion scores for routes from different aspects such as *happiness*, *quietness* and *beauty*. Novelly, BCIR contributes a general and flexible way to compute emotionally pleasant routes by specifying some emotion-aware query keywords. For example, if someone wants to find a happy route, a BCIR query with query keywords "happy", "joy" and "delight" will help. Moreover, some dangerous and negative routes can be also identified by using query keywords like "*crime*", "*robbery*" and "*accident*" in BCIR query.

Challenges and Contributions
It is non-trivial to process BCIR query efficiently due to its non-additive property (cf. Section 2.2.2), i.e., the route score is computed based on the text description of the entire route rather than simply summing the scores of the edges. This characteristic differentiates BCIR query from existing keyword coverage route queries which have additive route scores. Furthermore, BCIR query is actually an NP-hard problem (cf. Section 2.2.1). For such a computationally expensive problem, we propose three solutions, an exact solution with efficient pruning techniques and two approximate solutions with performance guarantees. The exact solution with multiple pruning methods can greatly reduce the search space and promptly return exact results for BCIR queries of small travel cost budgets. In contrast, the two approximate solutions report good approximate results for BCIR queries of large travel cost budgets.

In sum, our contributions in this work are threefold as listed below.

- We propose the bounded-cost informative route (BCIR) query to retrieve interesting routes based on the textual relevance and analyze its hardness (Section 2).
- We design an exact solution with effective pruning techniques to compute exact results for BCIR queries of small travel cost budgets and propose two scalable approximate solutions which can compute satisfying query results for BCIR queries of large travel cost budgets efficiently (Sections 3, 4, 5 and 6).
- We conduct extensive experiments on real data sets of different sizes to evaluate the performance of proposed solutions (Section 7).

Then, the related work is reviewed in Section 8 and the conclusion with future work is presented in Section 9.

2 Problem Statement and the Hardness

In this section, we first formally define the BCIR query problem. Then, we prove that BCIR query is an NP-hard problem and discuss its non-additive property. Table 1 lists those notations frequently used in this work.

TABLE 1
Frequently used notations.

notation	meaning
$G(V, E)$	a road network
$e_{i,j}$	the edge from vertex v_i to vertex v_j
$c_{i,j}$	the travel cost of edge $e_{i,j}$
R	a route
E_R	the set of edges on route R
V_R	the set of vertices on route R
$SR_{i,j}$	the shortest route from v_i to v_j
$q = (s, d, \mathcal{K}_q, B)$	a BCIR query
R^*	the optimal route of BCIR query
$\mathcal{K}_{i,j}$	the keywords on edge $e_{i,j}$
$\mathcal{K}_R$	the keywords on route R
$\mathcal{C}^B_{s,d}$	all candidate routes from s to d
$\|*\|$	the cardinality of $*$
$f_{k,e}$	the frequency of keyword k on edge e
$f_{k,R}$	the frequency of keyword k on route R
$\lambda(R)$	the travel cost of route R
$\tau(R)$	the score of route R

2.1 Problem Definition

Before defining BCIR query, we first give the definition of road network.

Definition 1 (Road Network). A road network is modeled by a directed graph $G(V, E)$, where each vertex $v_i \in V$ represents an intersection of roads; each edge $e_{i,j} \in E$ represents the directed road segment from v_i to v_j and is associated with a travel cost $c_{i,j}$ and a set of keywords $\mathcal{K}_{i,j}$ with their numbers of occurrences.

Example 1. Figure 2 illustrates a small road network. For simplicity, all edges are assumed to be bi-directed. The travel cost and keywords are labelled beside each edge. For example, the travel cost of edge $e_{s,3}$ is $c_{s,3}$=5 and its keywords are $\mathcal{K}_{s,3}$=$\{k_1$: 1, k_3: 1$\}$ where the numbers count the occurrences of the corresponding keywords on edge $e_{s,3}$.

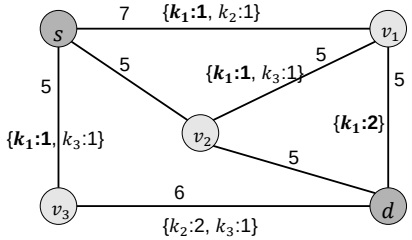


Fig. 2. A small road network with labelled travel costs and keywords for edges, and an example of BCIR query q=$(s, d, \mathcal{K}_q$=$\{k_1\}, B$=12).

The travel cost $c_{i,j}$ of edge $e_{i,j}$ can be any non-negative metric, e.g., the length of $e_{i,j}$ and the time to travel through $e_{i,j}$. The keywords $\mathcal{K}_{i,j}$ of edge $e_{i,j}$ can be extracted from various sources like the geo-textual comments on Foursquare and Facebook.

A route R=$\langle v_{x_1}, \ldots, v_{x_\gamma} \rangle$ consists of a sequence of vertices, where edge $(v_{x_i}, v_{x_{i+1}}) \in E$ and $v_{x_i} \neq v_{x_j}$ if $x_i \neq x_j$ ($1 \leq i, j \leq \gamma$). We use E_R and V_R to denote those edges and vertices on route R, respectively. The travel cost of R, $\lambda(R)$,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

is computed by summing the travel costs of its edges, i.e., $\lambda(R)=\sum_{e_{i,j}\in E_R} c_{i,j}$. The shortest route from v_i to v_j is denoted by $SR_{i,j}$ whose travel cost is $\lambda(SR_{i,j})$. The keywords of route R, $\mathcal{K}_R$, are computed by merging the keywords of its edges, i.e., $\mathcal{K}_R = \uplus_{e_{i,j}\in E_R} \mathcal{K}_{i,j}$, where $\uplus$ is a union operator for multi-set that allows duplicate instances.

The BCIR query is then defined as below.

Definition 2 (Bounded-Cost Informative Route (BCIR) Query). Given a road network G, a BCIR query is denoted by $q=(s, d, \mathcal{K}_q, B)$, where s and d specify the start location and destination, respectively, $\mathcal{K}_q$ is a set of query keywords, and B is a travel cost budget. The objective of q is to find the optimal route R^* from s to d such that:

$$R^* = \arg \max_{R\in\mathcal{C}^B_{s,d}} \tau(R) \quad (1)$$

where $\tau(R)$ computes the textual relevance between $\mathcal{K}_R$ and $\mathcal{K}_q$; $\mathcal{C}^B_{s,d}$ represents the set of candidate routes, from s to d, with travel costs less than or equal to B, i.e.,

$$\mathcal{C}^B_{s,d} = \{R | R \in \mathcal{C}_{s,d} \wedge \lambda(R) \leq B\} \quad (2)$$

and $\mathcal{C}_{s,d}$ represents the entire set of simple routes (routes without duplicate vertices) from s to d.

The cost budget B is specified by a deviation ratio $\mu \geq 0$ from the shortest route between two query locations, i.e., $B=(1+\mu)\cdot\lambda(SR_{s,d})$. However, the travel cost budget B can be also specified by users directly.

In this work, we compute route score $\tau(R)$ by utilizing the TF-IDF model [7] which is widely used in information retrieval. Concretely, we have

$$\tau(R) = \frac{\sum_{k\in(\mathcal{K}_R\cap\mathcal{K}_q)} (w_{k,R})\cdot(w_{k,q})}{\sqrt{\sum_{k\in\mathcal{K}_R}(w_{k,R})^2 \cdot \sum_{k\in\mathcal{K}_q}(w_{k,q})^2}} \quad (3)$$

where $w_{k,R}=1+\ln(f_{k,R})$ and $f_{k,R}$ counts the occurrences of keyword k in $\mathcal{K}_R$; $w_{k,q}=\ln(1+\frac{|E|}{|E_k|})$ and E_k are those edges containing keyword k.

Example 2. Figure 2 also illustrates a BCIR query, where the start and destination locations are s and d, respectively; $\mathcal{K}_q=\{k_1\}$ specifies one query keyword, and the travel cost budget is B=12. As listed in Table 2, there are five possible routes. Specifically, for route $R_1=\langle s, v_1, d\rangle$, we have

$$\mathcal{K}_{R_1} = \mathcal{K}_{s,1} \uplus \mathcal{K}_{1,d} = \{k_1 : 3, k_2 : 1\}$$

where the numbers record the occurrences of the corresponding keywords. For example, $k_1 : 3$ indicates that keyword k_1 appears three times on route R_1. We then have $w_{k_1,R_1}=1+\ln 3$ and $w_{k_2,R_1}=1+\ln 1$. Meanwhile, we have $|E_{k_1}|=4$, $|E_{k_2}|=2$ and $|E_{k_3}|=3$, which count the numbers of edges containing keywords k_1, k_2 and k_3, respectively. For query keyword k_1, we have

$$w_{k_1,q} = \ln(1+\frac{|E|}{|E_{k_1}|}) = \ln(1+\frac{7}{4}) = \ln\frac{11}{4}.$$

Accordingly, the score of route R_1 is

$$\begin{aligned}\tau(R_1) &= \frac{\sum_{k\in(\mathcal{K}_{R_1}\cap\mathcal{K}_q)}(w_{k,R_1})\cdot(w_{k,q})}{\sqrt{\sum_{k\in\mathcal{K}_{R_1}}(w_{k,R_1})^2\cdot\sum_{k\in\mathcal{K}_q}(w_{k,q})^2}} \\ &= \frac{(1+\ln 3)\cdot(\ln\frac{11}{4})}{\sqrt{((1+\ln 3)^2+(1+\ln 1)^2)\cdot(\ln\frac{11}{4})^2}} \\ &= 0.903\end{aligned}$$

The scores of the other four routes can be computed in the same way. The details of the five routes are summarized in Table 2. Routes R_4 and R_5 should be pruned since their travel costs are larger than the travel cost budget B=12. The other three routes are candidate routes and R_1 is returned as the query result because its score is larger than that of routes R_2 and R_3.

TABLE 2
All possible routes for the BCIR query in Figure 2

Route	Cost	Keyword	Score
$R_1=\langle s, v_1, d\rangle$	12	$\{k_1 : 3, k_2 : 1\}$	0.903
$R_2=\langle s, v_2, d\rangle$	10	$\{\}$	0
$R_3=\langle s, v_3, d\rangle$	11	$\{k_1 : 1, k_2 : 2, k_3 : 2\}$	0.385
$R_4=\langle s, v_2, v_1, d\rangle$	15	$\{k_1 : 3, k_3 : 1\}$	0.903
$R_5=\langle s, v_1, v_2, d\rangle$	17	$\{k_1 : 2, k_2 : 1, k_3 : 1\}$	0.767

2.2 Problem Hardness

2.2.1 NP-hardness

BCIR query can be proved to be NP-hard. First, we define the decision problem of BCIR query, Decision-BCIR, as below.

Definition 3 (Decision-BCIR). Given a road network G, a BCIR query $q=(s, d, \mathcal{K}_q, B)$ and a score threshold $X > 0$, Decision-BCIR decides whether there exists a route from s to d such that its cost is at most B and its score is at least X.

Then, we have the following theorem.

Theorem 1. Decision-BCIR problem is NP-hard.

Proof: This theorem can be proved by a reduction from the Hamiltonian Route/Path (Ham-Route for short) problem, a well-known NP-Complete problem, to the Decision-BCIR problem. Given a road network $G(V, E)$, Ham-Route problem decides whether there exists a route passing each vertex in V once and only once. We can formulate an instance of the Decision-BCIR problem based G as below.

- **Road network** $G'(V', E')$**:** We create a new road network $G'(V', E')$ by letting $V'=V\cup\{v_s, v_d\}$ and $E'=E\cup\{(v_i, v_j)|v_i\in\{v_s, v_d\}\wedge v_j\in V\}$, where v_s and v_d are two new vertices. For each edge $e_{i,j}\in E'$, we set its cost $c_{i,j}$=1. In addition, we assume that each edge $e_{i,j}\in E$ in G' contains one unique keyword and the edges in $E'-E$ do not contain any keywords.
- **Decision-BCIR problem** $q=(s, d, \mathcal{K}_q, B)$**:** The start location s and destination d correspond to vertices v_s and v_d in G', respectively. Query keywords $\mathcal{K}_q$ contains all the keywords in G', i.e., $|\mathcal{K}_q|=|E|$. We set the cost budget $B=n+1$, and set the score threshold $X=\sqrt{\frac{n-1}{|E|}}$, where $n=|V|$.

Then, we can prove that G has a Ham-Route R if and only if there exists a route R' from v_s to v_d on G' such that the cost of R' is at most $n+1$ and the score of R' is at least $\sqrt{\frac{n-1}{|E|}}$.

One the one hand, assuming that there is a Ham-Route R in G that passes all the n vertices in V, we can get a new route R' in G' by adding v_s and v_d to the two ends of R, respectively. The cost of R' is $n+1$ because R' has $n+2$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

vertices and the cost of each edge is exactly 1. Meanwhile, the score of R' is

$$\tau(R') = \frac{\sum_{k\in(\mathcal{K}_{R'}\cap\mathcal{K}_q)}(w_{k,R'})\cdot(w_{k,q})}{\sqrt{\sum_{k\in\mathcal{K}_{R'}}(w_{k,R'})^2\cdot\sum_{k\in\mathcal{K}_q}(w_{k,q})^2}}. \quad (4)$$

Considering that each edge (except for the edges connecting to v_s and v_d) in G' contains one unique keyword, we have $|E'_k|$=1 for keyword k and $|\mathcal{K}_{R'}|$=$n-1$. Furthermore, we have $w_{k,q}$=$\ln(1+\frac{|E'|}{|E'_k|})$=$\ln(1+|E'|)$ and $w_{k,R'}$=$1+\ln(f_{k,R'})$=$1+\ln(1)$=1. We then have

$$\begin{aligned}\tau(R') &= \frac{\sum_{k\in(\mathcal{K}_{R'}\cap\mathcal{K}_q)}1\cdot\ln(1+|E'|)}{\sqrt{\sum_{k\in\mathcal{K}_{R'}}1^2\cdot\sum_{k\in\mathcal{K}_q}(\ln(1+|E'|))^2}}\\ &= \frac{|\mathcal{K}_{R'}|\cdot\ln(1+|E'|)}{\sqrt{|\mathcal{K}_{R'}|\cdot|\mathcal{K}_q|}\cdot\ln(1+|E'|)}\\ &= \sqrt{\frac{|\mathcal{K}_{R'}|}{|\mathcal{K}_q|}}=\sqrt{\frac{n-1}{|E|}} \quad (5)\end{aligned}$$

where $\sum_{k\in(\mathcal{K}_{R'}\cap\mathcal{K}_q)}(w_{k,R'})\cdot(w_{k,q})=|\mathcal{K}_{R'}|\cdot\ln(1+|E'|)$ because $\mathcal{K}_q$ contains all the keywords and $\mathcal{K}_{R'}\subseteq\mathcal{K}_q$. Therefore, if there is a Ham-Route R in G, we can find a route R' in G' such that its cost is at most $n+1$ and its score is at least $\sqrt{\frac{n-1}{|E|}}$.

On the other hand, we assume that there is a route R' in G' such that its cost is at most $n+1$ and its score is at least $\sqrt{\frac{n-1}{|E|}}$. According to Eq. (5), we have $\tau(R')$=$\sqrt{\frac{|\mathcal{K}_{R'}|}{|\mathcal{K}_q|}}$=$\sqrt{\frac{|\mathcal{K}_{R'}|}{|E|}}$, where $|\mathcal{K}_{R'}|$ is the number of keywords on route R'. R' should contain at least $n-1$ keywords if its score is at least $\sqrt{\frac{n-1}{|E|}}$. Accordingly, R' must have at least $n+1$ edges considering that each edge in G' contains one unique keyword and the edges connected to v_s and v_d have no keywords. Meanwhile, R' has at most $n+1$ edges because its cost is at most $n+1$ and the cost of each edge is 1. Therefore, R' has exactly $n+1$ edges. By removing v_s and v_d from R', we get one Ham-Route R in G.

It completes the proof. □

2.2.2 Non-additive Property

The route score in BCIR query is non-additive, i.e., the score of a route cannot be computed by adding the scores of its sub-routes.

Example 3. We take the BCIR query in Figure 2 for example and consider route R_1=$\langle s,v_1,d\rangle$ and its sub-routes R_{11}=$\langle s,v_1\rangle$ and R_{12}=$\langle v_1,d\rangle$. With Eq. (3), we have $\tau(R_1)$=0.903, $\tau(R_{11})$=0.707, and $\tau(R_{12})$=1.0. Obviously, the score of R_1 cannot be computed by summing the scores of its sub-routes, i.e., $\tau(R_1)\neq\tau(R_{11})+\tau(R_{12})$.

The non-additive property not only makes BCIR query more difficult than existing keyword coverage route queries [1], [2], [3], [4], [5] in which the route scores are additive, but also prevents us from leveraging existing techniques to process BCIR query.

3 Solution Overview

Considering the hardness of BCIR query, we propose three solutions for different application scenarios.

Scenario I: Requiring the optimal result
In this scenario, users request for the exact results. BCIR query is NP-hard, indicating that it is impossible to devise an exact solution of polynominal time complexity to process BCIR query. Nevertheless, it is still possible to evaluate all the candidate routes if the travel cost budget is small. Therefore, we propose an exact solution to BCIR query problem by designing effective pruning techniques to reduce the search space (Section 4).

Scenario II: Limiting response time
In this scenario, users wish to obtain the query results within a specified time limit. To satisfy this requirement, we propose the time-bounded solution (TBS) which receives as input a processing time limit and aims to maximize the score of the returned route within the time limit (Section 5).

Scenario III: Guaranteeing answer quality
Some users would like to sacrifice a certain degree of exactness to reduce the running time as long as the answer quality is guaranteed. To meet this requirement, we propose the error-bounded solution (EBS) which imposes an approximation error threshold on the returned route (Section 6).

4 Exact Solution

4.1 Algorithm Sketch for Exact Solution

BCIR query retrieves the optimal route with the largest score among all the candidate routes $\mathcal{C}^B_{s,d}$ that satisfy the travel cost budget B. Therefore, one straightforward solution for solving BCIR query problem is to evaluate all the candidate routes and select the route with the largest score as the query result. Algorithm 1 sketches this exact solution which explores the candidate routes by a depth-first expansion from the start location s. Initially, a partial route set U is created to store partial routes (i.e., routes starting from s but not reaching the destination d) during the query processing and R^* is used to record the current optimal route. The initialization of U and R^* will be discussed in Section 4.5. In each iteration, a partial route is selected from U to generate more partial routes by expanding the adjacent vertices of its end vertex. Once a partial route reaches the destination d and its score is larger than the current optimal route R^*, R^* is updated. The algorithm terminates until U is empty.

However, it is computationally expensive to conduct such an exploration because the number of candidate routes could be considerably large. To reduce the search space, we design effective pruning techniques to avoid checking those candidate routes that cannot be the optimal result as soon as possible (line 6 in Algorithm 1, cf. Sections 4.3 and 4.4).

4.2 Indexing

In order to facilitate the query processing, we build a *Shortest Route Index* and an *Inverted Index* for road network G.

Shortest Route Index: Shortest route computation will be frequently invoked to check whether a route is feasible when processing the BCIR query. In order to facilitate the shortest route computation, we build a Contraction Hierarchy (CH) [8] index, one of the most efficient index structures for shortest route computation, for road network G.

Inverted Index: In order to check which edges contain the query keywords, an inverted index is built for all the edges

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

Algorithm 1 Algorithm sketch for the exact solution

Input: BCIR query $q{=}(s, d, \mathcal{K}_q, B)$
Output: The optimal route R^*

```
Initialize partial route set U
Initialize optimal route R* by heuristics        ▷ Section 4.5
while U is not empty do
    select a partial route R from U
    generate new partial routes C_R based on R
    prune unpromising routes in C_R    ▷ Sections 4.3 and 4.4
    for R' ∈ C_R do
        if R' reaches the destination d then
            update R*
        else
            add R' to U
return R*
```

E. Inverted index is widely used for indexing documents by listing for each keyword those documents containing it. Table 3 is an example of the inverted index for the road network in Figure 2, where the numbers after the colon record the occurrences of the corresponding keywords.

TABLE 3
The inverted index for the edges in Figure 2

Keywords	Edge list
k_1	$\langle e_{s,1}:1\rangle$, $\langle e_{s,3}:1\rangle$, $\langle e_{1,2}:1\rangle$, $\langle e_{1,d}:2\rangle$
k_2	$\langle e_{s,1}:1\rangle$, $\langle e_{3,d}:2\rangle$
k_3	$\langle e_{s,3}:1\rangle$, $\langle e_{1,2}:1\rangle$, $\langle e_{3,d}:1\rangle$

Shortest Route Cost Caching: In addition, during the query processing of each BCIR query, we build a temporary hash table S to cache the shortest route costs, $\lambda(SR_{i,j})$, that have been computed. Then, $\lambda(SR_{i,j})$ can be obtained from S directly if it is in the hash table, which will further facilitate the query processing. The corresponding algorithm for looking up $\lambda(SR_{i,j})$ is detailed in Algorithm 2.

Algorithm 2 Look up the shortest route cost $\lambda(SR_{i,j})$

Input: Cached hash table S, start and end vertices v_i and v_j
Output: The shortest route cost $\lambda(SR_{i,j})$

```
λ(SR_{i,j}) ← ∞
if S contains λ(SR_{i,j}) then
    get λ(SR_{i,j}) from S
else
    compute SR_{i,j} by CH index
    for each vertex v_x on route SR_{i,j} do
        if S does not contain λ(SR_{i,x}) then
            add λ(SR_{i,x}) to S
return λ(SR_{i,j})
```

In what follows, we will detail the proposed pruning techniques (cf. Sections 4.3 and 4.4) and the query processing algorithm that combines these techniques (cf. Section 4.5).

4.3 Cost Pruning

The goal of cost pruning is to prune those partial routes that violate the cost budget B during the query processing. In other worlds, the pruned partial routes cannot expand to generate any candidate routes.

Given a partial route $R{=}\langle s, \ldots, v_i\rangle$, the set of candidate routes, from the start location s to the destination d, generated by expanding R is

$$\mathcal{C}_R = \{R' | R' = R \oplus R_x, R_x \in \mathcal{C}_{i,d}^{B'}\} \tag{6}$$

where $C_{i,d}^{B'}$ is the set of candidate routes from the end vertex v_i of R to the destination d (cf. Eq. (2)), $B'{=}B{-}\lambda(R)$, and $\oplus$ is an operator of concatenating two routes with duplicate vertices being removed.

The essential idea of cost pruning is to compute a lower bound cost $\lambda^-(R)$ for all the candidate routes $\mathcal{C}_R$ generated by expanding partial route R, and verify whether $\lambda^-(R)$ violates the cost budget B. First, we define the lower bound cost $\lambda^-(R)$.

Definition 4 (Lower Bound Cost). Given a partial route $R{=}\langle s, \ldots, v_i\rangle$, a lower bound cost $\lambda^-(R)$ satisfies $\lambda^-(R) \leq \lambda(R'), \forall R' {\in} \mathcal{C}_R$.

With the lower bound cost $\lambda^-(R)$, the partial route R should be pruned if $\lambda^-(R){>}B$.

Considering that the shortest route cost between two vertices is a lower bound for all the routes between them, we compute $\lambda^-(R)$ by

$$\lambda^-(R) = \lambda(R) + \lambda(SR_{i,d}) \tag{7}$$

where $SR_{i,d}$ is the shortest route from the end vertex v_i of R to the destination d. We then have the following lemma.

Lemma 1. Given a partial route R, the $\lambda^-(R)$ computed by Eq. (7) is a lower bound for the cost of each candidate route in $\mathcal{C}_R$.

Proof: For any candidate route $R' {\in} \mathcal{C}_R$, we assume that $R'{=}R \oplus R_x$, where $R_x {\in} \mathcal{C}_{i,d}^{B'}$. Since $SR_{i,d}$ is the shortest route from v_i to d, we have $\lambda(SR_{i,d}){\leq}\lambda(R_x), \forall R_x {\in} \mathcal{C}_{i,d}^{B'}$. Therefore, for any candidate route $R' {\in} \mathcal{C}_R$, we have $\lambda(R'){=}\lambda(R){+}\lambda(R_x){\geq}\lambda(R){+}\lambda(SR_{i,d}){=}\lambda^-(R)$. $\lambda^-(R)$ is thus a lower bound for the cost of each candidate route in $\mathcal{C}_R$. □

Example 4. We take the BCIR query in Figure 2 for example. Assuming that the partial route is $R{=}\langle s, v_2, v_1\rangle$, R should be pruned since $\lambda^-(R){=}\lambda(R) + \lambda(SR_{1,d}){=}10 + 5{>}B$, where the cost budget $B{=}12$.

Computing $\lambda(SR_{i,d})$ to compute $\lambda^-(R)$ will incurs a high computation cost. With the observation that the destination d is fixed for all the shortest routes $SR_{i,d}$, we precompute the shortest route costs to the destination d for those vertices that are within the travel cost budget from d via a reverse Dijkstra's algorithm. With these computed shortest route costs, $\lambda(SR_{i,d})$ can be retrieved directly instead of being computed from the scratch during the query processing.

4.4 Score Pruning

Intuitively, a partial route R should be pruned if all of its expanded candidate routes, $\mathcal{C}_R$, are not better than the current optimal route R^*. Therefore, we seek to compute an upper bound score, $\tau^+(R)$, for all the candidate routes $\mathcal{C}_R$ and exploit $\tau^+(R)$ to verify whether there exists a route in $\mathcal{C}_R$ that is better than the current optimal route R^*.

First, we define the upper bound score $\tau^+(R)$ as below.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

Definition 5 (Upper Bound Score). Given a partial route $R=\langle s,\ldots,v_i\rangle$, an upper bound score $\tau^+(R)$ satisfies $\tau^+(R)\geq\tau(R'), \forall R'\in\mathcal{C}_R$.

Then, we have the following lemma about score pruning.

Lemma 2. Given a partial route R and the current optimal route R^*, R should be pruned if $\tau^+(R)\leq\tau(R^*)$, where $\tau^+(R)$ is an upper bound score for all the candidate routes generated by expanding R, i.e., $\mathcal{C}_R$.

Proof: Since $\tau^+(R)$ is an upper bound score for all the candidate routes in $\mathcal{C}_R$, we have $\tau(R')\leq\tau^+(R)$ for each route $R'\in\mathcal{C}_R$. By inequality transition, we have $\tau(R')\leq\tau(R^*)$, indicating that the candidate routes generated by expanding partial route R cannot be better than the current optimal route R^*. Therefore, R can be pruned. □

4.4.1 Computing $\tau^+(R)$

The challenge for computing $\tau^+(R)$ is twofold. First, $\tau^+(R)$ should be computed based on a route set, $\mathcal{C}_R$, rather than a single route. Second, as discussed in Section 2.2.2, the route score in BCIR query is non-additive and we cannot sum the scores of routes R_x and R_y to obtain the score of route $R_x\oplus R_y$. With these two challenges, the proposed techniques for computing the upper bounds of route scores in existing studies cannot be applied to the BCIR query directly.

In this study, according to the definition of route score in Eq. (3), we compute the upper bound score $\tau^+(R)$ by the following equation:

$$\tau^+(R)=\frac{\sum_{k\in\mathcal{K}_q}(1+\ln(f_{k,R}+F^+(k,i,d,B')))\cdot(w_{k,q})}{\sqrt{\sum_{k\in\mathcal{K}_R}(1+\ln(f_{k,R}))^2\cdot\sum_{k\in\mathcal{K}_q}(w_{k,q})^2}} \quad (8)$$

where $f_{k,R}$ is the frequency of keyword k on route R, and $F^+(k,i,d,B')$ is a function that computes an upper bound for the occurrences of keyword k on all the sub-routes $\mathcal{C}^{B'}_{i,d}$ such that $f_{k,R_x}\leq F^+(k,i,d,B'), \forall R_x\in\mathcal{C}^{B'}_{i,d}$. The computation of $F^+(k,i,d,B')$ will be discussed in Section 4.4.2.

Then, we have the following lemma.

Lemma 3. The $\tau^+(R)$ computed by Eq. (8) is an upper bound for the score of each candidate route in $\mathcal{C}_R$, i.e., $\tau(R')\leq\tau^+(R), \forall R'\in\mathcal{C}_R$.

Proof: We use R' to represent any candidate route in $\mathcal{C}_R$ and let $R'=R\oplus R_x$, where sub-route $R_x\in\mathcal{C}^{B'}_{i,d}$. Accordingly, the frequency of keyword k on route R' is the sum of the frequencies of keyword k on routes R and R_x, i.e., $f_{k,R'}=f_{k,R}+f_{k,R_x}$.

According to the definition of route score in Eq. (3), we have

$$\begin{aligned}\tau(R') &= \frac{\sum_{k\in(\mathcal{K}_{R'}\cap\mathcal{K}_q)}(w_{k,R'})\cdot(w_{k,q})}{\sqrt{\sum_{k\in\mathcal{K}_{R'}}(w_{k,R'})^2\cdot\sum_{k\in\mathcal{K}_q}(w_{k,q})^2}}\\ &\leq \frac{\sum_{k\in\mathcal{K}_q}(w_{k,R'})\cdot(w_{k,q})}{\sqrt{\sum_{k\in\mathcal{K}_{R'}}(w_{k,R'})^2\cdot\sum_{k\in\mathcal{K}_q}(w_{k,q})^2}} \quad (9)\end{aligned}$$

For simplicity, we denote $w_q=\sqrt{\sum_{k\in\mathcal{K}_q}(w_{k,q})^2}$ and then have

$$\tau(R')\leq\frac{\sum_{k\in\mathcal{K}_q}(w_{k,R'})\cdot(w_{k,q})}{w_q\cdot\sqrt{\sum_{k\in\mathcal{K}_{R'}}(w_{k,R'})^2}} \quad (10)$$

On the one hand, for Eq. (10), we have

$$\sum_{k\in\mathcal{K}_q}(w_{k,R'})\cdot(w_{k,q})=\sum_{k\in\mathcal{K}_q}(1+\ln(f_{k,R}+f_{k,R_x}))\cdot(w_{k,q}) \quad (11)$$

Since $F^+(k,i,d,B')$ computes an upper bound for the occurrences of keyword k for all the sub-routes $R_x\in\mathcal{C}^{B'}_{i,d}$, we have

$$\sum_{k\in\mathcal{K}_q}(w_{k,R'})\cdot(w_{k,q})\leq\sum_{k\in\mathcal{K}_q}(1+\ln(f_{k,R}+F^+(k,i,d,B')))\cdot(w_{k,q}) \quad (12)$$

On the other hand, the denominator in Eq. (10) has the following derivation.

$$\begin{aligned}w_q\cdot\sqrt{\sum_{k\in\mathcal{K}_{R'}}(w_{k,R'})^2} &= w_q\cdot\sqrt{\sum_{k\in\mathcal{K}_{R'}}(1+\ln(f_{k,R}+f_{k,R_x}))^2}\\ &\geq w_q\cdot\sqrt{\sum_{k\in\mathcal{K}_R}(1+\ln(f_{k,R}))^2} \quad (13)\end{aligned}$$

Combining Eq. (10), Eq. (12) and Eq. (13), we have

$$\begin{aligned}\tau(R') &\leq \frac{\sum_{k\in\mathcal{K}_q}(w_{k,R'})\cdot(w_{k,q})}{w_q\cdot\sqrt{\sum_{k\in\mathcal{K}_{R'}}(w_{k,R'})^2}}\\ &\leq \frac{\sum_{k\in\mathcal{K}_q}(1+\ln(f_{k,R}+F^+(k,i,d,B')))\cdot(w_{k,q})}{w_q\cdot\sqrt{\sum_{k\in\mathcal{K}_R}(1+\ln(f_{k,R}))^2}}\\ &= \tau^+(R)\end{aligned}$$

Therefore, we have $\tau(R')\leq\tau^+(R)$ for any candidate route $R'\in\mathcal{C}_R$. Thus, the $\tau^+(R)$ computed by Eq. (8) is an upper bound for the score of any candidate route in $\mathcal{C}_R$. □

4.4.2 Computing $F^+(k,i,d,B')$

Given a partial route $R=\langle s,\ldots,v_i\rangle$, function $F^+(k,i,d,B')$ computes an upper bound for the occurrences of keyword k on each sub-route in $\mathcal{C}^{B'}_{i,d}$, where i and d are the subscripts of the end vertex v_i of R and the destination d, and $B'=B-\lambda(R)$. Since $F^+(k,i,d,B')$ will be invoked frequently during the query processing to compute the upper bound score, an efficient method for computing $F^+(k,i,d,B')$ is required.

One simple method for computing $F^+(k,i,d,B')$ is to evaluate the occurrences of keyword k, f_{k,R_x}, on each route $R_x\in\mathcal{C}^{B'}_{i,d}$ and report the maximum f_{k,R_x}. However, $\mathcal{C}^{B'}_{i,d}$ could be considerably large, making it time prohibitive to evaluate all the routes in $\mathcal{C}^{B'}_{i,d}$. Therefore, instead of evaluating all the routes in $\mathcal{C}^{B'}_{i,d}$, we utilize those edges on routes $\mathcal{C}^{B'}_{i,d}$ to directly estimate the maximum possible number of occurrences of keyword k on each sub-route $R_x\in\mathcal{C}^{B'}_{i,d}$.

With the end vertex v_i of R, destination d and the remaining cost budget B', the set of edges $E^{B'}_{i,d}$ that may appear on routes $\mathcal{C}^{B'}_{i,d}$ and contain keyword k is computed by

$$E^{B'}_{i,d}(k)=\{e_{f,g}|e_{f,g}\notin R\wedge k\in\mathcal{K}_{f,g}\wedge\Gamma(i,f,g,d)\leq B'\} \quad (14)$$

where function $\Gamma(i,f,g,d)=\lambda(SR_{i,f})+c_{f,g}+\lambda(SR_{g,d})$ computes the shortest travel cost from v_i to d when passing $e_{f,g}$.

In Eq. (14), inverted index is utilized to check whether edge $e_{f,g}$ contains query keyword k and Algorithm 2 is invoked to compute $\lambda(SR_{i,f})$. Note that we can get $\lambda(SR_{g,d})$ in Eq. (14) directly since a reverse Dijkstra'a algorithm has

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

been conducted at the beginning of query processing to compute the smallest cost to the destination d for each vertex (cf. the last paragraph in Section 4.3).

Straightforwardly, $F^+(k,i,d,B')$ can be computed by counting the total occurrences of k on all edges in $E^{B'}_{i,d}(k)$. Nonetheless, the upper bound $F^+(k,i,d,B')$ computed in this way could be very loose since $E^{B'}_{i,d}(k)$ covers much more edges than each route $R_x \in \mathcal{C}^{B'}_{i,d}$.

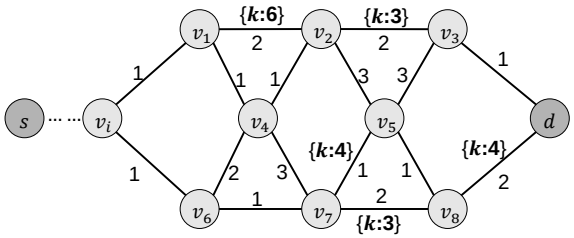


Fig. 3. Road network example for computing $F^+(k,i,d,B')$.

Example 5. We take Figure 3 for example and assume that the remaining travel cost budget is B'=6. With Eq. (14), the set of edges on routes $\mathcal{C}^{B'}_{i,d}$ and containing keyword k is $E^{B'}_{i,d}(k)=\{e_{1,2}, e_{2,3}, e_{5,7}, e_{7,8}, e_{8,d}\}$ whose total keyword occurrences is 20. However, the corresponding total cost is 9 which is larger than B'=6, indicating that the computed upper bound is loose.

To obtain a tighter $F^+(k,i,d,B')$, we compute the maximum number of keywords k that each route $R_x \in \mathcal{C}^{B'}_{i,d}$ could have by selecting a sub-set of edges from $E^{B'}_{i,d}(k)$ rather than using the whole set. To this end, we build an average keyword frequency histogram $\mathcal{H}$ for those edges in $E^{B'}_{i,d}(k)$ and leverage this histogram to compute $F^+(k,i,d,B')$.

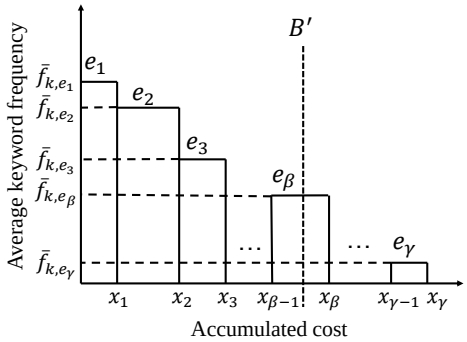


Fig. 4. Average keyword frequency histogram

First, with the frequency of keyword k on edge e, $f_{k,e}$, the corresponding average keyword frequency is $\bar{f}_{k,e}=\frac{f_{k,e}}{c_e}$, where c_e is the travel cost of edge e. All the edges in $E^{B'}_{i,d}(k)$ are then sorted based on the average keyword frequency. Without loss of generality, we assume that the sorted edges are $e_1, e_2, \ldots, e_\gamma$, where $\bar{f}_{k,e_i} \leq \bar{f}_{k,e_j}$ if $i<j$. The corresponding average keyword frequency histogram $\mathcal{H}$ for these sorted edges is illustrated in Figure 4, where the accumulated cost $x_i=x_{i-1}+c_{e_i}$ and $x_1=c_{e_1}$. Given the remaining travel cost budget B' and the average keyword frequency histogram $\mathcal{H}$, $F^+(k,i,d,B')$ is computed by

$$\begin{aligned} & F^+(k,i,d,B') \\ = \; & \bar{f}_{k,e_1} \cdot x_1 + \bar{f}_{k,e_2} \cdot (x_2 - x_1) + \cdots + \bar{f}_{k,e_\beta} \cdot (B' - x_{\beta-1}) \\ = \; & f_{k,e_1} + f_{k,e_2} + \cdots + \bar{f}_{k,e_\beta} \cdot (B' - x_{\beta-1}) \end{aligned} \quad (15)$$

where B' is assumed to be located at edge e_β, i.e., $x_{\beta-1} < B' \leq x_\beta$. We then have the following lemma.

Lemma 4. The $F^+(k,i,d,B')$ computed by using the average keyword frequency histogram $\mathcal{H}$ is an upper bound for the keyword frequency f_{k,R_x} of any route $R_x \in \mathcal{C}^{B'}_{i,d}$.

Proof: We first assume that the optimal route in route set $\mathcal{C}_R$ is R^* and the corresponding sub-route of R^* from vertex v_i to the destination d is R^*_x. Assuming that those edges containing keyword k on sub-route R^*_x are $E_{R^*_x}(k)=\{e_i, e_{i+1}, \ldots, e_\alpha\}$, we then have $E_{R^*_x}(k) \subseteq E^{B'}_{i,d}(k)$ since any edge that contains keyword k and is not in $E^{B'}_{i,d}(k)$ will violate the cost budget constraint (cf. Eq. (14)).

The frequency of keyword k on route R^*_x is $f_{k,R^*_x}=\sum_{e \in E_{R^*_x}(k)} f_{k,e}$ and the cost of R^*_x satisfies $\lambda(R^*_x) \leq B'$. In Figure 4, we record those edges on the left side of the cost budget B' or insects with B' as $E_{\mathcal{H}}=\{e_1, e_2, \cdots, e_\beta\}$. Then, for any edge e on route R^*_x, if $e \notin E_{\mathcal{H}}$, the average keyword frequency of e cannot be larger than any edge in $E_{\mathcal{H}}$. Therefore, the $F^+(k,i,d,B')$ computed based on $E_{\mathcal{H}}$ is an upper bound for the keyword frequency f_{k,R_x} of any route $R_x \in \mathcal{C}^{B'}_{i,d}$. □

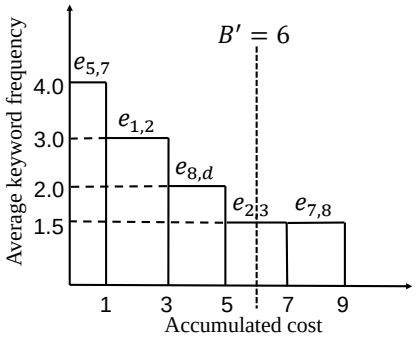


Fig. 5. Average keyword frequency histogram for Figure 3

TABLE 4
The edges $E^{B'}_{i,d}(k)$ containing keyword k in Figure 3

Edge	Cost	Keyword frequency	Average keyword frequency
$e_{1,2}$	2	6	3.0
$e_{2,3}$	2	3	1.5
$e_{5,7}$	1	4	4.0
$e_{7,8}$	2	3	1.5
$e_{8,d}$	2	4	2.0

Example 6. Continued with Example 5, the costs, keyword frequencies, and average keyword frequencies of these edges in $E^{B'}_{i,d}(k)$ are summarized in Table 4. The corresponding average keyword frequency histogram is illustrated in Figure 5. With Eq. (15), we have

$$\begin{aligned} F^+(k,i,d,B') &= f_{k,e_{5,7}} + f_{k,e_{1,2}} + f_{k,e_{8,d}} + \bar{f}_{k,e_{2,3}} \cdot (B' - 5) \\ &= 4 + 6 + 4 + 1.5 \cdot (6 - 5) \\ &= 15.5 \end{aligned}$$

4.5 Query Processing Algorithm

Algorithm 3 presents the algorithm for the exact solution. The initialization and processing procedures are elaborated below.

Initialization: Initially, a max-priority queue U is created to store partial routes during the query processing and a partial route R with the start location s is enqueued (lines 1-3). Meanwhile, route R^* stores the optimal route and is initialized to the shortest route from s to d, i.e., $SR_{s,d}$ (line

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

4). Note that R^* can be also initialized by other heuristic methods without modifying the algorithm. In addition, a reverse Dijkstra's algorithm is conducted to compute the shortest route cost of each vertex to the destination d (line 5).

Query Processing: In each iteration, the partial route R with the largest lower bound cost is dequeued from U (line 7). If the upper bound score of R is not larger than that of the current optimal route R^*, R is pruned (line 8). Otherwise, the end vertex v_i of R is computed (line 9). Each adjacent vertex v_j of v_i is then concatenated to R to generate a longer route R' if v_j has not been visited by R (lines 10-11). R' will be pruned if it is impossible to generate a candidate route (cost pruning, line 12). If R' survives from the cost pruning, arrives at the destination d and has a score larger than the current optimal route R^*, it is used to update R^* (lines 13-14). If R' is just a new partial route, its upper bound score $\tau^+(R')$ is computed (lines 15-16). If $\tau^+(R')$ is not larger than that of the current optimal route R^*, R' is pruned (line 17). Otherwise, R' is add to the priority queue U for further exploration (line 18). The algorithm terminates when U is empty (line 6) and the optimal route R^* is returned (line 19).

Algorithm 3 The exact solution

Input: BCIR query q=$(s, d, \mathcal{K}_q, B)$
Output: The optimal route R^*

```
1: Initialize U ← an empty priority queue
2: Initialize R ← ⟨s⟩
3: U.enqueue(R, 0)
4: R* ← SR_{s,d}
5: Compute λ(SR_{i,d}) for each v_i by reverse Dijkstra's algorithm
6: while U is not empty do
7:     R ← U.dequeue()
8:     if τ+(R) ≥ τ(R*) then                          ▷ score pruning
9:         v_i ←endVertex(R)
10:        for v_j ∈ adj(v_i) do
11:            R' ← R ⊕ {v_j}
12:            if λ−(R') ≤ B then                     ▷ cost pruning
13:                if v_j = d and τ(R') > τ(R*) then
14:                    R* ← R'      ▷ update the optimal route
15:                else
16:                    Compute upper bound score τ+(R')
17:                    if τ+(R') > τ(R*) then
18:                        U.enqueue(R', λ−(R'))
19: return R*
```

5 Time-Bounded Solution

In some applications, users would like to impose a constraint on the response time to avoid waiting for a long time. To meet this requirement, we propose time-bounded solution (TBS) which returns query results within the user-specified response time limit.

The idea of TBS is based on the observation that candidate routes $\mathcal{C}^B_{s,d}$ between two query locations s and d often share edges with each other. Therefore, it is possible to convert one candidate route to another by changing their sub-routes. One candidate route can be enhanced if we can replace its sub-routes with other sub-routes such that the new candidate route has a larger route score.

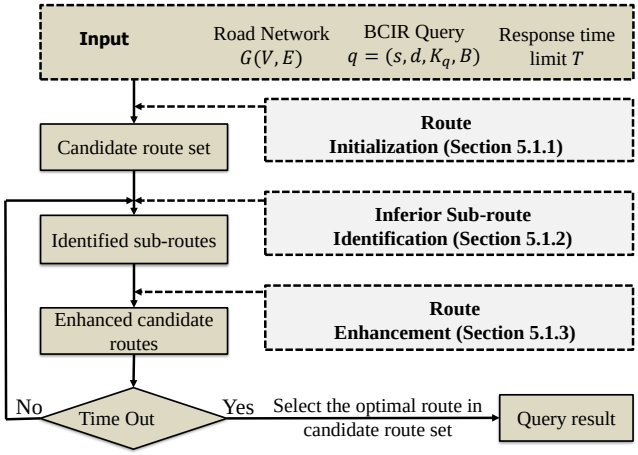


Fig. 6. Framework of time-bounded solution

Figure 6 illustrates the framework of TBS which takes as input the road network, the BCIR query and a response time limit, and returns an approximate query result. In this framework, we have three major components:

- **Routes Initialization** generates a set of candidate routes;
- **Inferior Sub-route Identification** randomly selects one candidate route from the generated candidate route set and identifies an inferior sub-route with limited contribution to the score of the selected candidate route for enhancement;
- **Route Enhancement** generates a new sub-route to replace the identified inferior sub-route and enhances the selected candidate route.

When response time limit permits, inferior sub-route identification and route enhancement repeat continuously. Finally, the optimal route in the candidate route set is returned as the query result. The details of the three components are elaborated in the subsequent sub-sections.

5.1 Routes Initialization

Route initialization generates a set of h candidate routes $\mathcal{C}$ so that time-bounded solution can further enhances these h candidate routes continuously until reaching the response limit.

The reason for generating h candidate routes rather than one is twofold. First, it is of high probability for one candidate route to get stuck in a local optimum, thus reducing the probability of obtaining the optimal route. Second, enhancing h candidate routes is faster to approach the optimal route than enhancing one. As illustrated in Figure 7 where the grey ellipse represents the whole search space, if only route R_1 is generated, it will be a long way to enhance R_1 to get the optimal route R^*. In contrast, if four routes R_1, R_2, R_3 and R_4 are generated, it should be fast to enhance R_4 to get the optimal route R^*.

The setting of h concerns two aspects. On the one hand, though a large h could have a good coverage of the search space, it also requires much time to generate and enhance the candidate route set. On the other hand, a small h cannot

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

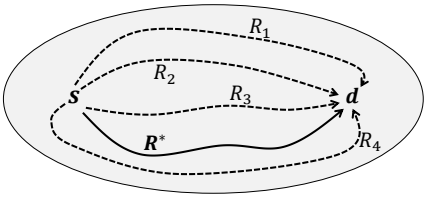


Fig. 7. Initialized candidate routes and search space.

reflect the advantage of route set initialization since it only covers a small search space. Therefore, we will tune h in the experiments to determine the appropriate value.

To generate h candidate routes that are distributed randomly over the whole search space, we propose two methods below.

A. Random Route Sampling

Random route sampling samples a candidate route by gradually expands edges from the start location s. Initially, a route R with the start location s is generated. In each expansion, one more adjacent edge $e_{i,j}$ is randomly selected to add to the end of R. To ensure that the added edge $e_{i,j}$ satisfies the cost budget, $e_{i,j}$ should satisfy $\lambda(R)+c_{i,j}+\lambda(SR_{j,d})\leq B$, where $SR_{j,d}$ is the shortest route from v_j to the destination d. Finally, R reaches the destination d and is returned as a candidate route.

Example 7. We take the BCIR query in Figure 2 for example. Initially, we have $R=\langle s\rangle$. There are three edges adjacent to s, i.e., $e_{s,1}$, $e_{s,2}$ and $e_{s,3}$. We randomly select one edge from the three edges and assume $e_{s,2}$ is selected, thus $R=\langle s, v_2\rangle$. Next, we consider the two adjacent edges $e_{2,1}$ and $e_{2,d}$ of the end vertex v_2 of R. Since $\lambda(R)+c_{2,1}+\lambda(SR_{1,d})=5+5+5>B=12$, only $e_{2,d}$ is feasible. Therefore, we have $R=\langle s, v_2, d\rangle$ which reaches the destination d, thus a candidate route.

B. Route Sampling by Adapted Nearest Neighbour Heuristics

Since random route sampling generates candidate routes without considering the query keywords on edges, the initialized candidate routes may have very small scores. In this case, it will take a long time to enhance the candidate routes. To deal with this issue, we propose to sample candidate routes by adapted nearest neighbours (NN) heuristics. Traditional NN heuristics method [9] starts from the start query location and gradually adds the nearest edge containing query keywords until reaching the destination. In this way, we can only generate one candidate route. However, time-bounded solution requires h candidate routes. Therefore, we adapt traditional NN heuristics method so as to generate h candidate routes.

The initialization of adapted NN heuristics method is the same as random route sampling. In each expansion, we compute the h nearest edges E^h containing query keywords for the end vertex v_i of R. Then, one edge is randomly selected from E^h and added to the end of R. The remaining issue is how to compute the h nearest edges for vertex v_i. For each BCIR query q, we compute all those edges, E_q, that contain at least one query keyword in $\mathcal{K}_q$ and satisfy the cost budget B, i.e.,

$$E_q = \{e_{f,g} | e_{f,g} \in E \wedge (\mathcal{K}_{f,g} \cap \mathcal{K}_q \neq \emptyset) \wedge \Gamma(s,f,g,d) \leq B\} \quad (16)$$

where function $\Gamma(s,f,g,d)=\lambda(SR_{s,f})+c_{f,g}+\lambda(SR_{g,d})$ is the same as that in Eq. (14). For ease of discussion, we call E_q positive edges. Then, we select the h nearest edges E^h for vertex v_i from E_q.

Example 8. We still take the BCIR query in Figure 2 for example and initialize $R=\langle s\rangle$. First, we have positive edges $E_q=\{e_{s,1}, e_{s,3}, e_{1,2}, e_{1,d}\}$. If h=2, we have $E^h=\{e_{s,1}, e_{s,3}\}$. Assuming that $e_{s,3}$ is selected, we have $R=\langle s, v_3\rangle$. Then, we consider the two nearest edges containing query keywords for v_3. However, we cannot find a feasible edge containing query keywords to add to v_3. Therefore, R goes directly to the destination d and generates a candidate route $R=\langle s, v_3, d\rangle$.

5.2 Inferior Sub-route Identification

After generating h candidate routes $\mathcal{C}$, the next step is to refine these h candidate routes until reaching the response time limit T. Each time, a candidate route R is randomly selected from $\mathcal{C}$. The goal of inferior sub-route identification is to find a sub-route of R for enhancement.

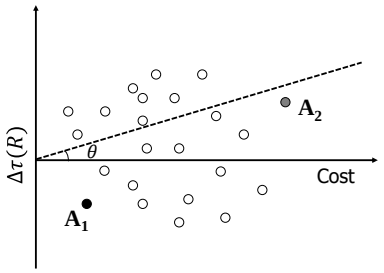


Fig. 8. The distribution of all sub-routes of R

Figure 8 plots the distribution of all the sub-routes of R with respect to their costs and $\Delta\tau(R)$ values, where $\Delta\tau(R)$ value of sub-route $R_{i,j}$ is computed by

$$\Delta\tau(R) = \tau(R) - \tau(R - R_{i,j}) \quad (17)$$

where $\tau(R-R_{i,j})$ computes the route score after removing sub-route $R_{i,j}$ from R. Obviously, $\Delta\tau(R)$ quantifies the change of route score after removing the sub-route $R_{i,j}$.

Since it is difficult to bound the value of $\frac{\Delta\tau(R)}{\lambda(R_{i,j})}$, we compute the corresponding angle $\theta(R_{i,j})$ in Figure 8 by

$$\theta(R_{i,j}) = \arcsin \frac{\Delta\tau(R)}{\lambda(R_{i,j})} \quad (18)$$

Angle $\theta(R_{i,j})$ quantifies the ratio between $\Delta\tau(R)$ and the cost of sub-route $R_{i,j}$. According to Eq. (18), we have $\theta(R_{i,j})\in(-90°, 90°)$ since $\lambda(R_{i,j})$ is always positive while $\Delta\tau(R)$ could be both positive and negative.

Intuitively, to increase the score of candidate route R, we need to replace those sub-routes whose costs are large while $\Delta\tau(R)$ values are small, i.e., sub-routes with small θ. To this end, we devise two methods as below.

A. Ranking-based Inferior Sub-route Identification

Ranking-based identification orders all sub-routes according to the θ value. First, the sub-route with the minimum θ value is selected. If we can replace this sub-route with a better one, this sub-route is identified. Otherwise, the second sub-route is checked. ranking-based identification repeats this operation until one sub-route is identified. For example, In Figure 8, ranking-based inferior sub-route identification will

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

first select the sub-route corresponding to the black point A_1 since it has the smallest angle θ.

B. Threshold-based Inferior Sub-route Identification

In threshold-based inferior sub-route identification, we set a threshold ϕ for θ and compute the longest sub-route $R_{i,j}$ of R such that

$$R_{i,j} = \arg \max_{R_{i,j} \in R \wedge \theta(R_{i,j}) \leq \phi} \lambda(R_{i,j}) \quad (19)$$

In Figure 8, assuming that the threshold is the dashed line, threshold-based inferior sub-route identification will select the sub-route corresponding to the grey point A_2 since it locates below the threshold line and has the largest cost. During the query processing, we first set ϕ=-80 and increase it by 10 gradually until a sub-route is identified.

5.3 Route Enhancement

After identifying the inferior sub-route $R_{i,j}$ of candidate route R, we need to find a new sub-route to replace $R_{i,j}$ such that the new candidate route is better than R. Actually, any route between v_i and v_j such that its cost is less than B'=$B-(\lambda(R)-\lambda(R_{i,j}))$ can be used to replace $R_{i,j}$. However, it is computationally prohibitive to evaluate all these routes to select the optimal one to replace $R_{i,j}$.

Intuitively, adding those edges containing query keywords to an existing candidate route is of high probability to increase the route score. Therefore, given sub-route $R_{i,j}$ of candidate route R, we compute a candidate route set $\mathcal{C}'$ by evaluating all the positive edges in E_q.

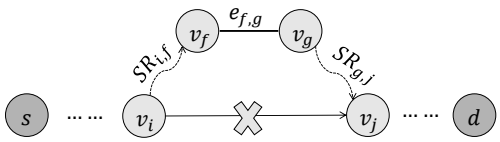


Fig. 9. Route enhancement in TBS.

For each edge $e_{f,g}$∈E_q, according to Figure 9, the new candidate route generated by replacing sub-route $R_{i,j}$ with respect to edge $e_{f,g}$ is

$$R_{new} = R_{s,i} \oplus SR_{i,f} \oplus e_{f,g} \oplus SR_{g,j} \oplus R_{j,d} \quad (20)$$

where $R_{s,i}$ is the sub-route (from s to v_i) of R, $R_{j,d}$ is the sub-route (from v_j to d) of R, $SR_{i,f}$ is the shortest route from vertex v_i to vertex v_f, and $SR_{g,j}$ is the shortest route from vertex v_g to vertex v_j.

The algorithm for computing the candidate route set $\mathcal{C}'$ is presented in Algorithm 4 which receives as input a candidate route R with its sub-route $R_{i,j}$, and the positive edges E_q. Initially, an empty route set $\mathcal{C}'$ is created to store potential candidate routes (line 1). In each iteration, one edge from E_q that is not in R is evaluated (line 2), and a new route R_{new} is computed (line 3). The new route R_{new} may contain duplicate edges since the two shortest routes, $SR_{i,f}$ and $SR_{g,j}$, may contain edges in sub-routes $R_{s,i}$ and $R_{j,d}$. After removing duplicate edges (line 4), R_{new} is added to the candidate route set $\mathcal{C}'$ if $\lambda(R_{new})$≤B and $\tau(R_{new})$>$\tau(R)$(lines 5-6). By evaluating all the edges in E_q, algorithm generates a set of candidate routes $\mathcal{C}'$ whose scores are larger than R (line 7).

With the computed candidate routes $\mathcal{C}'$, we have two methods to update the original candidate route R.

Algorithm 4 Enhance_route(R, $R_{i,j}$, E_q)

Input: Candidate route R and its sub-route $R_{i,j}$, and positive edges E_q
Output: A set of new candidate route $\mathcal{C}'$

1: $\mathcal{C}' \leftarrow \emptyset$ ▷ Store the candidate routes
2: **for** $e_{f,g} \in E_q - E_R$ **do**
3: $R_{new} \leftarrow R_{s,i} \oplus SR_{i,f} \oplus e_{f,g} \oplus SR_{g,j} \oplus R_{j,d}$
4: remove duplicate edges of R_{new}
5: **if** $\lambda(R_{new}) \leq B$ and $\tau(R_{new}) > \tau(R)$ **then**
6: add R_{new} to $\mathcal{C}'$
7: **return** $\mathcal{C}'$

A. Score-aware Enhancement

In score-aware enhancement, the optimal route in $\mathcal{C}'$ is selected to replace R in $\mathcal{C}$.

B. Randomness-aware Enhancement

Since score-aware enhancement only considers the optimal route in $\mathcal{C}'$, it is of high probability to get the local optimum. Therefore, we introduce randomness-aware enhancement in which one candidate route is randomly selected from $\mathcal{C}'$ to replace the original candidate route R.

6 Error-Bounded Solution

In some cases, users would like to sacrifice the quality of query answer to reduce processing time as long as the worst case is under control, i.e., the approximation error of the query result is bounded. Given an approximate route $\hat{R}$, the corresponding approximate error ϵ is computed by

$$\epsilon = \frac{\tau(R^*) - \tau(\hat{R})}{\tau(R^*)} \quad (21)$$

where R^* is the optimal route. Obviously, we have $\epsilon \in [0, 1]$.

Formally, an error-bounded solution (EBS) returns approximate query results such that the given approximation error threshold is guaranteed. In the exact solution, we can relax the upper bound score based on the approximation error threshold ϵ to search an approximate route instead of the optimal one, i.e., the following lemma.

Lemma 5. Given a partial route R and the current optimal route R_c^*, if $(1-\epsilon)\cdot\tau^+(R)$≤$\tau(R_c^*)$, R can be pruned while guaranteeing that the approximation error of final result is at most ϵ.

Proof: For each candidate route R'∈$\mathcal{C}_R$ generated by expanding R, we have $\tau(R')$≤$\tau^+(R)$. With the assumption that $(1-\epsilon)\cdot\tau^+(R)$≤$\tau(R_c^*)$, we have $(1-\epsilon)\cdot\tau(R')$≤$\tau(R_c^*)$. Assuming that the optimal route is R^*, R^* will be pruned if and only if there exists a route R_c^* such that $(1-\epsilon)\cdot\tau(R^*)$≤$\tau(R_c^*)$. Accordingly, we have $\tau(R^*) - \epsilon\cdot\tau(R^*) \leq \tau(R_c^*)$, i.e., $\frac{\tau(R^*)-\tau(R_c^*)}{\tau(R^*)}$≤$\epsilon$. Therefore, the approximation error of route R_c^* is at most ϵ. □

Therefore, we can adapt the exact solution to an error-bounded solution by using the relaxed score pruning. In addition, we also employ the time-bounded solution to improve the efficiency of EBS, thus EBS-T. For EBS-T, we call a time-bounded solution with a time limit T after Line 4 in Algorithm 3. By doing this, we can obtain a good route efficiently, thus improving the score pruning. The setting of time limit T will be discussed in the experiments.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

Though EBS and EBS-T can further reduce the number of iterations in Algorithm 3 by using a tighter score bound, they still require a high time cost to compute the final query results if the approximation error threshold is small and the travel cost budget is large, which will be analyzed in the experiments.

7 Experiments

7.1 Data Sets and Setup

We evaluate the performance of the proposed solutions over three data sets, i.e., the road networks of New York (NY), California (CA) and United Kingdom (UK), where NY data set is used as default. All data sets are extracted from the OpenStreetMap (OSM)[1] and their details are summarized in Table 5. The text descriptions of edges are extracted from the text descriptions of those POIs residing on them. Distance is used as the travel cost in the experiments.

The settings of those parameters in the experiments are listed in Table 6, where the default values are highlighted in bold. Particularly, the travel cost budget is specified by a deviation ratio $\mu \geq 0$ from the shortest route between two query locations, i.e., $B=(1+\mu)\cdot\lambda(SR_{s,d})$. We randomly generate 50 queries for each setting and compute the average value of the corresponding results for plotting. All algorithms were implemented in Java and run on a PC equipped with Intel(R) Core(TM) i3-2100 CPU @3.10 GHz, 8 GB RAM.

TABLE 5
Statistics of data sets

Data set	#vertices	#edges	average #keywords
NY	6, 393	13, 885	4.25
CA	90, 870	202, 250	2.46
UK	338, 838	738, 610	2.65

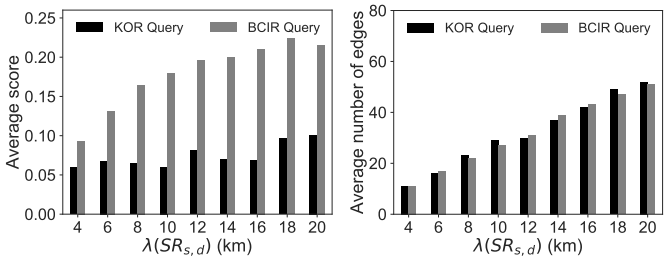
TABLE 6
Parameter setting

Parameter	Meaning	Value
$\lvert\mathcal{K}_q\rvert$	number of query keywords	1, **2**, 3, 4, 5
$\lambda(SR_{s,d})$	the shortest distance (km) from s to d	4, 6, 8, **10**, 12, 14, 16, 18, 20
μ	deviation ratio	0.05, 0.10, **0.15**, 0.20, 0.25
h	number of initialized candidate routes	1, 3, 5, 9, **11**, 13, 15, 17, 19
T	time threshold in TBS	0.5, **1.0**, 1.5, 2.0, 2.5 (sec)
ϵ	approximation error threshold in EBS	0.0, 0.1, 0.3, **0.5**, 0.7

7.2 Experimental Results

7.2.1 BCIR query vs. KOR query

In order to compare BCIR query with KOR query [1], we need to adapt KOR query. The cost budget is the same for KOR query and BCIR query. For each edge $e_{i,j}\in E$, we set the objective score in KOR query as $OS(e_{i,j})=\frac{|\mathcal{K}_{i,j}|}{c_{i,j}}$, where $|\mathcal{K}_{i,j}|$ and $c_{i,j}$ are the number of keywords and travel cost of $e_{i,j}$, respectively. Then, the adapted KOR query computes the optimal route such that (1) it covers the given query keywords, (2) its cost is less than the travel cost budget, and (3) it has the maximum objective score. Differently, BCIR query is targeted at computing the optimal route within the travel cost budget such that it is most textually relevant to the query keywords. Figure 10(a) shows the scores (i.e., textual relevance) of the routes returned by KOR query and BCIR query. The route scores of KOR query are much smaller than that of BCIR query. Therefore, existing route planning applications using keyword coverage query cannot effectively find the route that is the most textually relevant to the query keywords. In addition, Figure 10(b) presents the number of edges in returned routes for two queries. With the increase of the shortest distance between two query locations, both queries return routes of more edges. However, there is no obvious difference between the sizes of edges for two queries.



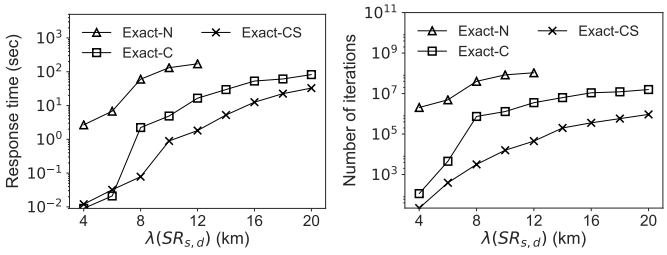
(a) Score (b) Number of edges

Fig. 10. Effect of the shortest distance $\lambda(SR_{s,d})$ between two query locations, where travel cost budget $B=(1+0.15)\cdot\lambda(SR_{s,d})$

7.2.2 Exact Solution

We evaluate the performance of exact solution while using no pruning (Exact-N), cost pruning (Exact-C) and cost-score pruning (Exact-CS), where Exact-N does not involve any pruning techniques, Exact-C employs the cost pruning, and Exact-CS leverages both cost pruning and score pruning.

Pruning effectiveness: Figure 11 shows the response time and the number of iterations in Algorithm 1 while using different pruning settings. Note that, we set the upper bound running times of Exact-N and Exact-C to 200 seconds since the real running times could be much longer than 200 seconds when the travel cost budget is large. As suggested by Figure 11, the cost pruning reduces the number of iterations by several orders of magnitude and the score pruning can further iterations by around one order of magnitude. After applying both cost and score pruning methods, the number of iterations is greatly reduced, thus making the algorithm efficient for BCIR queries of small travel cost budgets.



(a) Response time (b) Number of iterations

Fig. 11. Varying the shortest distance $\lambda(SR_{s,d})$, where cost budget $B=(1+0.15)\cdot\lambda(SR_{s,d})$

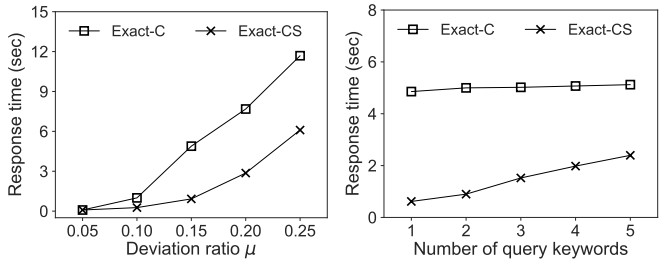
Varying deviation ratio μ: Figure 12(a) illustrates the response time of exact solution while varying the travel cost deviation ratio μ, where the cost of the shortest route is $\lambda(SR_{s,d})$=10 km. With the increase of μ, the response time

1. https://www.openstreetmap.org

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

of Exact-C increases rapidly. In contrast, if both cost pruning and score pruning are used, i.e., Exact-CS, the increase rate of response time is moderate.

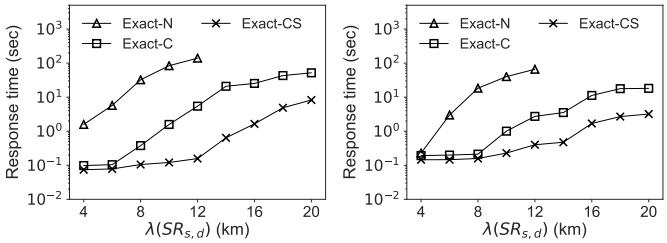
Varying the number of query keywords $|\mathcal{K}_q|$: Figure 12(b) illustrates the results while varying the number of query keywords. With the increase of the number of query keywords, the response time of Exact-CS gradually increases since it needs more time to compute the upper bound score. In contrast, the response time of Exact-C keeps almost the same because it only utilizes cost pruning and does not need to compute upper bound score, thus being less dependent on the number of query keywords.



(a) Response time (b) Number of iterations

Fig. 12. Varying the deviation ratio μ and number of query keywords

Scalability test: We also evaluate the performance of the exact solution on two large data sets, i.e., CA and UK data sets. As illustrated in Figure 13, exact solution still performs well when the road network have hundreds of thousands of vertices. In fact, the response time on these two data sets is smaller than that on NY data set because the query locations are randomly selected and could be out of city on the two large data sets. In general, the density of road network out of city is sparse, thus reducing the number of candidate routes in BCIR queries.

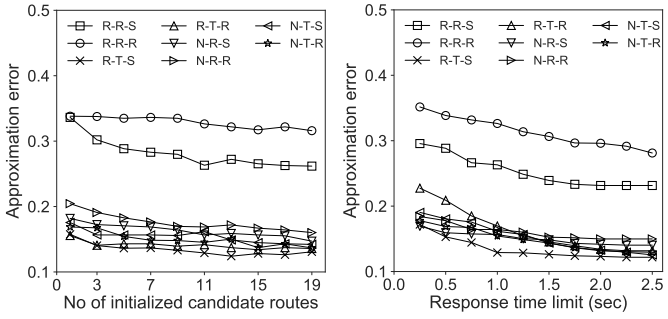


(a) California (b) United Kingdom

Fig. 13. Varying the shortest distance $\lambda(SR_{s,d})$ on CA and UK data sets

7.2.3 Time-Bounded Solution

Method combination comparison: Time-bounded solution has three components and each component has two methods, thus 8 combinations in total. For ease of presentation, we record these combinations as X-Y-Z where X represents routes initialization method, including random route sampling (R) and adapted NN heuristics route sampling (N); Y represents inferior sub-route identification method, including ranking-based identification (R) and threshold-based identification (T); and Z represents enhancement method, including score-aware enhancement (S) and randomness-aware enhancement (R). For example, R-R-S indicates that random route sampling, ranking-based identification and score-aware enhancement are used. Figure 14 presents the approximation error (cf. Eq. (21)) of returned routes using different combinations. According to Figure 14(a), we set the number of initialized candidate routes h=11. Since combination R-T-S performs the best among all the combinations, we only report the results of R-T-S in the subsequent plots. In addition, Figure 15 illustrates the distribution of response time for all combinations. Obviously, random route sampling consumes less time than adapted NN heuristics route sampling in route initialization.



(a) Varying h (T=1 sec) (b) Varying response time (h=11)

Fig. 14. Varying the number of initialized candidate routes h and the response time limit, where $\lambda(SR_{s,d})$=15 (km).

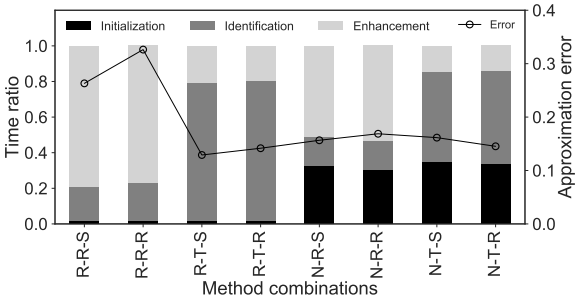
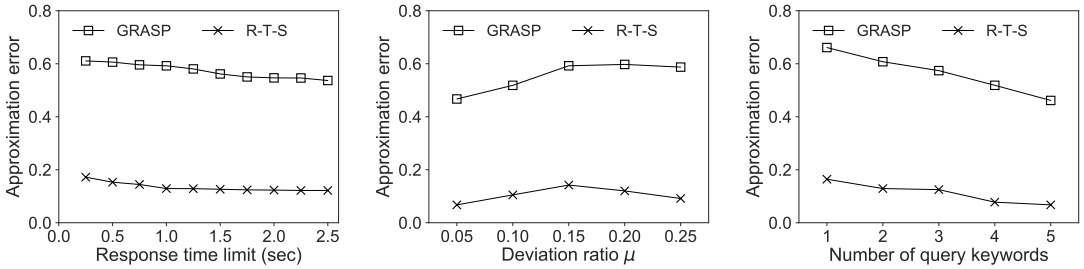


Fig. 15. The distribution of processing time among three components

GRASP vs. TBS: To demonstrate the performance of time-bounded solution, we adapted the GRASP [10] solution for arc orienteering problem (AOP) to our BCIR query. Similar to TBS, GRASP also gradually refines the current optimal route by generating new candidate routes. However, GRASP generates candidate routes by searching the whole road network and needs to pre-compute the shortest routes for all pairs of vertices. Since it is inapplicable to pre-compute all pair shortest routes for large-scale road networks, we utilize CH index to compute the shortest route between two locations for GRASP online. Figure 16(a) illustrates the results of R-T-S and GRSAP while varying the response time limit T. According to Figure 16(a), R-T-S greatly outperforms GRASP. The performance of GRASP is not satisfying because it needs to search the whole road network in each iteration and cannot iterate enough times to improve the quality of query results. Figures 16(b) and 16(c) present the results while varying the deviation ratio μ and the number of query keywords $|\mathcal{K}_q|$. With the increase of μ, the approximation error first increases because the search space increases. However, the approximation error then decreases when μ is larger than 0.15 because the increase of the optimal route score slows down. Differently, with the increase of $|\mathcal{K}_q|$, the approximation error always decreases. Still, R-T-S outperforms GRASP in all cases.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2721408, IEEE Transactions on Knowledge and Data Engineering

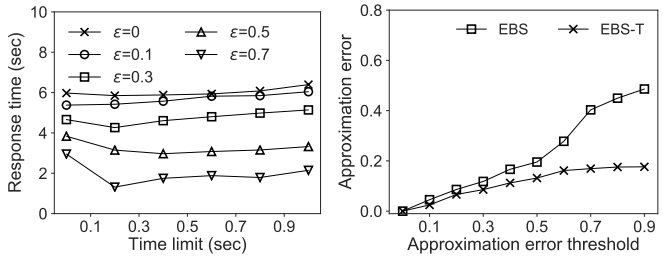


(a) Varying T (b) Varying μ (c) Varying $|\mathcal{K}_q|$

Fig. 16. Varying the processing time limit T, the deviation ratio μ, and the number of query keywords $|\mathcal{K}_q|$, where the travel cost budget B=$(1+\mu)\cdot\lambda(SR_{s,d})$ and $\lambda(SR_{s,d})$=15 (km)

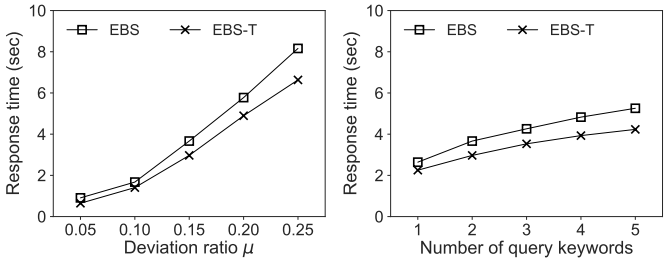
7.2.4 Error-Bounded Solution

Figure 17(a) illustrates the response time while varying the time limit T of time-bounded solution in EBS-T. According to Figure 17(a), the response time reduces the most when setting T=0.2 second. Figure 17(b) presents the approximation error of EBS and EBS-T (T=0.2 second) while varying the approximation error threshold ϵ. With the increase of ϵ, the approximation errors of both solutions increase. However, the increase speed of EBS-T is much smaller than that of EBS. In addition, we also evaluate the performance of EBS and EBS-T while varying the deviation ratio μ and the number of query keywords. As suggested by Figure 18, the response time will increase with the increase of μ and $|\mathcal{K}_q|$ and EBS-T performs better than EBS.



(a) Varying T (b) Varying ϵ

Fig. 17. The response time while varying time limit T, and the approximation error while varying the approximation error threshold ϵ, where the travel cost budget $B=(1+0.15)*$ 15 (km)



(a) Varying μ (b) Varying $|\mathcal{K}_q|$

Fig. 18. The response time while varying the deviation ratio μ and the number of query keywords $|\mathcal{K}_q|$, where ϵ=0.5 and the travel cost budget B=$(1+\mu)*$ 15 (km)

8 Related Work

BCIR query is relevant to *spatial keyword query* and *preference route query*, which are elaborated as below.

8.1 Spatial Keyword Query

Spatial keyword query retrieves geo-textual objects based on the textual relevance and spatial proximity, where geo-textual objects refers to these objects that contain both spatial locations and keyword descriptions, e.g., POIs and geo-tagged comments. A comprehensive survey on spatial keyword query can be found in [11]. According to the granularity of query results, existing studies on spatial keyword query can be classified into three categories, i.e., individual geo-textual objects queries [12], [13], [14], [15], [16], [17], [18], group geo-textual objects queries [4], [19], [20], [21], [22], [23], [24], [25], [26], and region of geo-textual objects queries [27], [28], [29], [30]. Concretely, individual geo-textual object queries, e.g., range spatial keyword queries and top-k spatial keyword queries, retrieve individual geo-textual objects such that each returned object satisfies the query keyword requirement independently. Differently, group geo-textual objects queries search for a set of geo-textual objects that collectively satisfy the query keyword requirement. In region of geo-textual objects queries, a region, e.g., rectangle and circle, is predefined and the objective is to retrieve a region which has the largest number of geo-textual objects relevant to the given query keywords.

Both spatial keyword query and BCIR query are partly motivated by the constantly increasing geo-textual data. However, spatial keyword query is generally targeted at locating geo-textual objects according to user-specified query keywords and the spatial proximity between query locations and the locations of geo-textual objects. Therefore, spatial keyword query generally has no routing function. In contrast, BCIR is designed for finding a route that is relevant to the query keywords within a certain travel cost budget.

8.2 Preference Route Query

In addition to shortest route query, many preference route queries have been proposed to provide personalized route query services, including *arc orienteering problem* [31], [10], [32] *object coverage route query* [33], [34], [35], [36], [37], [38], and *keyword coverage route query* [1], [2], [3], [4], [5], [39].

Arc Orienteering Problem (AOP): Arc orienteering problem is a variant of the classical orienteering problem (OP). In orienteering problem, each vertex is associated with a score and each edge is associated with a cost, and the objective is to find a route such that the cost is less than a given cost budget while the total score is maximized. Different from orienteering problem, the score in AOP problem is associated with edge instead of vertex. The objective of AOP

is also to search a route such that its total score is maximized and its cost is within a cost budget. BCIR is relevant to AOP because both problems have constrained cost and the score is associated with edge. However, the route score in BCIR query captures the textual relevance between the text description of the whole route and query keywords rather than summing up the separate scores of all edges on the route in AOP.

Object Coverage Route Query: In this line of studies, the whole set of geo-textual objects, e.g., POIs, is classified into different categories. The goal of this kind of query is to find a route that covers at least one object in some required categories of geo-textual objects with the smallest travel distance. In addition, the required categories can be visited sequentially [34], [38] or partially sequentially [35].

Keyword Coverage Route Query: Keyword coverage route query retrieves routes to cover user-specified query keywords. Cao et al. proposed the keyword-aware optimal route (KOR) query [1] in which each edge of the road network was associated with a cost and a score. Given a cost budget, KOR query is targeted at finding a route that covers a set of user-specified query keywords and maximizes the objective score within the given cost budget. Taking into account the weights of different keywords of query objects, Zeng et al. [4] proposed an optimization problem to optimize the keyword coverage on the required route. However, it is non-trivial to compute the accurate weights for the keywords of each object. Yao et al. [2] proposed approximate keyword route query to retrieve routes that cover the query keywords according to an approximate string similarity function rather than perfect match. Li et al. [3] contribute an extension to the KOR query by searching all those routes that are not dominated by others. In addition, the Route of Interest (ROI) [5] query is proposed to search for the optimal route that collects the most number of query keywords. Different from BCIR query, ROI query does not compute the textual relevance between routes and query keywords. Another relevant work is [39] which introduced the problem of identifying *Streets of Interest* (SOIs) to identify individual street segments that have dense relevant POIs around them.

BCIR query is different from the existing studies on preference route query in two folds. First, BCIR computes the textual relevance between the text description of the entire route and query keywords directly while existing studies on preference route queries usually focus on the keyword coverage or score collection. Second, if each road edge is associated with a score, the score is usually pre-computed in existing route queries, which makes it inflexible for users to describe their personal preferences. In contrast, users can easily specify the query keywords to describe their preferences in BCIR query.

9 Conclusion

In this paper, we propose the BCIR query to retrieve the route that is most textually relevant to the user-specified query keywords within a travel cost budget. To efficiently process BCIR queries, we propose an exact solution with effective pruning techniques and two approximate solutions regarding the response time and the quality of results, respectively. As demonstrated via extensive experiments, the proposed solutions achieve satisfying performance over different data sets. BCIR provides a new type of route query that can be applied in various applications ranging from route planning to location-aware recommendation. In addition, the future extensions may include 1) searching the top-k optimal routes rather than the optimal one, 2) finding a BCIR route that allows duplicate vertices and edges, 3) removing the query location constraints and retrieving the optimal route over the whole road network.

Acknowledgments

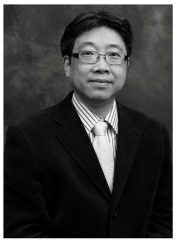
The work described in this paper was partially supported by the funding for Project of Strategic Importance provided by the Hong Kong Polytechnic University (Project Code: 1-ZE26). This work was also partially supported by grant GRF 152196/16E from the Hong Kong RGC. Wengen Li and Jiannong Cao were supported by the NSFC key project under grant No. 61332004 and the University's Support for Application of Major Research Funding provided by the Hong Kong Polytechnic University (Project Code: 1-BBA1). Wengen Li and Jihong Guan were supported by the National Natural Science Foundation of China under grant No. 61373036 and the Program of Shanghai Subject Chief Scientist under grant No. 15XD1503600. Shuigeng Zhou was supported by the Key Projects of Fundamental Research Program of Shanghai Municipal Commission of Science and Technology under grant No. 14JC1400300. Jihong Guan is the correspondence author.

References

[1] X. Cao, L. Chen, G. Cong, and X. Xiao, "Keyword-aware optimal route search," *PVLDB*, vol. 5, no. 11, pp. 1136–1147, 2012.
[2] B. Yao, M. Tang, and F. Li, "Multi-approximate-keyword routing in GIS data," in *SIGSPATIAL*, 2011, pp. 201–210.
[3] Y. Li, W. Yang, W. Dan, and Z. Xie, "Keyword-aware dominant route search for various user preferences," in *DASFAA*, 2015, pp. 207–222.
[4] Y. Zeng, X. Chen, X. Cao, S. Qin, M. Cavazza, and Y. Xiang, "Optimal route search with the coverage of users' preferences," in *IJCAI*, 2015, pp. 2118–2124.
[5] W. Li, J. Cao, J. Guan, M. L. Yiu, and S. Zhou, "Retrieving routes of interest over road networks," in *WAIM*, 2016, pp. 109–123.
[6] D. Quercia, R. Schifanella, and L. M. Aiello, "The shortest path to happiness: recommending beautiful, quiet, and happy routes in the city," in *HT*, 2014, pp. 116–125.
[7] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Comput. Surv.*, vol. 38, no. 2, 2006.
[8] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *WEA*, 2008, pp. 319–333.
[9] D. Deng, C. Shahabi, and U. Demiryurek, "Maximizing the number of worker's self-selected tasks in spatial crowdsourcing," in *SIGSPATIAL*, 2013, pp. 314–323.
[10] W. Souffriau, P. Vansteenwegen, G. V. Berghe, and D. V. Oudheusden, "The planning of cycle trips in province of eat flanders," *Omega*, vol. 39, no. 2, pp. 209–213, 2011.
[11] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *PVLDB*, vol. 6, no. 3, pp. 217–228, 2013.
[12] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *PVLDB*, vol. 2, no. 1, pp. 337–348, 2009.
[13] A. Cary, O. Wolfson, and N. Rishe, "Efficient and scalable method for processing top-k spatial boolean queries," in *SSDBM*, 2010, pp. 87–95.
[14] J. B. Rocha-Junior and K. Nørvåg, "Top-k spatial keyword queries on road networks," in *EDBT*, 2012, pp. 168–179.

[15] K. Zheng, H. Su, B. Zheng, S. Shang, J. Xu, J. Liu, and X. Zhou, "Interactive top-k spatial keyword queries," in *ICDE*, 2015, pp. 423–434.
[16] L. Chen, G. Cong, X. Cao, and K. Tan, "Temporal spatial-keyword top-k publish/subscribe," in *ICDE*, 2015, pp. 255–266.
[17] Z. Li, K. C. K. Lee, B. Zheng, W. Lee, D. L. Lee, and X. Wang, "Ir-tree: An efficient index for geographic document search," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 4, pp. 585–599, 2011.
[18] C. Zhang, Y. Zhang, W. Zhang, and X. Lin, "Inverted linear quadtree: Efficient top k spatial keyword search," in *ICDE*, 2013, pp. 901–912.
[19] D. Zhang, B. C. Ooi, and A. K. H. Tung, "Locating mapped resources in web 2.0," in *ICDE*, 2010, pp. 521–532.
[20] X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi, "Efficient processing of spatial group keyword queries," *ACM Trans. Database Syst.*, vol. 40, no. 2, p. 13, 2015.
[21] T. Guo, X. Cao, and G. Cong, "Efficient algorithms for answering the m-closest keywords query," in *SIGMOD*, 2015, pp. 405–418.
[22] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *SIGMOD*, 2011, pp. 373–384.
[23] C. Long, R. C. Wong, K. Wang, and A. W. Fu, "Collective spatial keyword queries: a distance owner-driven approach," in *SIGMOD*, 2013, pp. 689–700.
[24] Y. Gao, J. Zhao, B. Zheng, and G. Chen, "Efficient collective spatial keyword query processing on road networks," *IEEE Trans. Intelligent Transportation Systems*, vol. 17, no. 2, pp. 469–480, 2016.
[25] K. Deng, X. Li, J. Lu, and X. Zhou, "Best keyword cover search," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 61–73, 2015.
[26] L. Zhang, X. Sun, and H. Zhuge, "Density based collective spatial keyword query," in *SKG*, 2012, pp. 213–216.
[27] D. Choi, C. Chung, and Y. Tao, "A scalable algorithm for maximizing range sum in spatial databases," *PVLDB*, vol. 5, no. 11, pp. 1088–1099, 2012.
[28] J. Liu, G. Yu, and H. Sun, "Subject-oriented top-k hot region queries in spatial dataset," in *CIKM*, 2011, pp. 2409–2412.
[29] Y. Tao, X. Hu, D. Choi, and C. Chung, "Approximate maxrs in spatial databases," *PVLDB*, vol. 6, no. 13, pp. 1546–1557, 2013.
[30] X. Cao, G. Cong, C. S. Jensen, and M. L. Yiu, "Retrieving regions of interest for user exploration," *PVLDB*, vol. 7, no. 9, pp. 733–744, 2014.
[31] Y. Lu and C. Shahabi, "An arc orienteering algorithm to find the most scenic path on a large-scale road network," in *SIGSPATIAL*, 2015, pp. 46:1–46:10.
[32] C. Verbeeck, P. Vansteenwegen, and E.-H. Aghezzaf, "An extension of the arc orienteering problem and its application to cycle trip planning," *Transportation research*, vol. 68, p. 64C78, 2014.
[33] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S. Teng, "On trip planning queries in spatial databases," in *SSTD*, 2005, pp. 273–290.
[34] M. Sharifzadeh, M. R. Kolahdouzan, and C. Shahabi, "The optimal sequenced route query," *VLDB J.*, vol. 17, no. 4, pp. 765–787, 2008.
[35] H. Chen, W. Ku, M. Sun, and R. Zimmermann, "The multi-rule partial sequenced route query," in *ACM-GIS*, 2008, p. 10.
[36] X. Ma, S. Shekhar, H. Xiong, and P. Zhang, "Exploiting a page-level upper bound for multi-type nearest neighbor queries," in *ACM-GIS*, 2006, pp. 179–186.
[37] M. N. Rice and V. J. Tsotras, "Parameterized algorithms for generalized traveling salesman problems in road networks," in *SIGSPATIAL*, 2013, pp. 114–123.
[38] M. N. Rice and V. J. Tsotras, "Engineering generalized shortest path queries," in *ICDE*, 2013, pp. 949–960.
[39] D. Skoutas, D. Sacharidis, and K. Stamatoukos, "Identifying and describing streets of interest," in *EDBT*, 2016, pp. 437–448.

Wengen Li received his BS degree from the Department of Computer Science and Technology, Tongji University, in 2011. He is currently a joint PhD candidate at the Department of Computer Science and Technology, Tongji University and the Department of Computing, Hong Kong Polytechnic University. His research interests include spatio-textual data query and urban data mining.



Jiannong Cao received the BSc degree from Nanjing University, China, in 1982, and the MSc and PhD degrees from Washington State University, USA, in 1986 and 1990, all in computer science. He is currently a chair professor and the head of the Department of Computing at Hong Kong Polytechnic University. His research interests include parallel and distributed computing, computer networks, mobile and pervasive computing, fault tolerance, and middle-ware.

Jihong Guan received the bachelor's degree from Huazhong Normal University in 1991, the master's degree from Wuhan Technical University of Surveying and Mapping (merged into Wuhan University since 2000) in 1991, and the PhD degree from Wuhan University in 2002. She is currently a professor in the Department of Computer Science and Technology, Tongji University, Shanghai, China. Before joining Tongji University, she served in the Department of Computer, Wuhan Technical University of Surveying and Mapping from 1991 to 1997, as an assistant professor and an associate professor (since August 2000), respectively. She was an associate professor (2000-2003) and a professor (Since 2003) in the School of Computer, Wuhan University. Her research interests include databases, data mining, distributed computing, bioinformatics, and geographic information systems (GIS).

Man Lung Yiu received the bachelor's degree in computer engineering and the PhD degree in computer science from the University of Hong Kong in 2002 and 2006, respectively. Prior to his current post, he worked at Aalborg University for three years starting in the Fall of 2006. He is now an associate professor in the Department of Computing, the Hong Kong Polytechnic University. His research focuses on the management of complex data, in particular query processing topics on spatio-temporal data and multi-dimensional data.

Shuigeng Zhou received the bachelor's degree from Huazhong University of Science and Technology (HUST) in 1988, the master's degree from the University of Electronic Science and Technology of China (UESTC) in 1991, and the PhD degree in computer science from Fudan University, Shanghai, China, in 2000. He is currently a professor in the School of Computer Science, Fudan University. He served in Shanghai Academy of Spaceflight Technology from 1991 to 1997, as an engineer and a senior engineer (since 1995), respectively. He was a postdoctoral researcher in the State Key Lab of Software Engineering, Wuhan University from 2000 to 2002. His research interests include data management, data mining and bioinformatics.