WISER: A Web-based Interactive Route Search System for Smartphones

Roi Friedman
Technion, Israel Institute of Technology
roi7fri@gmail.com

Itsik Hefez
Technion, Israel Institute of Technology
itsikhefez@gmail.com

Yaron Kanza*
Technion, Israel Institute of Technology
kanza@cs.technion.ac.il

Roy Levin*
Technion, Israel Institute of Technology
royl@cs.technion.ac.il

Eliyahu Safra
Technion, Israel Institute of Technology
safraeli@gmail.com

Yehoshua Sagiv*
Hebrew University
sagiv@cs.huji.ac.il

ABSTRACT

Many smartphones, nowadays, use GPS to detect the location of the user, and can use the Internet to interact with remote location-based services. These two capabilities support online navigation that incorporates search. In this demo we presents WISER—a system for Web-based Interactive Search en Route. In the system, users perform *route search* by providing *(1)* a target location, and *(2)* search terms that specify types of geographic entities to be visited. The task is to find a route that minimizes the travel distance from the initial location of the user to the target, via entities of the specified types. However, planning a route under conditions of uncertainty requires the system to take into account the possibility that some visited entities will not satisfy the search requirements, so that the route may need to go via several entities of the same type. In an interactive search, the user provides feedback regarding her satisfaction with entities she visits during the travel, and the system changes the route, in real time, accordingly. The goal is to use the interaction for computing a route that is more effective than a route that is computed in a non-interactive fashion.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*

General Terms

Algorithms, Design

Keywords

Route Search, Path Planning, Navigation, Android

1. INTRODUCTION

The recent rapid growth in the popularity of smartphones, and other mobile devices, with an embedded GPS sensor, initiated a change in the way ordinary users carry out navigation tasks. Increasingly, navigation applications are replacing traditional printed maps. However, current applications are typically designed for tasks such as presenting a map, searching for an address (local search) or finding a route between two locations. Such applications are incapable of dealing with complex route-planning tasks where the route should go via several types of entities.

Example 1.1. *A user, Alice, is attending a conference abroad, and she wishes to plan a route from the conference center (her current location), to her hotel. Along the way, she wants to* (1) *stop by an ATM to withdraw cash,* (2) *go by a toy store to buy some gifts for her children, and* (3) *stop at a vegetarian restaurant for eating dinner. She can conduct ordinary local searches using her smartphone. This will aid her in locating some nearby ATMs, toy stores and vegetarian restaurants. However, combining these search results to create an effective route—from her current location to her destination—is a difficult task.*

A route search, such as the one in Example 1.1, is specified by a query that contains *(1)* a start location (usually the location of the user, given by the GPS), *(2)* a target location, and *(3)* a set of search terms specifying the types of geographical entities to be visited. The task is to compute a route that begins at the starting location, arrives at the target location, and goes via the geographical entities that are specified by the search terms. In many scenarios, it is also desired to specify constraints on the order by which the entities should be visited. For instance, in Example 1.1, Alice may desire to eat only after completing her other tasks. Several papers have shown how to formulate route-search queries and how to answer them effectively, in such cases, yet, without taking uncertainty into account (see [1, 2, 9]).

Uncertainty in route search means that only upon arrival at the entity the user knows whether the entity satisfies her. For instance, in Example 1.1, upon arrival at the toy store, Alice may discover that the cash-register lines are too long, the store is closed, the toys are not what she was looking for, or that the geographic entity was incorrectly deemed a toy store. She may also find out that, on her way, she passed close to some other potential stores; however, now there are no more toy stores nearby. Hence, finding an alternative toy store may now require heading back to previously vis-

*The work of these authors was partially supported by The Israeli Ministry of Science and Technology (Grant 3-6472).

WWW 2012, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1229-5/12/04.

ited locations or proceeding farther away from her current position, thus lengthening her travel distance.

One way to deal with uncertainty, is by assigning relevance scores to objects of the dataset so that the score of each entity indicates how relevant is the entity to the search terms specified in the query. Answering a route-search query then becomes an issue of achieving a proper trade-off between choosing a route that is as short as possible to one that passes via the most highly-ranked objects [3, 6, 7].

The recent ubiquitousness of smartphones that contain a GPS sensor and also have a constant (or highly frequent) Internet connection, supports a more advanced approach for dealing with uncertainty. Instead of providing, in advance, a pre-calculated static route, the system can gradually construct the route while interacting with the user. We refer to such process as *interactive route search* [4, 5].

An interactive route search starts as an ordinary route search and continues as an iterative process where each travel from the current location to an entity is considered as a step. In each step of the search, a route to the next geographical entity to be visited is provided. The user travels along the provided route, and upon arrival at the entity she provides feedback to the system, indicating whether the entity satisfies her. The feedback is used to indicate whether additional objects of the same type still need to be visited.

The database over which interactive route search is conducted is a *probabilistic database* where a *success probability* is assigned to each entity. The success probability specifies how likely it is that the entity satisfies the search requirements specified by the search terms. Thus, instead of having two competing goals as in the non-interactive case, i.e., finding the shortest route versus finding the route with the highest success probability, in the interactive case, the goal is merely to find the route that is as short as possible *in practice*. Hence, constructing an effective route involves taking into account the possibility that some objects will not satisfy the user, and constructing the route such that in case of a negative feedback there will be relevant alternative entities along some route toward the target. Note that before the travel starts, there can be many alternative routes and the one that is actually traveled is constructed according to the feedbacks. For instance, consider in Example 1.1 the case where Alice arrives at a toy store. If Alice is satisfied, the route continues without stopping at additional stores. Otherwise, the next steps of the route will be computed while taking into account the need to visit another toy store.

The concept of interactive route search and algorithms to answer route search queries were presented in [4, 5]. In this paper, we illustrate an actual implementation of interactive route search. The demo illustrates the simplicity of the interaction, and shows that such search can be done efficiently, effectively and in a scalable fashion, over real data.

2. ROUTE SEARCH QUERIES

In this section we briefly present route-search queries and the interactive search process. A route search query is formulated using an Android application, as depicted in Figure 1. The start location s, namely *source*, is the location of the user. The target location t, namely *destination*, is specified by providing an appropriate address. The types of entities to be visited are referred to as *stops*. Each stop is defined by inserting keywords such as "ATM", "shoe store" and "vegetarian restaurant". Order constraints specify limitations on the order by which entities should be visited. After the insertion of terms to define a stop, a spatial search is conducted to discover potential entities along possible routes to the target. Each discovered entity is assigned a probability of success and the entities are depicted as circles. Entities of each type of stops are depicted using a different color.

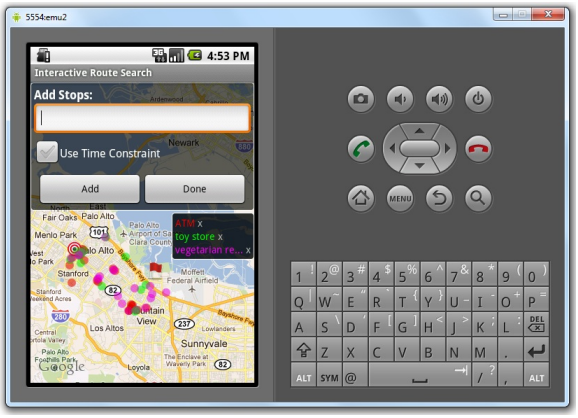


Figure 1: The query of Example 1.1 issued using the Android application.

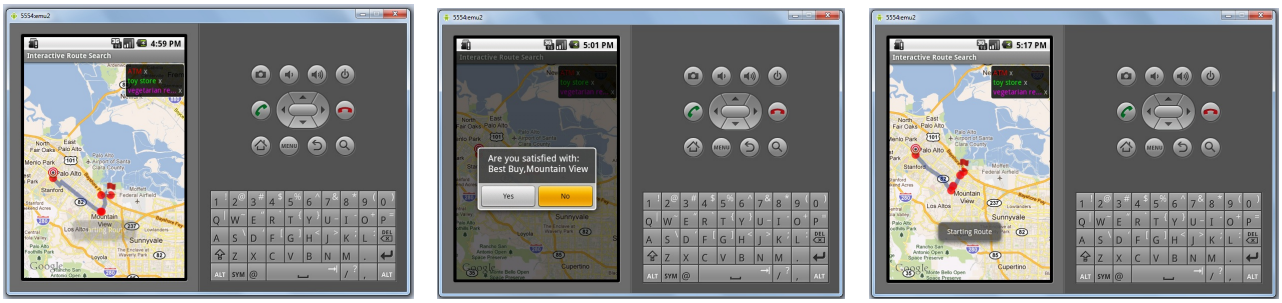
Traveling along the route is an iterative process. An initial route is provided to the user and at each stop the user provides a feedback. A positive feedback for an entity of type T means that there is no need to visit additional entities of type T, whereas a negative feedback requires visiting additional entities of type T. After each negative feedback, the route is reevaluated and depicted, as illustrated in Figure 2.

Although in each step of the travel only a single entity should be provided to the user, the system depicts a complete route because the evaluation of the route takes into account the need to visit additional entities later in the travel.

Computing the route whose expected travel distance is the shortest is an NP-hard problem. Thus, several heuristics where presented in [5]. We illustrate in this demo the Oriented-Greedy Heuristic whose computation is the most efficient among the Heuristics of [5]. In the Oriented-Greedy Heuristic, the next entity in each step is chosen while considering all the following aims. *(1)* The direction to the next entity is as close as possible to the direction of the target, in order not to go "backwards". *(2)* The entity is as close as possible to the current location, to prevent going too much forward and then going back. *(3)* The probability of success of the entity is as high as possible, to avoid visiting many entities that fail satisfying the user. *(4)* Order constraints are satisfied. A complete route to the target via the desired entities is computed in a greedy fashion while taking into account these four aims.

3. SYSTEM DESIGN

WISER is designed as a client server application. The client is an Android application that implements the user interface and uses the Google Maps API for depicting the map, potential stops and the computed route. The server tier accesses the geographical data and performs the query evaluation, initially and after each feedback. Instead of storing and accessing the geographical data on the server itself,



(a) Before feedback (b) Negative Feedback (c) After Feedback

Figure 2: An interaction with the system: (a) before the feedback, (b) providing a negative feedback and (c) change of the route after the feedback.

which introduces a significant amount of unnecessary complexity, the server is a mashup that utilizes several Web services. The server is implemented using the .NET Framework and is deployed on a Windows IIS Web server. The system architecture is presented in Figure 3.

There are several reasons to using a client-server architecture. First, this allows the server to collect feedbacks from various users in order to gather statistical data—this data is used for calculating the success probabilities of entities and can be used for various tasks such as query auto-completion and query suggestions. Second, this facilitates updates because it spears the need to access a large number of end-user devices. Thus, changes that are not related to the user interface do not require updating the client.

3.1 The Client

The client interacts with the user to receive a query, receive feedbacks and present search results. It interacts with the GPS to receive information about the location of the user and it interacts with the server for query evaluation. The communication between the client and the server is always initiated by the client and the messages are sent in XML format. The information being sent by the client to the server is shown in Table 1.

Description	Values
Route boundaries	Start and destination locations
Stop-description-1(s_1)	List of potential stops along with their probabilities
⋮	⋮
Stop-description-n (s_n)	List of potential stops along with their probabilities
Order constraint-1	(s_{i_1}, s_{j_1})
⋮	⋮
Order constraint-m	(s_{i_m}, s_{j_m})

Table 1: Data that the client sends to the server.

Upon arrival at a stop, the application requests a feedback from the user. If the feedback is negative (no satisfaction), a new route is computed. This is done by sending a new query to the server in the format presented in Table 1. In the new query, the start location is set to be the current user location, stops that were already been visited are removed from

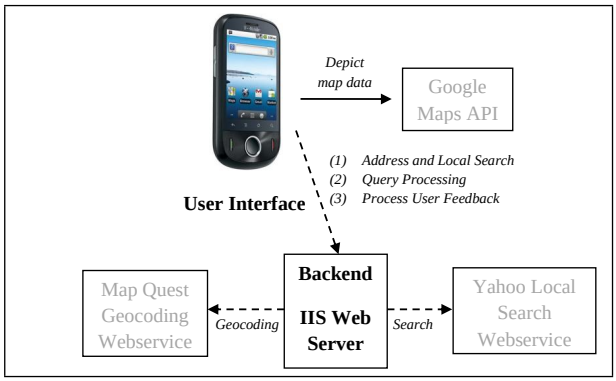


Figure 3: System architecture

the corresponding list of potential stops, stops whose type has been satisfied are removed, and any order constraint that refers to a satisfied stop type is removed. Note that, sending a full updated query to the server every time the user is unsatisfied, allows the server to remain stateless. This provides a scalable alternative to a server that maintains a state for every user query after each feedback. This approach also facilitates evaluation in a distributed manner where at different times, computation is performed by different servers. Positive responses are sent asynchronously to the server for updating the statistical data.

3.2 The Server

The server provides the necessary functionality in the form of Web services, as explained next.

3.2.1 Address and Local Search Service (ALSS)

ALSS uses a mashup of Web services to retrieve relevant points of interest (potential stops). This Web service receives a string which represents search terms or an address. It uses the MapQuest Geocoding Web service (see `http://www.mapquestapi.com/geocoding/`) to determine if the string is an address.

When search terms are provided, it is important to limit the search to an area defined by the start location s and the target location t. Otherwise, discovered entities will be too far to be relevant. The server uses the Yahoo! Local Search Web service (`http://developer.yahoo.com/search/local`

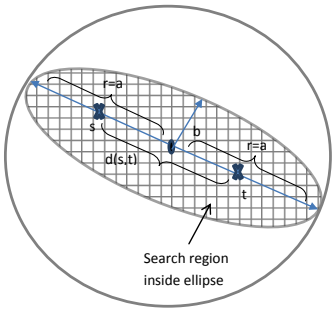


Figure 4: The search region, for s and t.

`/V3/localSearch.html`) which can find the k most-relevant entities, with respect to the given search terms, in a circular area whose center is p and whose radius is r. However, the search region of a route-search query is defined as all the objects contained in a *bounding ellipse* whose two foci are the start and target locations of the query [5]. Since the Yahoo! API does not support retrieval restricted to an elliptical area, the system conducts the retrieval restricted to a circular area, where the center point is the middle of the line that connects s and t, and the radius is the larger axis of the ellipse. This provides a bounding circle for the search region, as depicted in Figure 4. Yet, some of the retrieved objects may lay outside the elliptical search region and are, hence, discarded. Thus, to have a sufficient number of entities, redundant entities are retrieved, as follows. Let $a = m_a \cdot 0.5 \cdot d_h(s,t)$ and $b = m_b \cdot 0.5 \cdot d_h(s,t)$ denote the two axes of the ellipse. The distance $d_h(s,t)$ is the haversine distance between s to t, and m_a, m_b (set in the demo to $1.1, 0.55$, respectively) represent the magnification ratios for the two axes. Note that $m_a > 1$ allows a route to move in the opposite direction of the destination or progress further ahead from the destination.

Hence, to solve the problem described above, we assume that, on the average, only $\frac{\text{area\_ellipse}}{\text{area\_circle}}$ search results will be inside the ellipse. That is, if we desire k search results to be inside the search region, we need to request, at least, $K = \frac{\pi \cdot a \cdot b}{\pi r^2} \cdot k = \frac{m_b}{m_a} \cdot k$ (k and K are respectively set to 10 and 20 in the demo) from the Yahoo! API and filter out those that are not within the search region. The entities within the search region are referred to as the points-of-interest.

The success probabilities of points-of-interest are assigned to the entities based on their order in the search result. That is, if an entity e_1 precedes an entity e_2 in the result, then the probability assigned to e_1 is higher than that of e_2. The assigned probabilities are constructed in the range $[0.4, 0.9]$ using the distribution function $e^{-\gamma \cdot (i-1)} - (1 - p_h)$, where $\gamma = -\frac{ln(1+p_h-p_l)}{n_r-1}$, $p_h = 0.9$, $p_l = 0.4$, $n_r = 10$, and $1 \leq i \leq 10$ is the position of the entity in the search result. This represents a behavior that is similar to the well known "long tail" phenomenon in search. In a real system, the probabilities should be calculated based on entity types and user profiling, as well.

3.2.2 Query Processing

The query-processing Web service computes the routes. It receives data in the format described in Table 1. In order to run the Oriented-Greedy algorithm (see Section 2), the distances between every pair of points-of-interests needs to be determined. This distance is calculated using the haversine distance [8]. The reason we use the haversine distance instead of the network distances, is that the road network information is not available to the application, and using another Web service for this task is inefficient, because the service would be called for every pair of points of interest. The haversine distance is both easy to calculate and maintains a reasonable proportion to the actual network distance. Once the route is calculated, it is returned to the caller.

3.2.3 Process User Feedback

The user-feedback Web service receives a location, corresponding search terms and a boolean feedback specifying whether the user was satisfied with the stop. This data is asynchronously stored and serves as statistical data.

4. DEMONSTRATION

The demonstration presents WISER using an Android emulator. We show how route search queries are formulated, with and without order constraints, and how a resulting route is displayed. In addition, we show the manner by which a user provides feedback after visiting the stops. The demonstration illustrates the effectiveness and the efficiency of the approach

Our demonstration presents typical system usages. A video that illustrates such use is available via the following link: `http://www.youtube.com/watch?v=f-aXXKvVPXM`. In this video, we show a query with start location in the city of Mountain View, and a destination in the city of Santa Clara. Three stops are specified: at a gas station, at a restaurant and at a bank. Two order constraints are added to indicate that the restaurant should be visited last. The example shows the way user feedback is handled.

5. REFERENCES

[1] H. Chen, W.-S. Ku, M.-T. Sun, and R. Zimmermann. The partial sequenced route query with traveling rules in road networks. *Geoinformatica*, 15:541–569, 2011.

[2] L. Feifei, C. Dihan, H. Marios, K. George, and T. Shang-Hua. On trip planning queries in spatial databases. In *SSTD*, 2005.

[3] I. Hefez, Y. Kanza, and R. Levin. Tarsius: A system for traffic-aware route search under conditions of uncertainty. In *ACMGIS*, 2011.

[4] Y. Kanza, R. Levin, E. Safra, and Y. Sagiv. An interactive approach to route search. In *ACMGIS*, 2009.

[5] Y. Kanza, R. Levin, E. Safra, and Y. Sagiv. Interactive route search in the presence of order constraints. *Proc. VLDB Endow.*, 3:117–128, September 2010.

[6] Y. Kanza, E. Safra, and Y. Sagiv. Route search over probabilistic geospatial data. In *SSTD*, 2009.

[7] Y. Kanza, E. Safra, Y. Sagiv, and Y. Doytsher. Heuristic algorithms for route-search queries over geographical data. In *ACMGIS*, 2008.

[8] C. C. Robusto. The Cosine-Haversine formula. *The American Mathematical Monthly*, 64:38–40, Jan. 1957.

[9] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The optimal sequenced route query. *The VLDB Journal*, 17:765–787, July 2008.