KORS: Keyword-aware Optimal Route Search System

Xin Cao[§] Lisi Chen[§] Gao Cong[§] Jihong Guan[†] Nhan-Tue Phan[§] Xiaokui Xiao[§]

[§] *School of Computer Engineering, Nanyang Technological University, Singapore*
{xcao@e., lchen012@e., gaocong@, phan0032@e., xkxiao@}ntu.edu.sg

[†] *Department of Computer Science & Technology, Tongji University, China*
jhguan@tongji.edu.cn

***Abstract*—We present the Keyword-aware Optimal Route Search System (KORS), which efficiently answers the KOR queries. A KOR query is to find a route such that it covers a set of user-specified keywords, a specified budget constraint is satisfied, and an objective score of the route is optimized. Consider a tourist who wants to spend a day exploring a city. The user may issue the following KOR query: "find the most popular route such that it passes by *shopping mall*, *restaurant*, and *pub*, and the travel time to and from her hotel is within 4 hours."**

KORS provides browser-based interfaces for desktop and laptop computers and provides a client application for mobile devices as well. The interfaces and the client enable users to formulate queries and view the query results on a map. Queries are then sent to the server for processing by the HTTP post operation. Since answering a KOR query is NP-hard, we devise two approximation algorithms with provable performance bounds and one greedy algorithm to process the KOR queries in our KORS prototype. We use two real-world datasets to demonstrate the functionality and performance of this system.

I. Introduction

Identifying a preferable route in a road network is an important problem that finds applications in map services. For example, map applications like Baidu Lvyou (http://lvyou.baidu.com/) and Yahoo Travel (http://travel.yahoo.com) offer tools for trip planning. However, the routes that they provide are collected from users and are thus pre-defined. This is a significant deficiency since there may not exist any pre-defined route that meets the needs of users. The existing solutions (e.g., [3]–[5]) for trip planning or route search are often insufficient in offering the flexibility for users to specify their requirements on the route. We use the following example to illustrate the insufficiency of existing applications and solutions.

Example 1: Consider a user who wants to spend a day exploring a city with which she is not familiar. She would like to visit a shopping mall, a restaurant, and a pub. Meanwhile, she is not willing to spend too much time on the road. The following query might be posed: "Find the most popular route to and from my hotel such that it passes by *shopping mall*, *restaurant*, and *pub*, and the time spent on the road in total is within 4 hours." □

The example query above has two hard constraints: 1) the points of interests preferred by the user, as expressed by a set of keywords that should be covered in the route (e.g., shopping mall, restaurant and pub); 2) a budget constraint (e.g., travel time) that should be satisfied by the route. The query aims to identify the optimal route under the two hard constraints, such that an objective score is optimized (e.g., route popularity). In general, the budget constraint and the objective score can be of different types, such as travel duration, travel distance, route popularity, travel budget, etc. We consider two different attributes for budget constraint and objective score because users often need to balance the trade-off of two aspects when planning trips. For example, a popular route may be quite expensive, or a route with the shortest length is of little interests. In the example query, it is likely that the most popular route requires traveling time more than 4 hours. Hence, a route searching system should be able to allow users to set their preferences according to their requirements.

We refer to the aforementioned type of queries as *Keyword-aware Optimal Route query*, denoted as KOR [1]. Formally, a KOR query is defined over a road network graph $\mathcal{G}$, and the input to the query consists of five parameters, v_s, v_t, ψ, Δ, and f, where v_s is the source location of the route in $\mathcal{G}$, v_t is the target location, ψ is a set of keywords, Δ is a budget limit, and f is a function that calculates the objective score of a route. The query returns a path R in $\mathcal{G}$ starting at v_s and ending at v_t, such that R minimizes $f(R)$ under the constraints that R satisfies the budget limit Δ and passes through locations that cover the query keywords in ψ.

In this demonstration, we illustrate the Keyword-aware Optimal Route Search System (KORS) which efficiently retrieves the optimal route according to user requirements specified in the KOR query. We use the route popularity as the objective score and the travel distance as the budget score in our KORS prototype (Note that route popularity can be estimated by the number of users traveling on a route, obtained from the user traveling histories recorded in sources such as GPS trajectories or Flickr photos [2]).

We develop browser-based user interfaces for conventional desktop and laptop computers. We also provide an Android client for mobile devices. They enable users to formulate their

queries and view the relevant objects using Google Maps. On the server side, we model the dataset as a graph. Queries are sent from the browser or the client to the server by the standard HTTP post operation. The problem of solving KOR queries can be shown to be NP-hard. Hence, we devise two approximation algorithms with provable performance guarantees and one greedy algorithm in our KORS prototype. Two real-world datasets collected from Flickr (http://www.flickr.com) and Foursquare (http://foursquare.com) are used to show both the effectiveness and efficiency of this system.

This rest of the demonstration proposal is organized as follows. Section 2 introduces the data model and the formal definition of queries. Section 3 presents the architecture and design of KORS. Finally, Chapter 4 offers the demonstration details.

II. Data Model and Query

A. Data Model

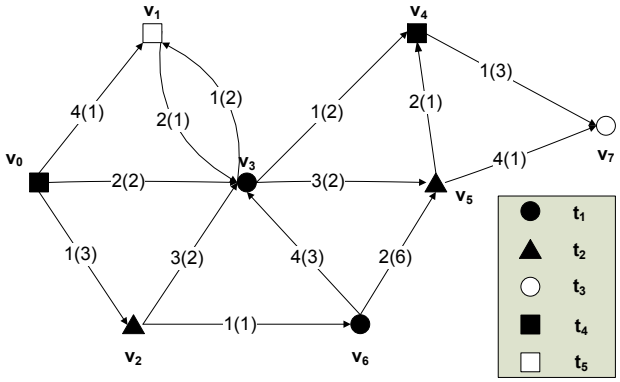

Fig. 1. Example of $\mathcal{G}$

We organize the road network as a graph $\mathcal{G} = (V, E)$, which consists of a set of nodes V and a set of edges $E \subseteq V \times V$. Each node $v \in V$ represents a location associated with a set of keywords denoted by $v.\psi$; each edge (v_i, v_j) in E represents a directed route from v_i to v_j in V.

Figure 1 shows an example of the graph $\mathcal{G}$. We consider five keywords (t_1–t_5), and each keyword is represented by a distinct shape. For simplicity, each node contains a single keyword.

B. Keyword-aware Optimal Route Search Query

Definition 1: Route. A route $R = (v_0, v_1, ..., v_n)$ is a path such that R goes through v_0 to v_n sequentially, following the relevant edges in $\mathcal{G}$. □

We define the optimal route based on two attributes on each edge (v_i, v_j): 1) one attribute is used as the *objective value* of this edge (route popularity used in KORS), and it is denoted by $o(v_i, v_j)$, and 2) the other attribute is used as the *budget value* of this edge (travel distance used in KORS), which is denoted by $b(v_i, v_j)$.

Definition 2: Objective Score and Budget Score: Given a route $R = \langle v_0, v_1, ..., v_n \rangle$, the objective score of R is defined as the sum of the objective values of all the edges in R, i.e.,

$$\mathsf{OS}(R) = \sum_{i=1}^{n} o(v_{i-1}, v_i),$$

likewise, the budget score is defined as the sum of the budget values of all the edges in R, i.e.,

$$\mathsf{BS}(R) = \sum_{i=1}^{n} b(v_{i-1}, v_i).$$

□

Intuitively, a *Keyword-aware Optimal Route* (KOR) query is to find an optimal route from a source to a target in a road network graph $\mathcal{G}$ such that the route covers all the query keywords, its budget score satisfies a given constraint, and its objective score is optimized. Formally, we define the KOR query as follows:

Definition 3: Keyword-aware Optimal Route (KOR) Query. Given $\mathcal{G}$, the keyword-aware optimal route query $\mathcal{Q} = \langle v_s, v_t, \psi, \Delta \rangle$, where v_s is the source location, v_t is the target location, ψ is a set of keywords, and Δ specifies the budget limit, aims to find the route R starting at v_s and ending at v_t (i.e.,$\langle v_s, \cdots, v_t \rangle$) such that

$$\begin{aligned} R = \quad & \arg\min_R \mathsf{OS}(R) \\ \text{subject to} \quad & \psi \subseteq \bigcup_{v \in R} (v.\psi) \\ & \mathsf{BS}(R) \leq \Delta \end{aligned}$$

□

Example 2: In the example graph in Figure 1, on each edge, the score inside a bracket is the budget value, and the other number is the objective value. Given a query $\mathcal{Q} = \langle v_0, v_7, \{t_1, t_2, t_3\}, 8 \rangle$, the optimal route is $R_{opt} = \langle v_0, v_3, v_4, v_7 \rangle$ with objective score $\mathsf{OS}(R_{opt}) = 4$ and budget score $\mathsf{BS}(R_{opt}) = 7$. If we set Δ to 6, the optimal route becomes $R_{opt} = \langle v_0, v_3, v_5, v_7 \rangle$ with $\mathsf{OS}(R_{opt}) = 9$ and $\mathsf{BS}(R_{opt}) = 5$. □

The problem of solving KOR queries is NP-hard. We omitted the detailed proof which could be found elsewhere [1].

III. KORS Prototype

A. Architecture of KORS

We develop the graphical user interfaces on both computers and mobile platforms (using Android-based smartphones). KORS adopts the browser-server model for desktop and laptop computers, and it adopts the client-server model for mobile devices. The architecture is shown in Figure 2. Users submit their queries through the web browser or the Android client application, and the queries are then sent to the server for processing. After the queries are processed, the results are sent back and displayed on Google Maps in the users' browser or client. On the Server side, since answering a KOR query is NP-hard, the query processor in KORS includes two approximation algorithms with performance bound and one greedy algorithm. We also do some pre-processing to accelerate the

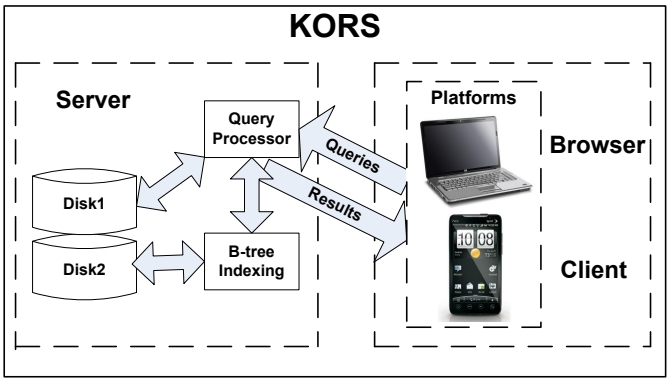

Fig. 2. KORS Architecture

algorithms, and the results could be accessed by the query processor.

In our KORS prototype, the route popularity is expressed as the objective score, and the travel distance is expressed as the budget score. We omit the details of computing the popularity scores for edges which could be found in the work [1]. Note that we can pick up any two attributes to define the optimal route depending on different applications.

B. Broswer/Client Side

The browser side in computers and the client side in Android mobile devices provide interfaces to users for generating queries and viewing the returned routes. This component presents a map and provides interactions with the map using the Google Maps API.

When generating a query, users specify a start location, a target location, a budget constraint, and input a set of keywords that describes their requirements to form a query. On the mobile platform, if a user does not point the locations, her current location is be detected by the mobile device and used as the start and target locations. Since we provides three algorithms for processing KOR queries, users are required to specify which algorithm they would like to select (OSScaling by default). They can also set the number of routes to be returned (1 by default). The query is then sent to the server for processing.

After the query is processed by the server, the routes found by a certain algorithm are returned and displayed on the map. Users can click the locations in a route shown in the map for more detailed information such as the text description.

C. Server

The web server in KORS is built using FastCGI (http://www.fastcgi.com) and Apache. Once a query is received by the Apache web server, the query processor expressed as a fastcgi implemented in C++ is called to find the routes.

1) Data Storage: The graph $\mathcal{G}$ is stored as an adjacency list and is maintained in the memory. In order to accelerate the query processing, we do some pre-processing: we find the following two paths for each pair of nodes (v_i, v_j): 1) the path with the smallest route popularity score; 2) the path with the smallest travel distance. We store these route popularity scores and travel distances in matrices. We also use an inverted file to organize the word information of nodes. An inverted file index has two main components: 1) A vocabulary of all distinct words appearing in the descriptions of nodes (locations), and 2) A posting list for each word t that is a sequence of identifiers of the nodes whose descriptions contain t. We use B^+-tree to index the inverted file (the B^+-tree index structures are disk resident).

2) Query Processor: There may exists many feasible routes for a given KOR query, namely, routes satisfying the keywords and the travel distance constraint. The goal of KORS is to find routes that approximates the optimal result in a short query time. Specifically, KORS provides two approximation algorithms with performance guarantees and one greedy algorithm.

Queries are sent from the browser or the client to the server by the HTTP post operation. Other user requirements are also attached with the query: 1) the number of routes to be returned; 2) algorithm type selection: the approximation algorithms with performance bounds (return more accurate results with longer query time, i.e., OSScaling and BucketBound) or the greedy algorithm (returned routes may not satisfy the constraints but quite fast, i.e., Greedy). After a query is received by the Apache web server, the FastCGI module will pass the query parameters to query processor. The processor then selects a corresponding algorithm to find answers to the query according to users' requirements. Finally, the results are sent back to users and displayed on the map using Google Maps API.

OSScaling **algorithm:** We first scale the route popular score on each edge to an integer by a parameter ϵ to obtain a scaled graph. In the scaled graph, each partial route is represented by a "label", which records the query keywords already covered by the partial route, the scaled popularity score, the original popularity score, and the travel distance of the route. At each node, we maintain a list of "useful" labels corresponding to the partial routes that go to that node. Starting from the source node, we keep creating new partial routes by extending the current "best" partial route using labels, until all the potentially useful labels on the target node are generated. Finally, the route represented by the label with the best popularity score at the target node is returned. We prove that the algorithm returns routes with popularity scores no worse than $\frac{1}{1-\epsilon}$ times of that of the optimal route.

BucketBound **algorithm:** The algorithm BucketBound improves on the algorithm OSScaling, and is more efficient than OSScaling. The algorithm can always return a route whose popularity score is at most β ($\beta > 1$ is a parameter) times of the one found by OSScaling. The algorithm divides the traversed partial routes into different "buckets" according to their best possible popularity scores. This enables us to develop a novel way to detect if a feasible route (covering all query keywords and satisfying the budget limit) is in the same bucket with one found by OSScaling. When we find a feasible route that shares the same bucket with the route found

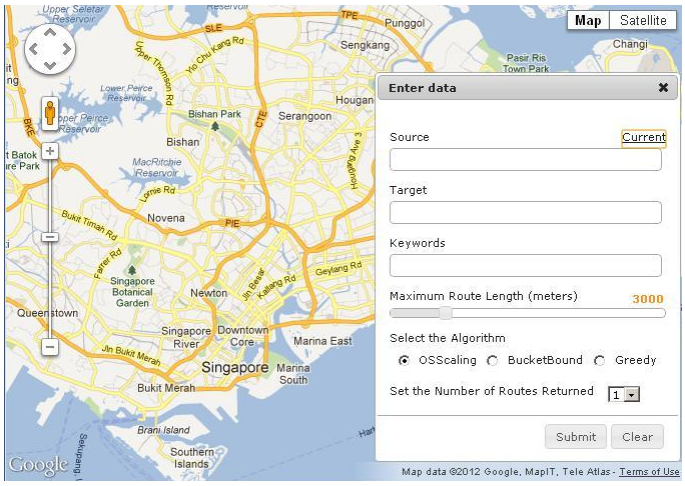

Fig. 3. Desktop/Laptop Interface

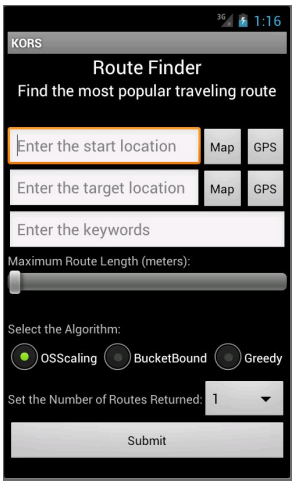

Fig. 4. Android Client

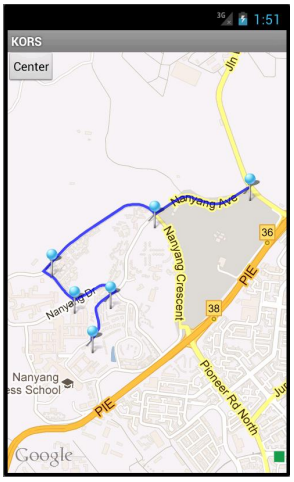

Fig. 5. Route Display

by OSScaling, we return it as the result. This algorithm avoids checking all feasible routes, thus being more efficient.

Greedy **algorithm:** The algorithm Greedy starts from the source node, and keeps selecting the next best one according to a certain strategy until finally the target node is reached. The strategy of selecting the next node takes into account all the three constraints simultaneously: a) the node contains uncovered query keywords; and b) the best route that can be generated after including this node into the current partial route is expected to have a small popularity score and fulfill the travel distance constraint. This algorithm may generate a route that violates the two hard constraints of KOR: covering all query keywords and satisfying the budget constraint.

Presentation of the complete details of the algorithms can be found in the work [1].

IV. Demonstration and Details

The interfaces of KORS are developed for desktop/laptop computers and Android-based smartphones as shown in Figures 3 and 4, respectively. Users specify the keywords in the "Keywords" text box. To specify the source and target location, users can either enter the name of a location or clicking a location on Google Maps (the latitude and longitude of the location as obtained using the Google Maps API). On the mobile devices, by clicking the button "GPS," the user's current location can be detected and used as either the source or the target location. The button "Map" allows users to specify the source and target locations by clicking on the map. Users then must specify the travel distance constraint using the slider bar. They also need to choose an algorithm (i.e., OSScaling, BucketBound and Greedy) for processing the query, and set the number of routes to be returned. The returned routes are displayed on the map. Figure 5 shows an example route on Android smartphones, which is from Nanyang Technological University to the road Jalan Bahar. Users can click a location on the map to see detailed text descriptions.

We use two real-world data sets for the demonstration. The first dataset is crawled from Foursquare in the region of Singapore. It consists of 7,883 points of interest, each of which contains a latitude and longitude, its categories (restaurant, mall, etc.), and a description provided by users who checked into the point of interest. The second one is collected from Flickr in the region of new York City in the United States, which contains over 1.5 million photos. Each photo is associated with a set of user-annotated tags and the latitude and the longitude of the place where the photo was taken. We utilize a clustering method to group the photos into locations. We associate each location with tags obtained by aggregating the tags of all photos in that location after removing the noisy tags, such as tags contributed by only one user. Finally, we obtain 5,199 locations in total. The demonstration will show that KORS is able to process the KOR queries effectively and efficiently over both data sets.

Acknowledgement

Gao Cong was supported in part by was supported in part by a grant awarded by a Singapore MOE AcRF Tier 2 Grant (ARC20/12). Jihong Guan was supported in part by the Shuguang Scholar Program of Shanghai Education Foundation. Xiaokui Xiao was supported in part by the Nanyang Technological University under SUG Grant M58020016, and by the Agency for Science, Technology, and Research (Singapore) under SERC Grant 102-158-0074.

References

[1] X. Cao, L. Chen, G. Cong, and X. Xiao. Keyword-aware optimal route search. *PVLDB*, 5(11):1136–1147, 2012.
[2] Z. Chen, H. T. Shen, and X. Zhou. Discovering popular routes from trajectories. In *ICDE*, pages 900–911, 2011.
[3] R. Levin, Y. Kanza, E. Safra, and Y. Sagiv. Interactive route search in the presence of order constraints. *PVLDB*, 3(1):117–128, 2010.
[4] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng. On trip planning queries in spatial databases. In *SSTD*, pages 273–290, 2005.
[5] M. Sharifzadeh, M. R. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. *VLDB J.*, 17(4):765–787, 2008.