Computing a k-Route over Uncertain Geographical Data

Eliyahu Safra[1], Yaron Kanza[2], Nir Dolev[1], Yehoshua Sagiv[3,⋆], and Yerach Doytsher[1]

[1] Department of Transportation and Geo-Information, Technion, Haifa, Israel
{safra,dolev,doytsher}@technion.ac.il
[2] Department of Computer Science, University of Toronto, Toronto, Canada
yaron@cs.toronto.edu
[3] School of Engineering and Computer Science, The Hebrew University, Jerusalem, Israel
sagiv@cs.huji.ac.il

Abstract. An uncertain geo-spatial dataset is a collection of geo-spatial objects that do not represent accurately real-world entities. Each object has a *confidence value* indicating how likely it is for the object to be correct. Uncertain data can be the result of operations such as imprecise integration, incorrect update or inexact querying. A *k-route*, over an uncertain geo-spatial dataset, is a path that travels through the geo-spatial objects, starting at a given location and stopping after visiting k correct objects. A k-route is considered shortest if the expected length of the route is less than or equal to the expected length of any other k-route that starts at the given location. This paper introduces the problem of finding a shortest k-route over an uncertain dataset. Since the problem 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. Three methods for computing a short k-route are presented. The three methods are compared analytically and experimentally. For these three methods, experiments on both synthetic and real-world data show the tradeoff between the quality of the result (*i.e.*, the expected length of the returned route) and the efficiency of the computation.

1 Introduction

Spatial datasets store objects that represent real-world geographical entities. When such datasets are *uncertain*, users who see only the information stored in the dataset cannot be sure whether objects correctly represent real-world entities. However, we assume that users can verify the correctness of objects by using additional information or by visiting the geographical locations of these objects. In such datasets, each object has a *correctness value* of either *true* or *false*, and a *confidence value*; yet, users do not know the correctness values. Thus, when querying uncertain datasets, users consider the confidence of an object as the probability that the correctness value of the object is *true*. Applications over uncertain datasets should be able to utilize confidence values.

Some cases in which uncertain datasets occur are integration of heterogeneous sources, incorrect updates and inexact querying. We start by describing the first case.

⋆ This author was supported by The Israel Science Foundation (Grant 893/05).

D. Papadias, D. Zhang, and G. Kollios (Eds.): SSTD 2007, LNCS 4605, pp. 276–293, 2007.

When integrating two geo-spatial sources, the result consists of pairs and singletons. A pair is correct if it comprises two objects that represent the same real-world entity in the different sources. A singleton (*i.e.,* a set that contains a single object) is correct if it represents a real-world entity that does not have a corresponding object in the other source. In the absence of keys, integration can be done by using object locations [3,4] or by using locations and additional attributes [17]. However, since locations are inaccurate, it is uncertain whether any given pair of the result is correct, that is, whether its two objects indeed represent the same real-world entity. Thus, the result of the integration is an uncertain spatial dataset.

Incorrect data manipulation can also yield uncertain datasets. The following example illustrates this.

Example 1. Consider a dataset of hotels, and suppose that no key constraint is imposed on this dataset. An incorrect insertion of data into the dataset may cause some hotel to appear twice with two different rates. In this case, users cannot know which object shows the correct rate of the hotel. Updates can cause a similar problem, for instance, when the name of some hotel is replaced with a name of a different hotel that already exists in the dataset.

Andritsos et al. [1] showed how to assign confidence values to objects in such cases.

Another important usage of uncertain datasets is representing the result of queries that contain an *imprecise condition,* namely, an adjective instead of a comparison between an attribute and a value. For example, *find good restaurants*, rather than *find restaurants that have a rating of five stars*. Additional examples are *find a luxury hotel*, *find a popular tourist site*, etc. The ability to cope with such queries is important in systems that are designed to answer requests for information, formulated by non-expert users. Such queries are useful when providing tourist and municipal information to laymen who send their request through some limited device, such as a cellular phone. When processing requests that are sent from a mobile device, one should bear in mind that the answer may depend on the location of the user.

Recently, *location-based services* have become a growing and important area in both research and commerce. Location-based services supply information through mobile devices, and the answer to a particular request depends on the location from which the request was sent, *i.e.,* the location of the mobile device [21]. For instance, a user who asks about a nearby restaurant will get different answers when the user location is in Times Square, Manhattan, and in Piccadilly Circle, London.

In this paper, we consider a specific location-based service of finding the *shortest k-route* over an uncertain dataset. In this application, the input consists of an uncertain geo-spatial dataset, a location and some k. The output is a route that starts at the given location and goes via objects of the given dataset. The route is such that the expected distance from the starting point till visiting k correct objects is minimal. The following examples demonstrate the need for providing this service.

Example 2. Consider a user located in Times Square, Manhattan, that is looking for an *inexpensive and good restaurant nearby*. The answer to this query can be a list of restaurants that presumably satisfy the request. However, it can also be an uncertain dataset that contains all the restaurants in Manhattan, such that the confidence value of

each restaurant is correlated with the likelihood that the user will consider this restaurant as inexpensive and good. Suppose that the user wants to compare three *good and inexpensive* restaurants before deciding in which one to dine. The user may also want to walk as little as possible when visiting restaurants until she sees three that she likes. In this case, the information system should find a 3-route starting at the location of the user in Times Square and going through restaurants in the dataset in a way that increases the likelihood to visit three inexpensive, good restaurants after a short walk.

There are many other scenarios in which finding the shortest k-route can be useful. For instance, before leasing or buying a house, it may be reasonable to visit and compare several options, and to do that efficiently means to go through a short route. Also, for planing a tour in some city or in some country, it may be useful to use such an application.

Finding the shortest k-route can be seen as the spatial version of computing top-k answers to a query. In many information-retrieval systems and also in some database applications, the result of a query contains only the top-k answers to the query. For instance, search engines on the World-Wide Web may present to users only the top 1000 results out of the millions of answers to a query. In geographical applications, answers should not be ranked merely according to how well they match the query. Objects should be returned with a recommended route to take in order to visit them. Moreover, choosing such a route may have an influence on how the objects are ranked in the answer to the user. The shortest k-route that we propose in this paper is one way of doing it.

The problem of finding the shortest k-route is a generalization of the traveling salesman problem (TSP). TSP (in the version where the salesman need not return to the origin) is the same as finding the shortest k-route in the case where there is no uncertainty (*i.e.,* all the objects have confidence equal to one) and k is equal to the number of objects in the dataset. Since TSP is known to be NP-hard, we do not expect to find an efficient polynomial-time algorithm to the shortest k-route problem. Thus, we settle for heuristics.

In this paper, we introduce three novel algorithms for finding a short k-route and we explain the differences between them. We also present the results of extensive experimentation that compares the algorithms on different types of data. Our experiments were conducted on both synthetic data and real-world data.

The main contributions of this paper are as follows.

- We introduce the problem of finding the shortest k-route over uncertain geo-spatial datasets.
- We present three algorithms for finding a short k-route and explain the different behaviors of these algorithms.
- We conducted thorough experiments that show the tradeoff between effectiveness of the algorithm (*i.e.*, the ability to compute a path with a short expected length) and its efficiency.

2 Framework

In this section, we formally present our framework and the problem of finding a shortest k-route over uncertain datasets.

Uncertain Geo-Spatial Datasets. A *geo-spatial dataset* is a collection of *geo-spatial objects*. Each object has a location and 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 objects are disjoint, *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.

An uncertain geographical dataset is a pair (D, φ_c), where D is a geo-spatial dataset and $\varphi_c : D \rightarrow [0, 1]$ is a function that maps each object of D to a value between 0 and 1, called *confidence*. An *instance* of (D, φ_c) is a pair (D, τ) where $\tau : D \rightarrow \{$*true, false*$\}$ is a function that maps each object of D to a *correctness value*, which is either *true* or *false*. An uncertain dataset (D, φ_c) has $2^{|D|}$ possible instances, where $|D|$ is the number of objects in D. We consider the confidence of an objects as an indication of how likely it is for the object to be correct, *i.e.,* to be mapped to *true* by τ. To each instance $I = (D, \tau)$, we assign a probability $P(I)$ according to the confidence values of the objects: $P((D, \tau)) = [\Pi_{\{o_i|\tau(o_i)=\textit{true}\}}\varphi_c(o_i)] \cdot [\Pi_{\{o_i|\tau(o_i)=\textit{false}\}}(1 - \varphi_c(o_i))]$. When computing a route over an uncertain dataset, the actual instance is not known. Hence, the probabilities of possible instances should be taken into account.

Usually, users know only D and φ_c when querying or using uncertain data. However, when developing algorithms for uncertain data, it is important to test them on data for which τ is known in order to determine the quality of the results of each algorithm. Thus, the datasets in our experiments included full information about τ.

Shortest k-Route. Consider a dataset D with n objects $o_1, \ldots, o_n$. A *complete route* over D is a sequence $\rho = o_{i_1}, \ldots, o_{i_n}$ where $i_1, \ldots, i_n$ is some permutation of $1, \ldots, n$. The complete route ρ provides an order for traversing the objects of D. Now, suppose that we are given an instance $I = (D, \tau)$, which includes τ in addition to D. Consider a traversal that starts at some given point s and visits the objects according to ρ. For each object o, we can count the number of correct objects and the distance until we get to o. Formally, we denote by *correct*$_\rho(o_{i_j})$ the number of correct objects among $o_{i_1}, \ldots, o_{i_j}$. That is,

$$\textit{correct}_\rho(o_{i_j}) = |\{o_{i_l} \mid 1 \leq l \leq j \textit{ and } \tau(o_{i_l}) = \textit{true}\}|.$$

Also, we denote by *distance*$_\rho(s, o_{i_j})$ the distance of the path that starts at s and leads to o_{i_j} according to ρ. That is,

$$\textit{distance}_\rho(s, o_{i_j}) = \textit{distance}(s, o_{i_1}) + \Sigma_{l=1}^{j-1}\textit{distance}(o_{i_l}, o_{i_{l+1}}).$$

Given an instance $I = (D, \tau)$ and a complete route $\rho = o_{i_1}, \ldots, o_{i_n}$ over D, a *k-route* is the shortest subsequence $o_{i_1}, \ldots, o_{i_j}$ such that *correct*$_\rho(o_{i_j}) = k$; however, if such a subsequence does not exist (*i.e., correct*$_\rho(o_{i_n}) < k$), then the k-route is ρ itself. Intuitively, a k-route is a traversal that stops at the k-th correct object. We denote by k-*distance*(s, ρ, I) the distance of the k-route $o_{i_1}, \ldots, o_{i_j}$ when starting at s, that is, k-*distance*$(s, \rho, I) =$ *distance*$_\rho(s, o_{i_j})$.

For an uncertain dataset, there can be many possible instances having k-routes with different lengths. Thus, we consider an expected length rather than an exact length. Given an uncertain dataset (D, φ_c), a starting point s and a complete route ρ over D, the *expected length* of a k-route is

$$\Sigma_{I \text{ is an instance of } (D,\varphi_c)}[P(I) \cdot k\text{-distance}(s, \rho, I)].$$

The shortest k-route over an uncertain dataset (D, φ_c) is a complete route ρ that has an expected length smaller or equal to the expected length of any other k-route over (D, φ_c). Since computing the shortest k-route is computationally hard, our goal in this work is to provide polynomial-time algorithms for computing a short k-route.

Assessing the Quality of the Result. In this work, we present three algorithms to the problem of finding a short k-route. In order to assess the quality of the results of these algorithms, we compare the expected length of the k-routes that the different algorithms compute. An algorithm A_1 is considered better than algorithm A_2 with respect to an uncertain dataset (D, φ_c) and a starting point s, if the expected length of the k-route produced by A_1 is shorter than the expected length of the k-route produced by A_2. Given a digital map that contains D, algorithm A_1 is better than A_2 for (D, φ_c) if the number of points s (of the map) for which A_1 is better than A_2 is greater than the number of points s for which A_2 is better than A_1.

3 Algorithms

In this section, we present three novel algorithms for finding a short k-route. We use the following notation when presenting the algorithms. We denote by (D, φ_c) the given uncertain dataset and by $o_1, \ldots, o_n$ the objects of D. We denote by s the location where the traversal should start. The result of the algorithms is a sequence $o_{i_1}, \ldots, o_{i_n}$ that defines a complete route.

3.1 The Greedy Algorithm

In the greedy algorithm, a route is constructed iteratively. Intuitively, in each iteration, the algorithm adds (to the sequence) the object that has the best ratio of confidence to distance among the objects that have not yet been added in previous iterations. The algorithm is presented in Fig. 1. Note that when choosing which object to add, while constructing the route, objects with high confidence are preferred over objects with low confidence and near objects are preferred over far objects.

The greedy algorithm is simple and efficient. No preprocessing is required and it has $O(|D|^2)$ time complexity. It usually performs well (*i.e.*, provides a short k-route) in the following two cases. First, when k is very small. In particular, this is true for $k = 1$. Secondly, when the objects of D are uniformly distributed and there is no correlation between confidence values and locations. Intuitively, in such cases, there is no preferred direction for the first leg of the traversal (which starts at s). Hence, the initial direction chosen by the greedy algorithm is as good as any other direction, and the produced route will have an expected distance close to the optimal.

Greedy (D, φ_c, s)

Input: A dataset D with confidence values φ_c, and a start location s
Output: A route over D

1: let π be an empty sequence
2: *CurrentLocation* $\leftarrow s$
3: *NotVisited* $\leftarrow D$
4: **while** *NotVisited* $\neq \emptyset$ **do**
5: let o be the object in *NotVisited* such that $\frac{\varphi_c(o)}{\mathit{distance}(\mathit{CurrentLocation}, o)} = \max\{\frac{\varphi_c(o')}{\mathit{distance}(\mathit{CurrentLocation}, o')} \mid o' \in \mathit{NotVisited}\}$
6: add o to π
7: remove o from *NotVisited*
8: let *CurrentLocation* be the location of o
9: **return** π

Fig. 1. The greedy algorithm

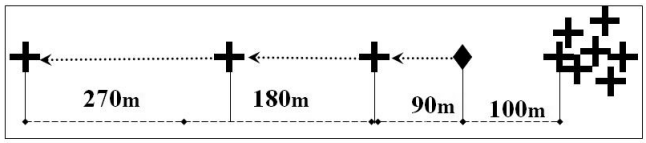

Fig. 2. An example where the greedy algorithm does not perform well. The starting point is marked by a diamond. Objects are marked by crosses.

When k is large and the distribution of either the objects or their confidences is not homogeneous, the greedy algorithm is not likely to provide good results. The following example illustrates a problematic behavior of the greedy algorithm.

Example 3. Fig. 2 shows a dataset that has a cluster of objects on the right side, and three objects with growing distances between them on the left side. Suppose that all the objects have the same confidence value. Given the starting location marked by a diamond, the route computed by the greedy algorithm will first go to the three objects on the left instead of going to the cluster on the right. For $k = 4$, for instance, it is better to start the route by going to objects in the cluster on the right side.

From Example 3, we can learn that the greedy algorithm is not an approximation algorithm to the shortest k-route problem. That is, for any given positive constant c, it is possible to construct an example in which the greedy algorithm will return a k-route whose expected length is greater than the expected length of the shortest k-route multiplied by c. Constructing such an example is done by generating a dataset similar to the one in Example 3, choosing a large enough k, and appropriately adding more objects (with growing distances between them) on the left side of the starting location and adding objects to the cluster on the right side.

AAG (D, φ_c, s)

Parameter: An accuracy parameter ϵ
Input: A dataset D with confidence values φ_c, and a start location s
Output: A route over D

1: generate a weighted graph G from D and φ_c
2: generate a transition matrix P from G
3: create a uniform distribution $X_1 = (\frac{1}{n}, \ldots, \frac{1}{n})$
4: $t \leftarrow 1$
5: **while** $||P^{t+1}X_1 - P^tX_1|| \geq \epsilon$ **do**
6: $\quad t \leftarrow t + 1$
7: create from P^tX_1 a function X^s that provides an aa-value to each node
8: $\pi \leftarrow$Greedy(D, X^s, s)
9: **return** π

Fig. 3. The Adjacency-Aware Greedy algorithm

While the greedy algorithm is not an approximation algorithm to the shortest k-route problem, it is an approximation algorithm for TSP [15]. This shows that in some aspects, the shortest k-route problem is inherently different from TSP.

3.2 The Adjacency-Aware Greedy Algorithm

Dealing with clusters of objects is important in many real-world scenarios. For example, in many cities, hotels are grouped near airports or tourist sites. Restaurants are usually located in the city center, near tourist sites and in the business district. Similarly, other utilities, such as shops or municipal buildings, are usually grouped together rather than being uniformly dispersed all over the city.

When a given dataset contains clusters of objects, a good heuristic is to give precedence to points that are in a cluster over points that are not in a cluster. This, however, is not done by the greedy algorithm, as shown in Example 3. The *Adjacency-Aware Greedy Algorithm* (AAG) improves the greedy algorithm by preferring objects that are surrounded by many near objects, especially if the near objects have high confidence values. This is done by means of assigning *adjacency-aware values* (abbr. *aa-values*) to objects as follows.

The aa-value given to an object should be based not only on the distances of the other objects and their confidence values, but also on their configuration. For example, we should prefer an object that has a neighboring cluster of four objects, within a distance of 100 meters, over an object that has four neighbors, all of them at a distance of 100 meters but in four different directions.

To compute the aa-values, we represent the dataset as a graph with weighted edges. We use the weighs to compute, for each object, an aa-value that is the probability of reaching that object in a random walk on the graph. The weight of an edge (o_1, o_2) represents the probability of moving from o_1 to o_2 and is determined by the distance between the two objects and their confidence values. In a random walk on the graph, an object with many near neighbors has a higher probability to be visited than an object

with fewer near neighbors. Furthermore, an increase in the aa-value of a node raises the aa-values of its neighbors for the following reason. If a node o has a higher probability to be visited in a random walk, then there is an increased likelihood of visiting the near neighbors of o. Hence, the aa-values of objects are affected by the configuration of the dataset.

Now, we formally define the weighted graph and show how to compute the probability of reaching a node by a random walk on this graph. Given the uncertain dataset (D, φ_c), we generate a weighted graph $G = (V, E, w)$, where the set of nodes V consists of all the objects in D, the set of edges E is $D \times D$, *i.e.,* there is an edge in G between every two nodes, and w is a function that maps each edge $e = (o_1, o_2)$ of E, where $o_1 \neq o_2$, to the weight $w(e) = \frac{\varphi_c(o_2)}{\mathit{distance}(o_1,o_2)}$. For each object o, we define $w((o, o)) = 0$. A random walk over G is a stochastic process that chooses the next node to visit as follows. If we are at some node v, we randomly choose an outgoing edge of v. The probability of choosing an edge is proportional to its weight. The random walk creates a sequence $v_1, v_2, \ldots, v_t, \ldots$ of nodes. Since the walk is random, the node v_t that is visited after t steps can be any node of G—each node with a different probability. We denote by X_t the probability distribution over V of being at each node after t steps. We represent X_t as a vector of probabilities of length $|D|$. That is, $X_t[i]$ is the probability to be at node o_i after t steps.

The random walk is a memoryless process, that is, each step depends only on the last state. In other words, the probability of choosing an outgoing edge for making the next step is independent of the path that led to the current node. Hence, it is a Markov chain, which means that the random walk can be described using an $n \times n$ *transition matrix* P, such that $X_{t+1} = PX_t$ holds for every t (note that n is the number of objects in D). We denote by P_{ij} the element in the ith column and the jth row of P. The element P_{ij} is the probability to move from node o_i to node o_j. Since the choice of edges is according to their weights, we define P as follows.

$$P_{ij} = \frac{w(o_i, o_j)}{\Sigma_{j'=1}^{n} w(o_i, o_{j'})}$$

Note that $\Sigma_{i=1}^{n} P_{ij} = 1$ holds for every row j.

The transition matrix P defines an irreducible and aperiodic Markov chain. (Intuitively, irreducible means that from each node there is a non-zero probability to reach any other node, since the graph is connected; aperiodic means that for each node, 1 is the greatest common divisor of the lengths of all paths from this node to itself, since the graph is not bipartite.) So, given an initial uniform distribution $X_1 = (\frac{1}{n}, \ldots, \frac{1}{n})$, we have that $P^t X_1 \to X^s$ as $t \to \infty$, where X^s is a stationary distribution, that is, $PX^s = X^s$. For each i, the distribution X^s gives the probability to be at o_i in a random walk on G.

The AAG algorithm of Fig. 3 computes the stationary distribution X^s and then applies the greedy algorithm where X^s replaces φ_c. Computing X^s can be done as a preprocessing step. Thus, given a user request with a specific location, the time complexity of computing a route is the same as the time complexity of the greedy algorithm.

Our experiments show that the AAG algorithm improves the greedy algorithm. However, AAG has the disadvantage that the probability distribution X^s must be computed

k-EG (D, φ_c, s)

Parameter: The number k of correct objects to visit
Input: A dataset D with confidence values φ_c, and a start location s
Output: A route over D

```
1: K ← ∅
2: S ← {{o} | o ∈ D}
3: while S ≠ ∅ do
4:    for each set S in S do
5:       if k ≤ Σ_{o∈S} φ_c(o) then
6:          add S to K
7:          remove S from S
8:       else
9:          minLength ← ∞
10:         for each o ∈ D − S do
11:            let l_{S,o} be the length of the route created by a greedy algorithm for the
               starting point s and the objects S ∪ {o}, based on merely distances,
               without using confidence values (at each iteration the greedy adds to
               the path the nearest object to the current location among the objects not
               added so far)
12:            if l_{S,o} < minLength then
13:               objToAdd ← o
14:               minLength ← l_{S,o}
15:      add objToAdd to S
16: minLength ← ∞
17: chosenSet ← ∅
18: for each S in K do
19:    let l_S be the length of the route created by a greedy algorithm for the starting
       point s and the objects of S, according to distance and without using the confi-
       dence values
20:    if l_S < minLength then
21:       chosenSet ← S
22:       minLength ← l_S
23: let π be the route that starts at s and is generated by a greedy algorithm using only
    distances, over the objects of chosenSet
24: complete π to include all the objects of D using the greedy algorithm as in Fig. 1
    (when choosing which object to add use the ratio of distance and confidence)
25: return π
```

Fig. 4. The k-Expectancy Grouping algorithm

before computing a route, and hence AAG is less efficient than the greedy algorithm for datasets that change frequently. AAG also suffers from the following two problems.

1. AAG ignores k when computing the route. For instance, consider the case that is depicted in Fig. 5, assuming that all the objects have the same confidence value. There is a small cluster on the left side of the starting point and a larger cluster on the right side of the starting point. The smaller cluster is closer to the starting point

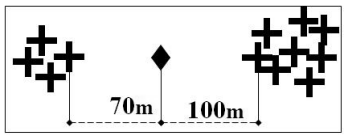

Fig. 5. An example where the AAG algorithm does not perform well. The starting point is marked by a diamond. Objects are marked by crosses.

than the larger cluster. For large values of k, it is better to go to the bigger cluster first. But for small values of k, going to the nearer (and smaller) cluster may be a better approach. In AAG, however, the same path is returned for all values of k.

2. A second problem is that by going directly to points in a cluster, there may be points on the way to the cluster, such that visiting them would not increase the distance of the route and yet, in the AAG method, such points are not always visited.

Our third method solves the above problems.

3.3 The k-Expectancy Grouping Algorithm

We now present the third method, namely, the *k-Expectancy Grouping* (k-EG) algorithm. Differently from the previous methods, the route generated by this algorithm depends not only on the dataset and the starting point, but also on the value of k. The k-EG algorithm consists of two steps. The first creates sets of objects such that the expected number of correct objects in each one is k. The second step applies the greedy algorithm to each one of these sets, and chooses the set for which the greedy algorithm generates the shortest route.

The k-EG algorithm is shown in Fig. 4. In the first part of the algorithm, sets of objects are generated and inserted into $\mathcal{K}$. The sets in $\mathcal{K}$ are constructed so that the sum of confidence values, of the objects in each set, is greater than k. This means that for the sets in $\mathcal{K}$, the average number of correct objects is at least k. Initially, $\mathcal{K}$ is empty.

The algorithm uses $\mathcal{S}$ to store sets that are eventually moved to $\mathcal{K}$. Initially, for each object o in D, the set $\{o\}$ is in $\mathcal{S}$. Then, we iteratively extend the sets in $\mathcal{S}$ by adding one object at a time, as described below. When a set has (for the first time) a confidence sum that is at least k, it is moved to $\mathcal{K}$. In order to extend a set S of $\mathcal{S}$ by one object, we examine all the objects o of D that are not yet in S. For each object o, we compute a route that starts at s and traverses the objects of $S \cup \{o\}$. This route is computed by a greedy algorithm that uses ordinary distances (i.e., it is essentially the same algorithm as in Fig. 1, except that all the confidence values are equal to 1). The object o for which the constructed route is the shortest is the one that is added to S.

After constructing the sets (Lines 1–15), we choose the one that has the shortest route (Lines 16–22). Then, a route is created from the chosen set by applying the greedy algorithm with ordinary distances. After traversing all the objects of the chosen set, we continue the route by visiting all the remaining objects of D, but now we apply the greedy algorithm that uses the ratio of the confidence to the distance.

In general, k-EG has $O(n^5)$ time complexity, where n is the number of objects in D. To see why this is true, note that initially there are n sets in $\mathcal{S}$. Since the number of sets

in $\mathcal{S}$ does not grow, there are at most n sets in $\mathcal{S}$ during the entire run of the algorithm. Also, each set contains at most n objects. Every set can be extended at most n times, each time by choosing an object from a set of at most n possible objects. So, there are at most n^2 times of considering whether to add a certain object to a certain set, which means no more than n^3 times of computing a route using a greedy algorithm, for all the n sets. Since for each set S the greedy algorithm has an $O(|S|^2)$ running time, the total time is $O(n^5)$.

In practice, the sets in $\mathcal{S}$ are expected to have a size that is much smaller than n. It is reasonable to assume that in practical cases, the sets of $\mathcal{S}$ (and hence, also the sets in $\mathcal{K}$) have an $O(k)$ size. If we consider, for instance, the case where all the objects in D have confidence values greater than 0.5, then every set in $\mathcal{S}$ has at most $2k$ objects. Under the assumption that sets in $\mathcal{S}$ have an $O(k)$ size, the running time of the algorithm is $O(n^2k^3)$. When k is constant, we actually get an $O(n^2)$ running time.

4 Experiments

In this section, we describe the results of extensive experiments on both real-world data and synthetically-generated data. The goal of our experiments was to compare the three methods presented in Section 3, over data with varying levels of object spread and different distributions of confidence values.

4.1 Tests on Synthetic Data

We used synthetic datasets to test the differences between our algorithms. One of the synthetic datasets on which we conducted experiments is depicted in Fig. 6. In this figure, objects are marked by crosses. Potential starting points are marked by circles and have a letter (A, B or C) next to the circle. The confidence values were chosen randomly according to a Gaussian distribution (normal distribution) with mean 0.7 and standard deviation 0.1. We do not show the confidence values in Fig. 6 because in some parts of the figure, objects are too dense for writing visible numbers next to them.

For estimating the expected distance of a route ρ over some given dataset (D, φ_c), when testing the quality of some algorithm, we generated 100 instances of (D, φ_c) and computed the average distance of a k-route over these instances. That is, for every given dataset (D, φ_c), we generated 100 instances $(D, \tau_1), \ldots, (D, \tau_{100})$ where each τ_i was the result of randomly choosing truth values $\tau_i(o_1), \ldots, \tau_i(o_n)$, such that $\varphi_c(o_j)$ and $1 - \varphi_c(o_j)$ were the probabilities of choosing $\tau_i(o_j)$ to be *true* and *false,* respectively. We then computed the distances $d_1, \ldots, d_{100}$, where d_i is the length of the route from the starting point to the kth correct object when traversing (D, τ_i) according to ρ. We consider the average $(\Sigma_{i=1}^{100} d_i)/100$ as the expected distance of ρ over (D, φ_c).

Fig. 8 shows the results of our algorithms when computing a route over the dataset of Fig. 6, where A is the starting point. The graph in this figure shows the expected k-distance, of the routes computed by the algorithms, as a function of k. The results of the greedy algorithm are presented by diamonds. For AAG, the results are depicted by squares, and for k-EG, the results are depicted by triangles. The graph shows that for small k values ($k = 1$ or $k = 2$), all three algorithms provide a route with a similar expected distance. For larger k values, the greedy algorithm is much worse than AAG and

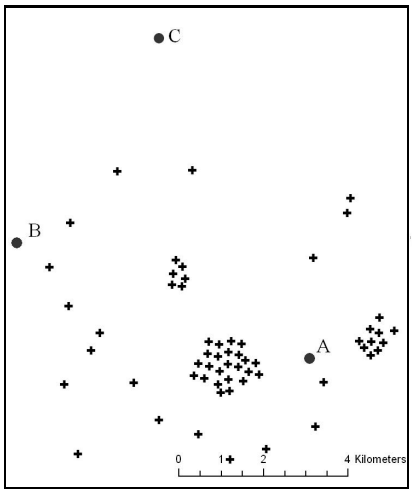

Fig. 6. A synthetic dataset

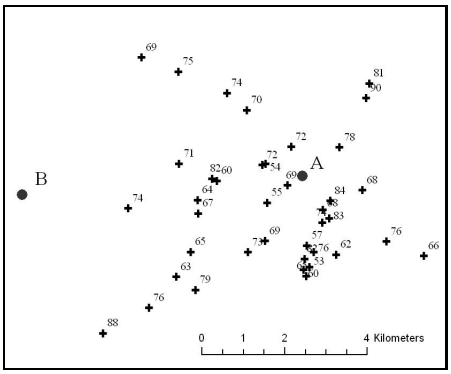

Fig. 7. A dataset of hotels in Soho, Manhattan

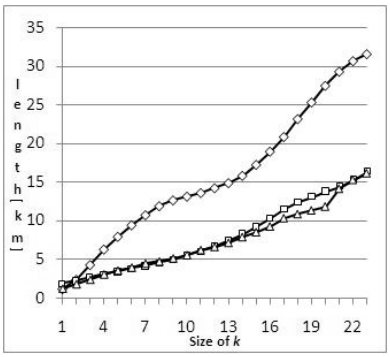

Fig. 8. Results on the dataset of Fig. 6, starting at point A

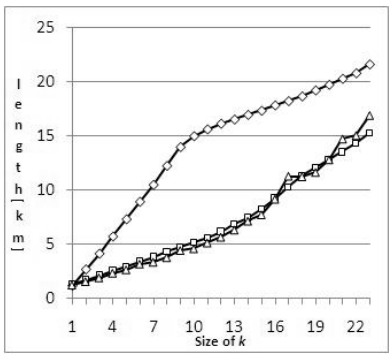

Fig. 9. Results on the dataset of Fig. 6, starting at point B

k-EG. For instance, when $k = 7$, the route of greedy algorithm has an expected length that is greater than 10 kilometers while AAG and k-EG provide routes with expected lengths of less than 5 kilometers. The differences are because AAG and k-EG generate a route that goes directly to a near cluster while the route generated by the greedy algorithm does not go directly to a cluster. For the starting point B, the differences in the quality of the results, between the greedy and the other two algorithms, are even larger, because it takes longer for the route of the greedy algorithm to get to a cluster.

Fig. 10 shows the results of our algorithms when computing a route over the dataset of Fig. 6 using C as the starting point. In this case, there is a difference between the results provided by AAG and those of k-EG. In order to understand the behavior of the different algorithms in this case, we present the routes that are computed. The greedy

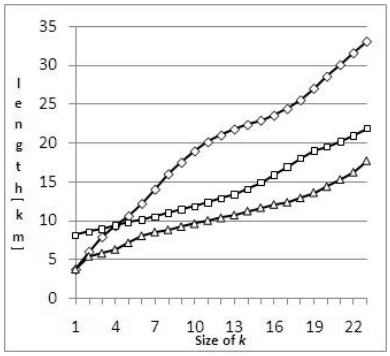

Fig. 10. Results on the dataset of Fig. 6, starting at point C

algorithm returns the route that is depicted in Fig. 11. AAG returns the route in Fig. 12. The route that k-EG returns for $k = 7$ is presented in Fig. 13. In these figures, it can be seen that the route computed by the greedy algorithm reaches a cluster after a long travel. AAG reaches a cluster directly and thus is better than the greedy algorithm for large k values. The main problems with the route that AAG computes is that it goes directly to a cluster and skips objects that are on the way to the cluster. Going through these objects increases the likelihood of reaching k correct object sooner without lengthening the route. Thus, for this case, k-EG provides a better route than AAG.

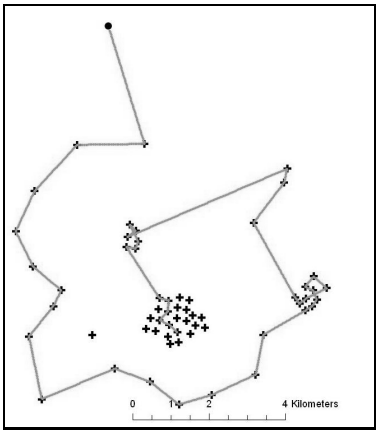

Fig. 11. The route by the greedy algorithm on the dataset of Fig. 6 starting at point C

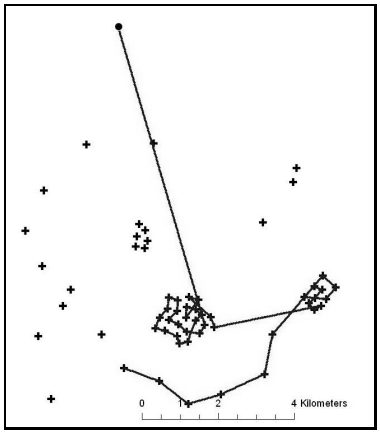

Fig. 12. The route by AAG on the dataset of Fig. 6 starting at point C

We conducted several additional tests on synthetic datasets. In these tests, we had datasets with a few large clusters, datasets with several small clusters and datasets with no clusters at all. Our experiments confirmed that in the presence of clusters, the greedy

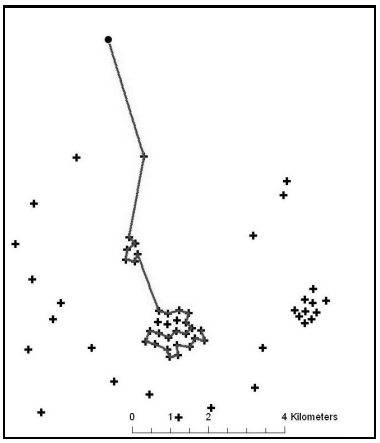

Fig. 13. The route by k-EG on the dataset of Fig. 6 starting at point C, for $k = 7$

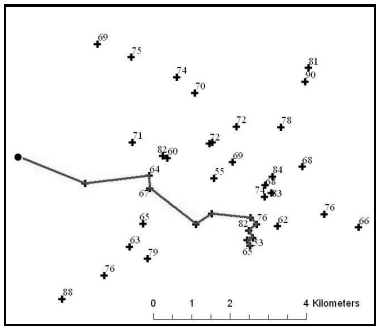

Fig. 14. The route by k-EG on the dataset of Fig. 7 starting at point B, for $k = 7$

algorithm is much worse than the other two algorithms, and they showed that k-EG provides the best results in almost all cases.

4.2 Tests on Real-World Data

We tested our algorithms on several real-world datasets to which we added confidence values. A dataset of hotels in Soho, Manhattan, is depicted in Fig. 7. The objects were taken from a map of New-York City and the confidence values were added randomly according to a Gaussian distribution with mean 0.7 and standard deviation 0.1. The results of our algorithms on this dataset are depicted in Fig. 15 and Fig. 16 for the starting points A and B, respectively. In this test, once again, the greedy algorithm provides the worst route and k-EG provides the best route, for almost all cases. The routes computed by the greedy, AAG and k-EG algorithms are depicted in Figures 17, 18 and 14, respectively.

In k-EG, a route is chosen from a set of possible routes. Intuitively, this reduces the number of cases where the algorithm produces an extremely bad route. To show it, we conducted experiments over three real-world datasets that are very different one from the other, using two distinct confidence distributions. One dataset that we used is of embassies in Tel-Aviv. In this dataset, almost all the objects are in two clusters that are quite far one from the other. A second dataset is of gas stations in the area of Tel-Aviv. This dataset contains three large clusters (dense urban areas) but also many objects that do not belong to a cluster. A third dataset that we used is of points of interest where objects are dispersed without any visible cluster. For each one of these datasets, we chose confidence values randomly. First, according to a uniform distribution in the range 0 to 1, and secondly, according to a Gaussian distribution with mean 0.7 and standard deviation 0.1. For each case, we chose a starting location.

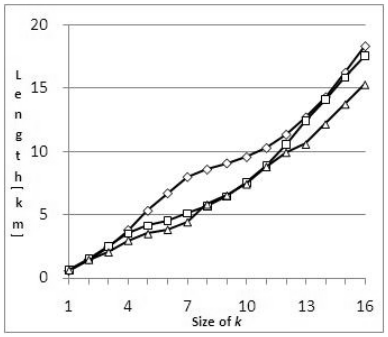

Fig. 15. Results on real-world dataset, starting at point A

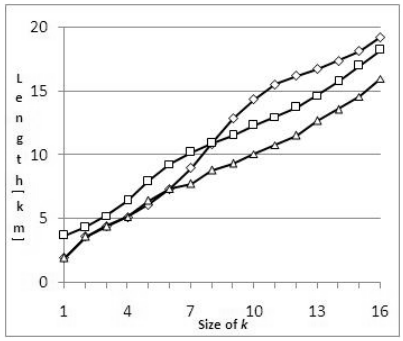

Fig. 16. Results on real-world dataset, starting at point B

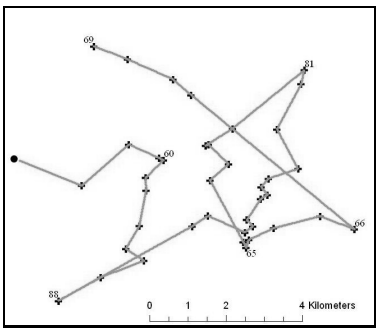

Fig. 17. The route by the greedy algorithm on the dataset of Fig. 7, starting at point B

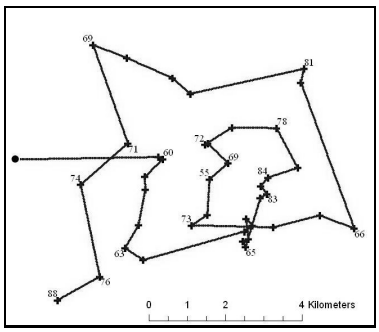

Fig. 18. The route by AAG on the dataset of Fig. 7, starting at point B

Over each dataset, we summarized for AAG and k-EG the quality of the result with respect to the result of the greedy algorithm. To do so, we computed for $k = 2, \ldots, 10$ the ratio of the distance of the route produced by the tested algorithm (AAG or k-EG) to the distance of the route produced by the greedy algorithm. We show the minimal and the maximal ratios for these cases in Fig. 19.

The graph in Fig. 19 shows that AAG sometimes generates a route that is much worse than that of the greedy algorithm. This is due to the fact that in the presence of clusters, the route generated by AAG goes directly to a cluster even when all the clusters are far from the starting point. This approach can be expensive, especially for small k values. In the presence of clusters, both AAG and k-EG sometimes produce a route that is much better than the route produced by the greedy algorithm. Not surprisingly, when there are no clusters, the differences between the algorithms are smaller. Note that we get similar results for different distributions of confidence values, but an increase in the variance of confidence values leads to an increase in the difference between the smallest and the largest ratios.

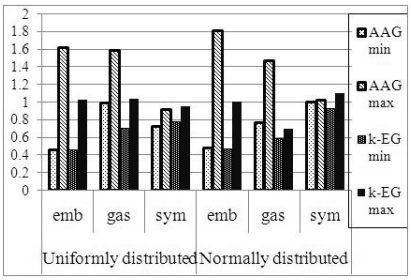

Fig. 19. The results of the algorithms summed up for several real-world sources

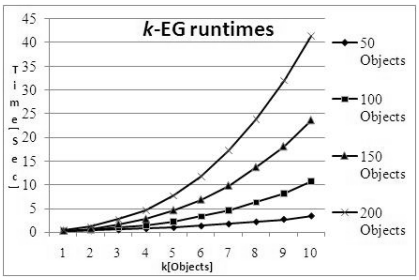

Fig. 20. The runtimes (in seconds) of k-EG as a function of k, on dataset of different sizes

4.3 Running Times

We now consider the time it takes to compute a route using our algorithms. To give running-time estimations, we measured the computation of a route on datasets of different sizes. When measuring the times, we used a PC with a Core 2 Duo, 2.13 GHz, processor (E6400) and 2GB of main memory. In Table 1, we show the time it takes to compute, using the greedy and AAG algorithms, a route over four datasets with 50, 100, 150 and 200 objects. For AAG, we show both the time it takes to compute adjacency-aware values, in the preprocessing part of the method, and the time it takes to compute a route after the preprocessing has been completed. For k-EG, we present in Fig. 20 the time for computing a route as a function of k. Table 1 and Fig. 20 show that the greedy algorithm is the most efficient among the three algorithms while k-EG is less efficient than the other two methods. AAG is less efficient than the greedy algorithm when including the preprocessing time, but without the preprocessing time, AAG is as efficient as the greedy algorithm.

Table 1. The time for computing a route over datasets of different sizes

	50 objects	100 objects	150 objects	200 objects
Greedy	<0.01 sec	0.02 sec	0.02 sec	0.02 sec
AAG preprocessing	0.02 sec	0.03 sec	0.08 sec	0.13 sec
AAG compute route	<0.01 sec	<0.01 sec	0.02 sec	0.02 sec

5 Related Work

With the ongoing advances in the areas of wireless communication and positioning technologies, it has become possible to provide mobile, location-based services. These services may track the movements and requests of their customers in multidimensional data warehouses, and later use this information for answering complex queries [10]. Data models for location-based services have been developed and implemented in recent years. An R-tree-based technique for indexing data about the current positions of objects in highly dynamic databases has been proposed by Saltenis and Jensen [18]. An

efficient search for specific information over multiple collections has been described by Goodchild and Zhou [9], who have also reported on several conceptual designs for a searching process that is based on collection-level metadata (CLM). Miller and Shaw [12] have described the use of GIS-T data models and different aspects of path finding in geospatial systems for transportation purposes.

Manipulating uncertain and probabilistic data has received a lot of attention recently. Several papers deal with managing probabilistic and uncertain data, and propose models for representing the data [2,5,8,11]. In some papers, the problem of querying probabilistic data is considered and various techniques for efficient evaluation of queries over probabilistic data are proposed [6,7,14,16,23]. The above papers are concerned with probabilistic data in general, and not with spatial data. For probabilistic spatial data, the problem of computing a join of spatial polygonal-shaped objects with imprecise locations is investigated in [13]. Computing nearest-neighbor on probabilistic spatial databases is discussed in [22]. Probabilistic spatial data has also been considered in the context of dealing with moving objects [18,19,20]. All these problems are different from the one discussed in this paper, namely, finding the shortest k-route.

6 Conclusion

In this work, we introduced the problem of finding the shortest k-route over uncertain geo-spatial datasets. Since the problem is computationally hard, we presented three heuristic algorithms for computing a short k-route, and illustrated the differences between these algorithms. We compared the algorithms using extensive experiments over synthetic and real-world data. Our experiments show that in most cases, k-EG provides the best route (*i.e.,* provides a route that is expected to lead to k correct objects within a shorter distance) and the greedy algorithm provides the worst route. However, for these algorithms, there is a tradeoff between the quality of the results and the efficiency of the algorithm. The greedy algorithm is the most efficient and k-EG is the least efficient among the three. As future work, we intend to develop optimization techniques to improve the efficiency of k-EG.

References

1. Andritsos, P., Fuxman, A., Miller, R.J.: Clean answers over dirty databases: A probabilistic approach. In: Proceedings of the 22 International Conference on Data Engineering (2006)
2. Barbara, D., Garcia-Molina, H., Poter, D.: The management of probabilistic data. IEEE Transaction on Knowledge and Data Engineering 4(5), 487–502 (1992)
3. Beeri, C., Doytsher, Y., Kanza, Y., Safra, E., Sagiv, Y.: Finding corresponding objects when integrating several geo-spatial datasets. In: ACM-GIS, Bremen, Germany, pp. 87–96. ACM Press, New York (2005)
4. Beeri, C., Kanza, Y., Safra, E., Sagiv, Y.: Object fusion in geographic information systems. In: VLDB, pp. 816–827 (2004)
5. Cavallo, R., Pittarelli, M.: The theory of probabilistic databases. In: Proceedings of 13th International Conference on Very Large Data Bases (1987)
6. Cheng, R., Kalashnikov, D., Parbhakar, S.: Evaluating probabilistic queries over imprecise data. In: Proc. of ACM SIGMOD International Conference on Management of Data, San Diego (CA, USA), ACM Press, New York (2003)

7. Dalvi, N.N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: Proceedings of the 30th International Conference on Very Large Data Bases (2004)
8. Fuhr, N.: A probabilistic framework for vague queries and imprecise information in databases. In: Proc. of the 16th International Conference on Very Large Data Bases (1990)
9. Goodchild, M.F., Zhou, J.: Finding geographic information: Collection-level metadata. Geoinformatica 7(2), 95–112 (2003)
10. Jensen, C.S., Kligys, A., Pedersen, T.B., Timko, I.: Multidimensional data modeling for location-based services. The VLDB Journal 13(1), 1–21 (2004)
11. Lakshmanan, L.V.S., Leone, N., Ross, R., Subrahmanian, V.S.: Probview: A flexible probabilistic database system. ACM Trans. on Database Systems 22(3), 419–469 (1997)
12. Miller, H.J, Shih-Lung, S.: Geographic Information Systems for Transportation: Principles and Applications (Spatial Information Systems). Oxford University Press, Oxford (2001)
13. Ni, J., Ravishankar, C.V., Bhanu, B.: Probabilistic spatial database operations. In: Proc. of the 8th International Symposium on Advances in Spatial and Temporal Databases (2003)
14. Pittarelli, M.: An algebra for probabilistic databases. IEEE Transactions on Knowledge and Data Engineering 6(2), 293–303 (1994)
15. Rosenkrantz, D.J., Stearns, R.E., Lewis II, P.M.: An analysis of several heuristics for the traveling salesman problem. SIAM Journal on Computing 6, 563–581 (1977)
16. Ross, R., Subrahmanian, V.S., Grant, J.: Aggregate operators in probabilistic databases. Journal of the ACM 52(1), 54–101 (2005)
17. Safra, E., Kanza, Y., Sagiv, Y., Doytsher, Y.: Integrating data from maps on the world-wide web. In: Proceedings of the 6th International Symposium on Web and Wireless Geographical Information Systems, pp. 180–191 (2006)
18. Saltenis, S., Jensen, C.S.: Indexing of moving objects for location-based services. In: Proceedings of the 18th International Conference on Data Engineering, Washington DC (USA) (2002)
19. Trajcevski, G., Wolfson, O., Hinrichs, K., Chamberlain, S.: Managing uncertainty in moving objects databases. ACM Transactions on Database Systems 29(3), 463–507 (2004)
20. Trajcevski, G., Wolfson, O., Zhang, F., Chamberlain, S.: The geometry of uncertainty in moving objects databases. In: Proceedings of the 8th International Conference on Extending Database Technology (2002)
21. Virrantaus, K., Markkula, J., Garmash, A., Terziyan, Y.V.: Developing GIS-supported location-based services. In: Proceedings of the 1st International Conference on Web Geographical Information Systems, pp. 423–432 (2001)
22. Zhang, S.: A nearest neighborhood algebra for probabilistic databases. Intelligent Data Analysis 4(1), 29–49 (2000)
23. Zimányi, E.: Query evaluation in probabilistic relational databases. Theoretical Computer Science 171(1-2), 179–219 (1997)