Heuristic Algorithms for Route-Search Queries over Geographical Data

Yaron Kanza*
Technion
Haifa, Israel
kanza@cs.technion.ac.il

Eliyahu Safra
ESRI
Redlands, CA, USA
esafra@esri.com

Yehoshua Sagiv†
Hebrew University
Jerusalem, Israel
sagiv@cs.huji.ac.il

Yerach Doytsher
Technion
Haifa, Israel
doytsher@technion.ac.il

ABSTRACT

In a *geographical route search*, given search terms, the goal is to find an *effective* route that *(1)* starts at a given location, *(2)* ends at a given location, and *(3)* travels via geographical entities that are relevant to the given terms. A route is effective if it does not exceed a given distance limit whereas the *ranking scores* of the visited entities, with respect to the search terms, are maximal. This paper introduces route-search queries, suggests three semantics for such queries and deals with the problem of efficiently answering queries under the different semantics. Since the problem of answering route-search queries is a generalization of the traveling salesman problem, it is unlikely to have an efficient solution, i.e., there is no polynomial-time algorithm that solves the problem (unless P=NP). Hence, in this work we consider heuristics for the problem. Methods for effectively computing routes are presented. The methods are compared analytically and experimentally. For these methods, experiments on both synthetic and real-world data illustrate their efficiency and their effectiveness in computing a route that satisfies the constraints of a route-search query.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Experimentation

*Supported by G.I.F Research Grant No. 2165-1738.6/07.

†Supported by Grant 893/05 of The Israel Science Foundation and by a grant from Google.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACM GIS '08, November 5-7, 2008. Irvine, CA, USA.
Copyright 2008 ACM 978-1-60558-323-5/08/11 ...$5.00.

Keywords

Geographic information system, search, route, path, heuristic algorithms, approximation algorithms

1. INTRODUCTION

For several year now, online map services are widespread on the world-wide web, and the popularity of such services in hand-held devices is rapidly growing. Since map services are accessible to many novice users, they should allow easy formulation of queries that specify complex needs, such as a *route search*, i.e., finding a route that goes through some specified types of geographical entities.

In a traditional search over the world-wide web, users usually specify conditions in the form of a set of keywords. The result of a search is a ranked list of documents that satisfy the search conditions. The principle of a traditional search is, however, unsuitable for geographic search, because users need to actually visit the entities of the result. Traditional ranking methods ignore the location of the entities, the distance of entities from the location of the user and the distance of relevant entities from other relevant entities. Consequently, visiting geographical objects in an order that is the result of a traditional search may produce a long travel, whereas there is likely to be a much shorter route that also satisfies the user needs.

In this paper, we introduce the paradigm of route search. In a route search, the user provides keywords that specify the entities she wants to visit and a target destination. The source location is either provided by the user or discovered automatically, e.g., using a GPS device. The goal is to find a route that starts at the source location, ends at the target location and goes via entities that are relevant to the search.

Example 1. *Suppose a tourist, after landing at JFK airport in New York and renting a car, travels to her hotel. On the way she wants to visit a pharmacy, a coffee shop and a department store. A route-search application should allow the tourist to pose a suitable query (e.g., using the car-navigation system, her cellular phone or her laptop), and it should return a route starting at the parking lot of the car-rental agency, traveling through a pharmacy, a coffee shop and a department store, perhaps in a different order, and ending at the hotel. The route should be effective and should not make the tourist unnecessarily go back and forth in the*

city. Moreover, the entities on the route should match the search terms as much as possible in order to guarantee that they are indeed entities the tourist wants to visit.

There are many online map services on the web, for example Google Maps, Live Search Maps, MapQuest and Yahoo Maps. These services support applications, such as viewing a map at different scales, searching for a geographical entity using keywords or an address, and finding a route between two given addresses. These services, however, are not effective for route-search tasks as the one in Example 1.

EXAMPLE 2. *When using a standard geographic search tool for the search in Example 1, the user will need to pose three different queries—one query to search for a pharmacy, another query to search for a coffee shop and a third query to search for a department store. Then, the user would need to choose an object from the result of each query and combine these results. Generating an effective route from these search results is not straightforward. Moreover, choosing the first object from the result of each query will not necessarily yield a short route.*

When constructing a route, there are two conflicting goals. On one hand, the route should be as short as possible. On the other hand, the route should go via the most relevant entities. For instance, in many search scenarios, the following two routes will be different: the *shortest* route that goes via relevant entities and a (usually, longer) route that goes via the *most relevant* entities. The preferred route should be selected according to some chosen semantics.

We propose the following three semantics. In all of them, the geographical entities are categorized according to the search terms, and one entity from each category must be visited. For instance, in Example 1, there would be a set of pharmacy entities, a set of department-store entities and a set of coffee-shop entities. The route should include an entity from each set. Under the *shortest-route semantics*, the goal is to return the shortest route that goes via relevant entities. A second semantics is finding the *most-profitable route*, which is the route that goes via the entities having the highest accumulative relevance, subject to the constraint that the length of the route should not exceed a given limit. A third semantics is of computing the *most-reliable route*, which is a route whose length does not exceed a given limit, while trying to avoid, as much as possible, going via entities that have low relevancy. The goal in this semantics is to provide the best guarantee that all the visited entities will satisfy the user. The difference between the second and the third semantics is that in the second semantics, visiting a highly relevant entity compensates for visiting an entity whose relevancy is low. This is not the case under the third semantics where the quality of a route is determined by the entity with the lowest relevancy.

The most-profitable route is useful when the scores that represent the relevancy of items, in a search result, are accumulative values, e.g., profits. However, usually scores in the results of a search are not accumulative. For such cases, the most-reliable route is more appropriate. For instance, choosing the most-profitable route for the search task of Example 1 may yield a route that goes via highly relevant coffee shop and pharmacy, but visits a third entity that has low relevancy to the terms "department store" (e.g., when arriving at the place, the tourist may discover that the entity is not a department store). The goal of the most-reliable route is to avoid such cases. In most search systems, relevancy scores are not accumulative and therefore, in this paper, we will focus on computing the most-reliable route.

We note, however, that we have also investigated the problem of computing the most-profitable route. One approach for solving this problem is to construct the route from the result of a heuristic algorithm for some known optimization problem. In particular, this can be done by exploiting an algorithm for either the orienteering problem or the multiple-choice knapsack problem. Due to lack of space, our results about computing the most-profitable route will only be presented in the full version of this paper.

In order to provide route-search services, on the web or in devices with a limited computation power (such as cellular phones or car-navigation systems), it is crucial that the computation of queries will be efficient. Moreover, the amount of geographical data on the web is rapidly growing, hence, scalability also has a great importance. However, every one of the three semantics presented above is a generalization of the traveling-salesman problem (TSP). Since TSP is an NP-hard problem, it is unlikely that there is a polynomial-time algorithm for computing a route, under any of the three semantics. Therefore, in this paper, we present efficient heuristic algorithms for answering route-search queries.

The contribution of this paper includes introducing route-search queries, suggesting semantics for such queries, presenting efficient heuristic algorithms for computing queries and providing the results of experiments that illustrate the effectiveness and the efficiency of the proposed algorithms.

2. DATASETS AND SEARCH QUERIES

In this section, we present our framework, and we formally define the concept of *route search*.

2.1 Geographical Datasets

A *geo-spatial dataset* is a collection of geo-spatial objects. Each object represents a real-world geographical entity and has a location—the location of an object is the location of the entity it represents. An object may have additional spatial and non-spatial attributes. Height and shape are examples of spatial attributes. Address and name are examples of non-spatial attributes. We assume that locations are points and are unique, i.e., different objects have different locations. For objects that are represented by a polygonal shape and do not have a specified point location, we consider the center of mass of the polygonal shape to be the point location.

The distance between two objects is the Euclidean distance between their point locations. We denote the distance between two objects o_1 and o_2 by $distance(o_1, o_2)$. Similarly, if o is an object and l is a location, then $distance(o, l)$ is the distance from o to l.

In many scenarios, traveling from one object to another must be on a road and cannot be done in a straight line. In such cases, traversal is according to a *road network*. A road network is represented as a set of intersecting polygonal lines. The *network location* of an object o is the point on the network that is nearest to the actual location of o. Over networks, we assume that the distance between two objects is the length of the shortest path between their network locations. (For methods of computing distances over a road network, see the work of Samet *et al.* [23] and the work of Shahabi *et al.* [25].)

2.2 Search Queries

Users specify what entities they would like to visit by *search queries*. A search query consists of a set of keywords and constraints on attributes. We represent a query as a pair $Q = (W, C)$, where *(1)* W is a set of keywords, and *(2)* C is a set of constraints having the form $A \diamond v$, such that A is an attribute name, v is a value and $\diamond$ is a comparison symbol among $=, <, >, \neq, \leq$ and $\geq$. For instance, `Hotel, Wireless Internet Access, rank` ≥ 3, `price` ≤ 100 specify that the user would like to go via a hotel that provides an Internet wireless connection, has a ranking of at least three stars and a rate that does not exceed $100.

For a search, we consider the *textual component* of an object to be the concatenation of the values in the non-spatial attributes of the objects. An object o *satisfies* a search query Q when at least one keyword of Q appears in its textual component and the constraints of Q are satisfied in the usual way. Each object that satisfies a query is given a *ranking score* (or *score*, for short). The score is a value between 0 and 1, and it indicates how relevant is o to the search. We denote the score of an object o by $score(o)$.

In the literature, there are various approaches for computing relevance scores for textual elements and a set of keywords, e.g., TF-IDF, Okapi BM25 [11, 21] and others [22].[1]

2.3 Route-Search Queries

In a *route-search query*, the user specifies a *source location*, a *target location* and the entities that the route should visit. We represent a route-search query as a triplet $R = (s, t, \mathcal{Q})$, where s is a source location, t is a target location and $\mathcal{Q}$ is a set of search queries.

EXAMPLE 3. *Consider again the route-search task presented in Example 1. A suitable route-search query for this task should include* (1) *the location s of the parking lot of the car-rental agency,* (2) *the location t of the hotel, and* (3) *the following three search queries:* $Q_1 = \{$`pharmacy`$\}$*;* $Q_2 = \{$`coffee shop`$\}$*; and* $Q_3 = \{$`department store`$\}$.

A *pre-answer* to a route-search query is a route that starts at s, ends at t and for each query Q in $\mathcal{Q}$, goes via one object of the result of Q. That is, if $A_1, \ldots, A_k$ are the answers to the search queries of R, a route is a sequence $s, o_1, \ldots, o_k, t$, where $o_i \in A_i$ for $1 \leq i \leq k$. The *length* of such a route is the sum of the distances between adjacent objects, i.e., $distance(s, o_1) + \Sigma_{i=1}^{k-1} distance(o_i, o_{i+1}) + distance(o_k, t)$. The *total score* of the route is $\Sigma_{i=1}^{k} score(o_i)$. The *minimal score* of the route is $\min\{score(o_i) \mid 1 \leq i \leq k\}$. The *answer* to a route-search query is a pre-answer chosen according to a specific semantics. In this paper we present three semantics for route-search queries.

2.3.1 Semantics for Route-Search Queries

When presenting the semantics, we assume that D is a dataset, $R = (s, t, \{Q_1, \ldots, Q_k\})$ is a given route-search query, and $A_1 = Q_1(D), \ldots, A_k = Q_k(D)$ are the answers to the search queries of R, over D. Also, we assume that the sets $A_1, \ldots, A_k$ are pairwise disjoint.

Shortest Route (SR). Under the *shortest-route semantics*, the answer is the shortest pre-answer.

The next two semantics are defined with respect to a given distance limit ℓ.

Most-Profitable Route (MPR). Under the *most-profitable semantics*, the answer is the pre-answer that has the highest total score among the pre-answers whose length does not exceed ℓ.

Most-Reliable Route (MRR). Under the *most-reliable semantics*, the answer is the pre-answer with the highest minimal score among the pre-answers whose length does not exceed ℓ.

2.3.2 Comparison to Known Optimization Problems

Next, we compare the proposed semantics to optimization problems that exists in the literature.

The problem of finding the shortest route is a version of the *generalized traveling-salesman problem* (GTSP). In GTSP, given a partition of the nodes of a weighted graph to k clusters, the goal is to find the least-cost cycle passing through each cluster exactly once. Thus, GTSP is similar to computing the shortest route when the source and the target have the same location. Yet, note that our problem is limited in the following two aspects. We assume that there is an edge between every two nodes and that the weights on the edges define a metric space (i.e., the weights satisfy the triangle inequality).

GTSP has been studied extensively over the years. It was introduced by Henry-Labordere [10], Saksena [24] and Srivastava *et al.* [28] for problems that arise in computer design and in routing. Later, additional applications of GTSP were discovered [13]. Many approaches were proposed for solving GTSP, including dynamic programming [4], integer programming [15], Lagrangian relaxation [14, 18], branch-and-cut [7], genetic algorithms [27] and transforming the problem into a standard traveling salesman problem [17]. The algorithms we present in this paper are different from the above by giving precedence to efficiency over the quality of the results.

The problem of finding the most-profitable route has some similarity to the *orienteering problem*. In the orienteering problem, the input consists of a distance limit, a start location and a set of objects where each object has a score. The problem is to compute a route that (1) starts at the given starting location, (2) have a length that does not exceed the given distance limit and (3) goes via objects whose total score is maximal. The orienteering problem has been studied extensively [3, 8] and several heuristics [2, 9, 12, 16, 29] and approximation algorithms [20] were proposed for it.

There are three main differences between orienteering and the problem of computing the most-profitable route. First, in MPR, the objects are divided into sets (the sets are the answers to the queries) and an object from each set must be visited. In the orienteering problem, objects differ only in their location and score. Secondly, in MPR, exactly k objects must be visited, where k is the number of search queries in the route-search query. The number of objects in the answer to the orienteering problem is not known in advance. Thirdly, there is always an answer to the orienteering problem (a route that does not include any object is a possible answer), whereas MPR is not always satisfiable.

[1]Route search can be defined with respect to search queries with a different syntax or a different semantics. The description of search queries provided here is just for the completeness of exposition.

Because of these differences, there is no simple way of using heuristic algorithms for the orienteering problem to solve MPR.

The problem of the most-profitable route also has some similarity to the *multiple-choice knapsack problem*. In the multiple-choice knapsack problem, there are k sets of objects $N_1, \ldots, N_k$. Each object $o \in N_i$ has an associated profit and a weight. The objective is to choose exactly one item from each set N_i, such that the total profit of the chosen items is maximized while their total weight does not exceed a given capacity c. The problem has been studies from many different perspectives and several heuristics were suggested for it [1, 5, 6, 19, 26, 30].

The difference between the multiple-choice knapsack problem and MPR is that in the first, the weights of items are part of the input and do not change. Thus, when choosing an item, the total weight is increased by an amount that is independent of the other chosen items. In the most-profitable route problem, when we construct a route by adding new objects one by one, the increase in the length of the route caused by a newly added object depends on (its distances from) the preceding and the succeeding objects. So, for instance, the same objects in different orders may produce routes with different lengths.

The three semantics that we consider in this paper are a generalization of the traveling-salesman problem (TSP). In TSP, the goal is to find the shortest path that starts at a given location, ends at a given location and goes via all the objects. It is easy to show that computing route-search queries under either one of the three semantics is (at least) as hard as TSP.

3. ALGORITHMS

Since TSP is an NP-hard problem, computing a route, under any of the proposed semantics, is NP-hard. Hence, assuming $P \neq NP$, there is no polynomial-time algorithm for answering route-search queries and, thus, in this section we present heuristic algorithms for query answering.

When devising heuristic algorithms, in many cases there is a tradeoff between the efficiency of the computation and the quality of the results. Intuitively, a heuristic algorithm that examines many possible solutions will, in most cases, provide more accurate results than a heuristic algorithm that examines only a few possible solutions; however, the first algorithm will probably be less efficient than the second.

Algorithms for online services should be highly efficient. Many users will not be willing to wait for an answer more than several seconds. Thus, our goal is to provide algorithms that have time complexity that is linear or close to linear, in the size of the input.

Throughout this section, we use the following notations. We denote by $R = (s, t, \mathcal{Q})$ the route-search query. We denote by D the dataset on which R is computed. By ℓ we denote the distance limit, when relevant.

3.1 Algorithms for Shortest Route

In this section, we present three variants of a greedy algorithm for the shortest-route problem. These algorithms are simple and our focus is on their efficient implementation.

Before presenting the algorithms, we introduce some notations. Consider a sequence of objects σ. With a slight abuse of notation, we also consider σ as the set of the objects it contains. By $indexes(\sigma) = \{i \mid \sigma \cap A_i \neq \emptyset\}$ we denote the indexes of the sets that have a representative in σ. By $\sigma[j]$ we denote the object in the j-th position of σ, e.g., for $\sigma = s, o_1, t$, it holds that $\sigma[2] = o_1$. We denote by $insert(\sigma, o, j)$ the sequence that is created by inserting the object o into σ, after the object in position j and before the object in position $j + 1$.

<u>*Greedy Extension* $((s, t, Q_1, \ldots, Q_k), D)$</u>

Input: Source location s, target location t, search queries $Q_1, \ldots, Q_k$, a dataset D
Output: A route starting at s, ending at t and visiting at least one object from each answer to Q_i over D

```
 1: for i = 1 to k do
 2:    A_i ← Q_i(D)
 3: let σ = s,t be an initial sequence
 4: I ← {1,...,k}
 5: j ← 1
 6: while I ≠ ∅ do
 7:    let U = ∪_{i∈I} A_i
 8:    find o in U such that length(insert(σ,o,j)) =
       min{length(insert(σ,o',j)) | o' ∈ U}
 9:    σ ← insert(σ,o,j)
10:    remove from I the index i' of the set A_i' that con-
       tains o
11:    j ← j + 1
12: return σ
```

Figure 1: The Greedy Extension heuristic algorithm for answering queries under the shortest-route semantics.

3.1.1 Greedy Extension

The *Greedy Extension Algorithm* (GExt), presented in Figure 1, is a greedy algorithm for the shortest-route problem. Given a route-search query R and a dataset D, GExt evaluates the search queries of R over D and then constructs a route by greedily inserting objects at the end of the sequence. Each insertion is of the object that has the smallest effect on the length of the route.

In GExt we construct a route iteratively, starting with the sequence s, t. In each iteration, we insert an object into the last segment of the sequence. That is, given the initial sequence s, t, we add an object between s and t. In later iterations, the sequence is of the form $s, o_1, \ldots, o_m, t$ $(m \geq 1)$ and we insert an object between o_m and t. In each iteration, we add to the sequence σ an object from a set A_i, such that $i \notin indexes(\sigma)$. The added object is the one that yields the smallest increase in $length(\sigma)$.

In GExt, there are k iterations. In each iteration, we examine at most n possible extensions to the constructed sequence, where n in the size of the dataset D. Thus, GExt has linear time complexity.

PROPOSITION 1. *The time complexity of Greedy Extension is $O(k|D|)$, where k is the number of search queries and $|D|$ is the size of the dataset over which R is computed.*

We can decrease the number of objects being examined in each iteration by using a grid index (mesh). In a grid index, the given area is partitioned into squares and for each square, the index contains an entry that stores references to the objects located in that square.

We use the index by applying a two-step retrieval process. In the first step, we find an object near the interval where an object should be inserted. Then, we verify that the inserted object is the one that yields the smallest increase in the length of the route. A detailed description is as follows.

Consider the sequence $s, o_1, \ldots, o_m, t$. We need to insert an object between o_m and t. First, we build a buffer around the line that connects o_m and t. The size of the buffer is chosen so that the expected number of objects it contains, from each set A_i, will be constant. Thus, suppose that S is the whole area of the map and the objects of D (and of each set A_i) are distributed uniformly in S. Let $d = \frac{\min\{|A_i||1 \le i \le k\}}{S}$ be the *density* of objects in S, and suppose that the minimum is obtained for A_h (i.e., when $i = h$). Let l be the distance between o_m and t. In an area of size $\frac{1}{d}$, the expected number of objects from A_h will be 1, and the expected number of objects from every other set A_j among $A_1, \ldots, A_n$ will be the constant $\frac{|A_j|}{|A_h|}$ that is greater than 1. The buffer we construct is the area S_x containing all the points whose distance from the line that connects o_m and t is not greater than x, where x is determined as follows. The size of the area S_x is $l \cdot 2x + 2(\frac{1}{2}\pi x^2)$, as illustrated in Figure 3(a). In order to construct S_x so that its area will be equal to $\frac{1}{d}$, we choose x to be

$$x = \frac{\sqrt{l^2 + \frac{\pi}{d}} - l}{\pi}.$$

Using the grid index, we retrieve the objects in the area S_x and find the object o' whose addition causes the smallest increase in the length of the sequence. (If we cannot find a suitable object in S_x, we increase the size of the search area by adding $\frac{1}{d}$, i.e., replacing $\frac{1}{d}$ with $\frac{2}{d}$ when computing x. We continue increasing the search area, till we find a suitable object.)

In the second step of the retrieval, we check whether the object o' is indeed the one that causes the smallest increase in the length of the sequence. We do it by examining the elliptic area S_E of all the points p, such that $\mathit{distance}(o_m, p) + \mathit{distance}(p, t) \le \mathit{distance}(o_m, o') + \mathit{distance}(o', t)$. We retrieve the objects in S_E using the index, and examine whether one of them should be added to the sequence instead of o'. It is easy to see that for objects outside of S_E, adding them to the sequence will yield an increase in the length that is greater than the increase caused by inserting o'. According to the construction of the buffer S_x, for the chosen object o' it holds that $\mathit{distance}(o_m, o') + \mathit{distance}(o', t) \le 2x + l$. So, the length of the major axis of the ellipse S_E, denoted by a, is less than or equal to $2x + l$, and the length of the minor axis, denoted by b, is at most $2\sqrt{x(x+l)}$, because $(\frac{b}{2})^2 \le (\frac{2x+l}{2})^2 - (\frac{l}{2})^2$.

Now, let $\varepsilon < 1$ be a small constant such that if $x < \varepsilon l$, then $\pi x^2 \ll S_x$, i.e., πx^2 is only a small percentage of the area S_x. The density d is considered *small* with respect to l when $x \ge \varepsilon l$. In this case, the area of S_E, which is $\frac{\pi}{4}ab$, is proportional to S_x, and hence, the expected number of objects it contains is bounded by a constant. When the density is not small, in almost all cases the object o' will be near the line that connects o_m and t, since $x < \varepsilon l$. So, S_E will lie inside S_x, and again, the expected number of objects in S_E is bounded by a constant. Consequently, in each iteration of GExt, we examine a constant number of objects. Thus, the time complexity of each step is a function of the number of index entries we examine, which is usually much smaller than $|D|$.

Greedy Insertion $((s, t, Q_1, \ldots, Q_k), D)$

Input: Source location s, target location t, search queries $Q_1, \ldots, Q_k$, a dataset D
Output: A route starting at s, ending at t and visiting at least one object from each answer to Q_i over D

1: **for** $i = 1$ to k **do**
2: $\quad A_i \leftarrow Q_i(D)$
3: let $\sigma = s, t$ be an initial sequence
4: $I \leftarrow \{1, \ldots, k\}$
5: $i \leftarrow 1$
6: **while** $i \le k$ **do**
7: $\quad$ let $U = \cup_{i' \in I} A_{i'}$
8: $\quad$ let o, j be the object and the position such that $\mathit{length}(\mathit{insert}(\sigma, o, j)) \le \mathit{length}(\mathit{insert}(\sigma, o', j'))$ for all o' in U and $1 \le j' \le i$
9: $\quad \sigma \leftarrow \mathit{insert}(\sigma, o, j)$
10: $\quad$ remove from I the index i' of the set $A_{i'}$ that contains o
11: $\quad i \leftarrow i + 1$
12: **return** σ

Figure 2: The Greedy Insertion heuristic algorithm for answering queries under the shortest-route semantics.

3.1.2 The Greedy-Insertion Algorithm

In GExt, each extension is by adding an object to the last segment of the sequence. This approach helps keeping the algorithm efficient; however, in many cases, after constructing part of the route, we may discover that for some set A_i, the best position to insert any object of A_i into the sequence is not in the last segment. Thus, in the *Greedy-Insertion Algorithm* (GIns), we allow insertion of objects into any segment of the sequence. That is, given a sequence $s, o_1, \ldots, o_m, t$, GIns inserts the object that yields the smallest increase in the length of the route, where the insertion can be between any two elements of the sequence, i.e., between s and o_1, between o_i and o_{i+1} for some $1 \le i \le m-1$, or between o_m and t. GIns is presented in Figure 2.

In each iteration of GIns, we examine at most $|D|$ objects and for each object, at most k segments are considered as candidates for a place where the object can be inserted. This provide the following complexity.

PROPOSITION 2. *The time complexity of Greedy Insertion is $O(k^2|D|)$, where k is the number of search queries and $|D|$ is the size of the dataset over which R is computed.*

For boosting the efficiency of the computation, we use a grid index in the same way we used it in GExt. In each iteration, we first construct a buffer having a width x around the route constructed in the previous iteration. This is illustrated in Figure 3(c). (The width x is calculated as for GExt, where l is the sum of lengths of all the segments of the sequence.) We retrieve the objects of the grid cells that intersect the buffer, and we find the best candidate o' and position i for the insertion of o' in position i. Then, for every pair of objects o_j and o_{j+1} that are adjacent in the sequence

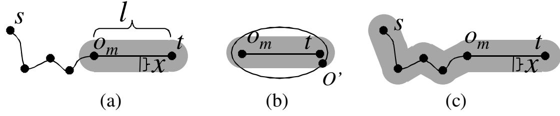


Figure 3: (a) A buffer S_x, depicted in gray, around the line that connects o_m and t. (b) The elliptic area S_E. (c) A buffer around a partially created route.

prior to the insertion of o', we consider the elliptic area containing all the points whose distance from the two objects o_j and o_{j+1} does not exceed $distance(o_i, o') + distance(o', o_{i+1})$. As earlier, we check whether any object in this elliptic area should be inserted between o_j and o_{j+1} instead of inserting o' between o_i and o_{i+1}.

3.1.3 The Infrequent-First Heuristic

The sets $A_1, \ldots, A_k$ can be of different size. So, if a set A_i is much larger than a set A_j and the objects of the sets are distributed approximately uniformly, then at a random point there is a greater chance of finding a near object of A_i than a near object of A_j. Similarly, there is a higher chance of finding an object of A_i in adjacency to a partially created route than of finding there an object of A_j. Consequently, in an ordinary run, GExt and GIns are expected to add objects of large sets before adding objects of smaller sets. The intuition behind the *Infrequent-First Heuristic* (IFH) is to reverse that order of insertion and start by inserting objects from small sets, since such objects tend to be infrequent and may not exist in adjacency to a partially created route.

In IFH, we order the sets $A_1, \ldots, A_k$ by descending size. Let $A_{j_1}, \ldots, A_{j_k}$ be such an order, i.e., for every $1 \le i < i' \le k$, we have $|A_{j_i}| \le |A_{j_{i'}}|$. Then, we apply the algorithm GIns with the following change. In each iteration i, the inserted object is chosen from the set A_{j_i} rather than from the union of several sets.

IFH is expected to be slightly more efficient than GIns, since in each iteration we examine a smaller set of objects. The ordering of the sets uses some sorting algorithm and has a complexity of $O(k \log k)$, so when k is much smaller than $|D|$, it has an insignificant influence on the running time.

3.2 Algorithms for Most Reliable Route

In this section, we present two algorithms for the most-reliable route problem. One algorithm works in a bottom-up fashion by starting with the shortest route and improving it iteratively. The second algorithm works in a top-down fashion. It starts with a small set of objects and extends this set as long as it is possible to build a route whose length is smaller than the given distance limit. Next, we describe these algorithms more precisely. In order to simplify the description of the algorithms, we assume that scores are unique, i.e., for every two objects $o_1 \in A_i$ and $o_2 \in A_j$, if $o_1 \neq o_2$ then $score(o_1) \neq score(o_2)$. Note, however, that changing the algorithms to handle the case where different objects may receive the same score is straightforward.

3.2.1 Ascending from Shortest

The *Ascending-from-Shortest Algorithm* (ASA), presented in Figure 4, is a heuristic algorithm for MRR. It starts by computing the shortest route, e.g., using one of the algorithms previously presented. It then applies an iterative

Ascending from Shortest $((s, t, Q_1, \ldots, Q_k), \ell, D)$

Input: Source location s, target location t, search queries $Q_1, \ldots, Q_k$, a distance limit ℓ, a dataset D
Output: A route with length not exceeding ℓ and high minimal score, starting at s, ending at t and visiting at least one object from each answer to Q_i over D

```
 1: for i = 1 to k do
 2:    A_i ← Q_i(D)
 3: compute the shortest route and let σ be the result
    sequence (e.g., using IFH)
 4: if length(σ) > ℓ then
 5:    return ∅
 6: stop ← false
 7: repeat
 8:    find the object o with the lowest score in σ
 9:    σ' ← σ − o   (* σ' results by removing o from σ *)
10:    find the set A_i such that o ∈ A_i
11:    M ← {(o', j') | o' ∈ A_i and score(o') > score(o)
       and 1 ≤ j' ≤ k and length(insert(σ', o', j')) ≤ ℓ}
12:    if M = ∅ then
13:       stop ← true
14:    else
15:       find (o'', j'') where length(insert(σ', o'', j'')) =
          min{length(insert(σ', o', j')) | (o', j') ∈ M}
16:       σ ← insert(σ', o'', j'')
17: until stop
18: return σ
```

Figure 4: The Ascending from Shortest heuristic algorithm for answering queries under the most-reliable semantics.

sequence of improvement steps. In each step, it finds the object o of the route with the smallest score. Then, it examines all the possible candidates to replace o. We denote by $replace(\rho, o, o')$ the route that is created by removing an object o from a sequence ρ and inserting o' into the position where the increase in the length is the smallest.

Suppose that $o \in A_i$ and the current route is ρ. Then, a candidate to replace o is an object o' from A_i such that (1) $score(o) < score(o')$, and (2) $length(replace(\rho, o, o')) \le \ell$ (recall that ℓ is the given limit distance). The algorithm replaces o by the candidate object that causes the smallest increase in the length of the route.

The algorithm stops when for the object with the lowest score, there are no candidates to replace it. Note that each replacement increases the minimal score of the route.

For analyzing the complexity of the algorithm, we note that it comprises two steps. The first step is of generating a shortest route. The time complexity of this step depends on the algorithm that is used. As previously shown, we can use an $O(k^2|D|)$-time heuristic algorithm for this computation.

The second step is an iterative process of improving the minimal score. It has at most $|D|$ iterations. This is because in each iteration an object is replaced, and the algorithm never adds to the route an object that has been previously removed. In each iteration, there are at most $|D|$ objects to examine, and for each object we consider k possible segments in which this object can be inserted. Consequently, the complexity of ASA is as follows.

RMHR $((s, t, Q_1, \ldots, Q_k), \ell, D)$

Input: Source location s, target location t, search queries $Q_1, \ldots, Q_k$, a distance limit ℓ, a dataset D
Output: A route with length not exceeding ℓ and high minimal score, starting at s, ending at t and visiting at least one object from each answer to Q_i over D

```
1: for i = 1 to k do
2:    A_i ← Q_i(D)
3: U ← ∪_{i=1}^k A_i
4: sort U according to object scores in a descending order
5: T ← ∅
6: for i = 1 to k do
7:    find the object o_i with the highest score in A_i
8:    T ← T ∪ {o_i}
9:    remove o_i from U
10: while true do
11:    compute the shortest route σ over s, t and the objects of T (e.g., using IFH)
12:    if length(σ) ≤ ℓ then
13:       return σ
14:    else if U = ∅ then
15:       return ∅
16:    else
17:       find in U the object o with the highest score, and move o from U to T
```

Figure 5: The algorithm Route over the Most-Highly Ranked Objects, for answering queries under the most-reliable semantics.

PROPOSITION 3. *The Ascending-from-Shortest heuristic algorithm has $O(k|D|(k + |D|))$ time complexity.*

In order to increase the efficiency of the algorithm, we reduce the number of objects being examined in each step by removing objects that cannot affect the result. When the algorithm starts, we remove from the sets $A_1, \ldots, A_k$ all the objects o such that $distance(s, o) + distance(o, t) > \ell$. A second reduction is done during the run of the algorithm. In each iteration, we compute $l' = \ell - length(\sigma')$. We then, consider only objects o, such that there are two adjacent objects in σ', say o_1 and o_2, for which $distance(o_1, o) + distance(o, o_2) \leq l'$. We do not extend σ' by objects that do not satisfy this condition.

3.2.2 A Route over the Most-Highly Ranked Objects

The algorithm *Route over the Most-Highly Ranked Objects* (RMHR) tackles MRR in a top-down fashion. The algorithm is presented in Figure 5. In the algorithm, we examine *sets of highly-ranked objects*. We define $T \subseteq D$ as a set of highly-ranked objects if for every object $o_t \in T$ and every object $o_d \in D \setminus T$, it holds that $score(o_t) \geq score(o_d)$. We search for a set T of highly-ranked objects that is minimal in the following sense: There is a route σ over objects of T that satisfies $length(\sigma) \leq \ell$ and the other conditions (i.e., starting at s, ending in t and going via one object of each set among $A_1, \ldots, A_k$), but such a route does not exist over any subset $T' \subset T$ of highly-ranked objects.

We search for the minimal set of highly-ranked objects by sorting the objects of each set A_i in a descending order according to their score. Initially, we add to T the object with the highest score for each A_i. If we find a route over T whose length is smaller than or equal to ℓ, we return this route. Otherwise, we add to T the object that has the highest score among the objects that are not in T. We stop when we find a route whose length is smaller than ℓ or when there are no more objects that we can add to T.

RMHR employs a heuristic algorithm for computing the shortest route. However, when using an exact algorithm for computing the shortest route, RMHR computes an optimal solution to the most-reliable route problem.

PROPOSITION 4. *Let R be a route-search query under the most-reliable semantics. When RMHR uses an exact algorithm for computing the shortest route, it correctly computes an optimal answer to R if there is one.*

For improving the efficiency of RMHR, the following three optimizations can be used.

1. Initially, we discard all the objects o of U such that $distance(s, o) + distance(o, t) > \ell$, since such objects do not affect the result. Discarding these objects can be done in a single pass over U, and it decreases the number of objects being sorted in Step 4 of the algorithm. Since the sort has $O(|U| \log |U|)$ time complexity and discarding the objects has linear time complexity, we expect this step to decrease the running time in most practical cases.

2. In the first insertion of objects to T (after Line 9 in Figure 5), if o is the object with the smallest score in T, then we can move from U to T any object that has a score greater than or equal to $score(o)$.

3. Finding the set T can be carried out in the form of a binary search. After having all the relevant objects sorted in the queue U, we partition U into two sets. Let T be the set that contains the $\frac{1}{2}|U|$ objects in U with the highest score. If we can compute over T a route whose length does not exceed ℓ, we let T be the set that contains the $\frac{1}{4}|U|$ objects in U with the highest score. Otherwise, we take T to be the $\frac{3}{4}|U|$ objects in U with the highest score. We continue this way (adding or removing $\frac{1}{2^i}|U|$ objects in each step i), till we find the minimal set of highly-ranked objects.

In our tests we used an implementation of RMHR that does not employ the above optimizations. Experiments that show the effect of the optimizations on the running time of RMHR will be included in the full version of the paper.

If we use the binary-search approach, we get $\log_2 |D|$ iterations in RMHR and, thus, the complexity is as follows.

PROPOSITION 5. *If RMHR employs an $O(k^2|D|)$-time algorithm for finding the shortest route, then RMHR runs in $O(k^2|D| \log |D|)$ time.*

4. EXPERIMENTS

We tested our algorithms on both synthetically-generated datasets and real-world datasets. Our goal was to compare the efficiency and the effectiveness of our algorithms, for different queries and over various datasets. The experiments were conducted on a PC equipped with a Core 2 Duo processor 2.13 GHz (E6400), 2 GB of main memory and Windows XP Professional operating system.

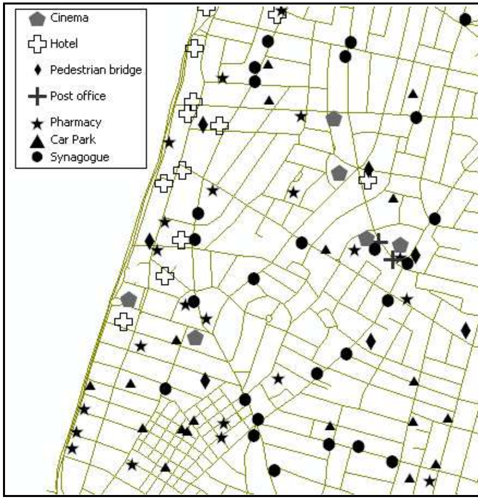


Figure 6: Map of Tel-Aviv (fragment).

4.1 Real-World Test

The real-world data that we used in our experiments is part of a digital map, of the city Tel-Aviv, that has been generated by the Mapa company. A fragment of that map is presented in Figure 6. In our tests, we used the "Point Of Interest" (POI) layer of the map. The objects in this layer represent many different types of geographical entities. We extracted from the map 628 objects of seven different types (20 cinemas, 29 hotels, 31 pedestrian bridges, 54 post offices, 136 pharmacies, 169 parking lots and 189 synagogues). That is, in the tests we had $k = 7$. These objects received scores that are normally distributed, with mean of 69.7 percent and standard deviation of 9.98 percent.

In our experiments we examined three cases of locations of the source s and the destination t. Case A: The source and the destination are the same location. Case B: There is a medium distance between s and t. Case C: There is a large distance between s and t, i.e., each location is in a different corner of the map.

	GExt	**GIns**	**IFH**
Length (meters)	7068	4666	4503
Time (milliseconds)	1	5	3

Table 1: The length of the result route and the running time of the algorithms for SR, when evaluated over the Tel-Aviv dataset.

Table 1 shows the results of the three SR algorithms over the Tel-Aviv dataset. The results are presented for Case B (the distance between s and t is neither small nor large). For Case A and Case C the algorithms provide similar results. The test results support our analysis. They show that the route provided by IFH is shorter than the routes of the other two methods. The route of GExt is the longest among the three routes. As for the running times, GExt is the most efficient and GIns is the least efficient.

In Figures 7, 8 and 9, we present the results of experimenting with RMHR and ASA over the Tel-Aviv dataset, when computing the most-reliable route. Figure 7 presents the minimal scores of routes computed by RMHR and ASA, for different values of the distance limit ℓ. The results show that in all circumstances, RMHR computes better routes than ASA. When the distance bound ℓ is increased, the results of ASA are improved, and eventually become as good as the results of RMHR.

Figure 8 presents the running times of RMHR and ASA as a function of the distance limit ℓ. In most cases, RMHR is more efficient than ASA. However, when ℓ is small, ASA is faster. This is because for a small ℓ, the number of iterations that ASA performs is small, whereas RMHR needs many iteration to complete its task. When ℓ is not small, ASA performs many iterations, so RMHR finds a route faster. Figure 9 shows that when ℓ is small, the route computed by RMHR is shorter than the route computed by ASA. When ℓ is large, ASA computes the shorter route.

4.2 Tests on synthetic Data

Testing our algorithms over synthetic data allows us to examine the algorithms over datasets with specific, sometimes extreme, properties. In a synthetic dataset we have control over the distribution of the locations of objects in the area of the map, the way that the objects are partitioned into sets, etc. For generating the synthetic datasets, we implemented a random-dataset generator. Our generator is a two-step process. First, the objects are generated. The locations of the objects are randomly chosen according to a given distribution, in a square area. In the second step, we partition the objects into sets and a score value is attached to each object. The partitioning of objects into sets can be uniform or according to a distribution specified by the user.

The user provides the following parameters to the dataset generator. The number of objects, the size of the square area in which the objects are located and the minimal distance between objects; for simulating search results, the user provides the distribution of scores, and the distribution of the size of the sets in the partition. These parameters allow a user to generate tests with different sizes of datasets and different partitions of the datasets into sets.

In Table 2 we present the results of experiments with the SR algorithms over synthetic datasets, one containing 10,000 objects and the other containing 100,000 objects. These tests illustrate the efficiency of our algorithms and they provide another evidence that IFH generates better results than the other two algorithms.

	10,000			**100,000**		
	GExt	GIns	IFH	GExt	GIns	IFH
Length	537	466	437	688	521	511
Time	23	109	62	234	773	652

Table 2: The length of the result route (meters) and the running time (milliseconds) of the algorithms for SR, when evaluated over a synthetic dataset of size 10,000 and a synthetic dataset of size 100,000.

In order to compare the efficiency of RMHR and ASA, we present in Table 3 their running times over datasets with various sizes. The table presents the mean of many runs using different distance limits. The standard deviation is also presented. The results show that RMHR is much more efficient than ASA. Moreover, having a small standard deviation shows that RMHR is efficient in almost all the cases.

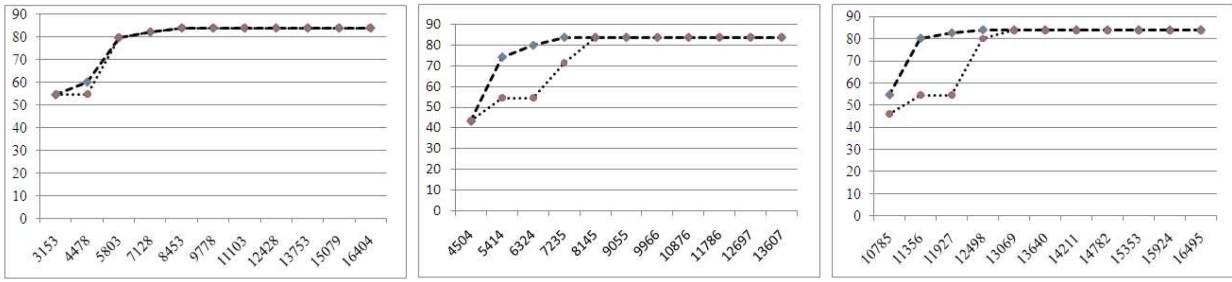


Figure 7: The highest minimal score (percent) of the routes constructed by RMHR (dark dashed line) and ASA (light dotted line) as a function of the distance limit ℓ, over the data of Figure 6. On the left Case A ($s = t$), in the middle Case B, on the right Case C (s is far from t).

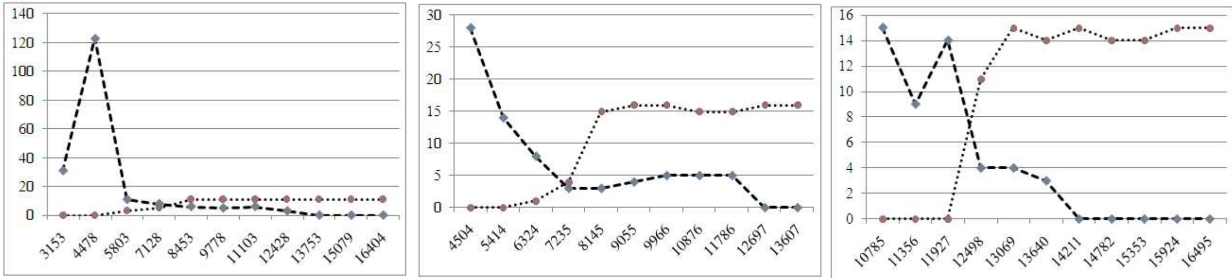


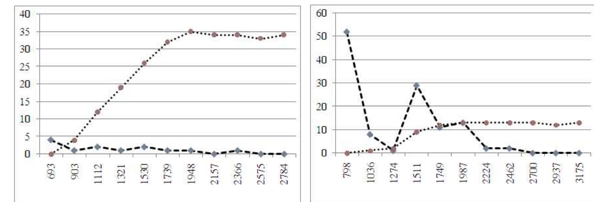
Figure 8: The running time (in milliseconds) of RMHR (dark dashed line) and ASA (light dotted line) as a function of the distance limit. On the left Case A, in the middle Case B, on the right Case C.

	RMHR		ASA	
	mean	σ	mean	σ
1,000 objects	10	16	9	5
10,000 objects	16	7	267	154
100,000 objects	149	2	2714	1243

Table 3: Running times (milliseconds) of RMHR and ASA, over datasets with various sizes. Case B holds for s and t.

ASA, for comparison, has a large standard deviation because it is efficient in some cases and not efficient in others.

Figure 10 illustrates the effect of the partitioning into sets on the running times of RMHR and ASA (the size of the dataset is 1,000 objects). When the partitioning is into sets of approximately equal size, i.e., every two sets among $A_1, \ldots, A_n$ have a similar size, then RMHR is very efficient and ASA is not efficient (Figure 10(a)). When the partitioning is uneven, then the efficiency of RMHR decreases while the efficiency of ASA increases (Figure 10(b)). To see why this happens consider the case where some set A_i is small. When ASA tries to improve a route by replacing an element of A_i, it quickly fails and stop running. So having a small set increases the efficiency of ASA. RMHR iteratively adds objects according to their score, to the set T over which it works. So, when A_i is small, RMHR will add objects of A_i to T at a low rate (for each object of A_i added to T, many objects of other sets will be added to T). Consequently, the computation will be slower.



(a) Even partitioning (b) Uneven part.

Figure 10: Running times of RMHR and ASA as a function of ℓ, when the partitioning of the dataset is to even sets (left) or uneven sets (right).

5. CONCLUSION

We introduced the paradigm of a route-search query and proposed three semantics for route-search queries. For the shortest-route semantics, we presented three efficient algorithms, namely GExt, GIns and IFH. GExt is the most efficient algorithm among the three, whereas IFH provides better results than the other two algorithms. For the most-reliable semantics, we developed the algorithms ASA and RMHR. Our experiments show that in almost all circumstances, RMHR is more efficient than ASA and also provides better results. We explained how to efficiently implement our algorithms, analyzed their complexity, and tested them over both synthetic and real-world data.

Efficiency is crucial for having route search as a web service. Our experiments provide evidence for the feasibility

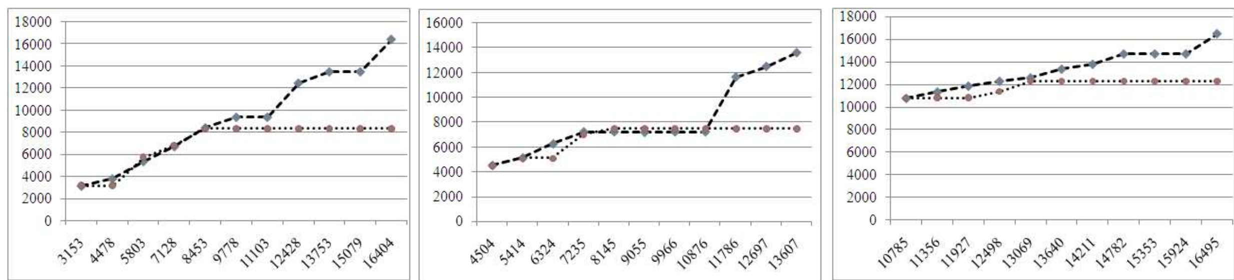


Figure 9: The length of the route constructed by RMHR (dark dashed line) and ASA (light dotted line) as a function of the distance limit. On the left Case A, in the middle Case B, on the right Case C.

of developing a practical system based on our algorithms. As future work, we plan to develop a rich syntax for route-search queries, build a system that is capable of answering queries online, and examine different approaches to presenting the result of a search in an interactive fashion.

6. REFERENCES

[1] R. Armstrong, D. Kung, P. Sinha, and A. Zoltners. A computational study of a multiple-choice knapsack algorithm. *ACM Transactions on Mathematical Software*, 9:184–198, 1983.

[2] I. Chao, B. Golden, and E. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88(3):475–489, 1996.

[3] I. Chao, B. Golden, and E. Wasil. The team orienteering problem. *European Journal of Operational Research*, 88:464–474, 1996.

[4] A. G. Chentsov and L. N. Korotayeva. The dynamic programming method in the generalized traveling salesman problem. *Mathematical and Computer Modeling*, 25(1):93–105, 1997.

[5] M. Dyer. An o(n) algorithm for the multiple-choice knapsack linear program. *Mathematical Programming*, 29:57–63, 1984.

[6] M. Dyer, N. Kayal, and J. Walker. A branch and bound algorithm for solving the multiple choice knapsack problem. *IJCAM*, 11:231–249, 1984.

[7] M. Fischetti, J. J. Salazar-González, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.

[8] B. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[9] B. Golden, Q. Wang, and L. Liu. A multifaceted heuristic for the orienteering problem. *Naval Research Logistics*, 35:359–366, 1988.

[10] A. Henry-Labordere. The record balancing problem - a dynamic programming solution of a generalized traveling salesman problem. *Revue Francaise D Informatique DeRecherche Operationnelle*, 2:43–49, 1969.

[11] S. Jones, S. Walker, and S. Robertson. A probabilistic model of information retrieval: Development and comparative experiments (parts 1 and 2). *Information Processing and Management*, 36(6):779–840, 2000.

[12] P. Keller. Algorithms to solve the orienteering problem: A comparison. *European Journal of Operational Research*, 41:224–231, 1989.

[13] G. Laporte, A. Asef-Vaziri, and C. Sriskandarajah. Some applications of the generalized traveling salesman problem. *Journal of the Operational Research Society*, 47(12):1461–1467, 1996.

[14] G. Laporte, H. Mercure, and Y. Nobert. Finding the shortest hamiltonian circuit through n clusters: A lagrangian approach. *Congressus Numerantium*, 48:277–290, 1985.

[15] G. Laporte and Y. Nobert. Generalized traveling salesman problem through n-sets of nodes - an integer programming approach. *INFOR*, 21(1):61–75, 1983.

[16] A. Leifer and M. Rosenwein. Strong linear programming relaxations for the orienteering problem. *European Journal of Operational Research*, 73:517–523, 1994.

[17] Y. Lien, E. Ma, and B. W. S. Wah. Transformation of the generalized traveling-salesman problem into the standard traveling-salesman problem. *Information Sciences*, 74(1-2):177–189, 1993.

[18] C. E. Noon and J. C. Bean. A lagrangian based approach for the asymmetric generalized traveling salesman problem. *Operations Research*, 39(4):623–632, 1991.

[19] D. Pisinger. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, 83(2):394–410, 1995.

[20] R. Ramesh, Y. Yoon, and M. Karwan. An optimal algorithm for the orienteering tour problem. *ORSA Journal on Computing*, 4(2):155–165, 1992.

[21] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In *Proc. of the Text REtrieval Conference (TREC-3)*, pages 109–126, Gaithersburg, USA, 1994.

[22] G. Salton and M. McGill. *Introduction to modern information retrieval.* McGraw-Hill, 1983.

[23] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *ACM SIGMOD*, pages 43–54, 2008.

[24] J. P. Saskena. Mathematical model for scheduling clients through welfare agencies. *J. of the Canadian Operational Research Society*, 8:185–200, 1970.

[25] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh. A road network embedding technique for k-nearest neighbor search in moving object databases. *GeoInformatica*, 7(3):255–273, 2003.

[26] P. Sinha and A. Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27(3):503–515, 1979.

[27] L. V. Snyder and M. S. Daskin. A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational research*, 174:38–53, 2006.

[28] S. S. Srivastava, S. Kumar, R. C. Garg, and P. Sen. Generalized traveling salesman problem through n sets of nodes. *Journal of the Canadian Operational Research Society*, 7:97–101, 1969.

[29] T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35(9):797–809, 1984.

[30] E. Zemel. An o(n) algorithm for the linear multiple choice knapsack problem and related problems. *Information Processing Letters*, 18:123–128, 1984.