Efficient Orienteering-Route Search over Uncertain Spatial Datasets

Nir DOLEV, Yaron KANZA, and Yerach DOYTSHER, Israel

Key words: Route Search; "Path Planning; Uncertain Spatial Datasets

SUMMARY

A *route search* is a geographical search in which a route is returned rather than a list of objects. In this paper, we investigate route search over *uncertain geographical datasets.* Uncertain geographical datasets consist of uncertain geographical objects, i.e., objects for which it is unknown whether they represent correctly a real-world entity. The likelihood of an object to represent correctly a real-world entity is indicated in the dataset as a *confidence* value. Given some length *l* and a position *s*, an *orienteering route*, over an uncertain dataset, is the route with the maximal sum of confidence values among all the routes that have a length not-greater-than *l* and that start at the location *s*. An orienteering route is useful when a user needs to visit as many as possible geographic entities of some type, under a constraint on the traveling distance. We present in this paper four efficient and scalable heuristics for computing an orienteering route. Our heuristics are analyzed and tested over both real-world and syntactically generated datasets. The tests prove that it is possible to compute an orienteering route effectively and efficiently. A comparison between the methods shows that increasing the effectiveness (i.e., generating a route that provides a large sum of confidence values) has an overhead of longer running time.

Efficient Orienteering-Route Search over Uncertain Spatial Datasets

Nir DOLEV, Yaron KANZA, and Yerach DOYTSHER, Israel

1. INTRODUCTION

A geographic search, where users provide spatial and non-spatial properties, of geographical entities that the system should find, helps users to utilize geographical data on the World-Wide Web. A geographic search engine may receive general queries similar to queries of standard search engines (e.g., Google, Yahoo). However, a standard search engine returns a list of Web pages ordered according to their relevance to the search terms, yet, in a geographic search, the user may need to actually visit the discovered entities. Visiting the entities according to their order in the result may not be effective, since a path that goes through the entities may travel back and forth. Hence, we suggest an alternative solution, where objects in the search result are ordered in a way that forms a route based on both their relevance to the search and their locations. We consider such a search as a *route search*.
Route search has many important applications in various fields such as commerce, transportation, tourism, security, and health-care services. In such applications, a route search should be efficient, intuitive and expressive, allowing a user to specify complex search queries and receive an immediate answer. However, current route-search applications on the Web are limited to a point-to-point search.

Example 1.1 *Suppose that a tourist, arriving at a city, wants to find a hotel that is accessible to people on wheel chairs. She wants to see and compare several hotels; however, being tired she does not want her search to exceed a total distance of two kilometers. Moreover, she will only be able to check that a hotel is suitable for her by actually going to see it. An ordinary search for hotels that are at distance two kilometers from her current location will not yield a desired route, i.e., a route that does not exceed two kilometers and goes via hotels while maximizing the likelihood to find a suitable hotel.*

In this paper, we consider route search over uncertain datasets. Spatial data is inherently uncertain due to various reasons such as its acquisition process, imprecise modeling and manipulation (e.g., integration, incorrect updating and inexact querying) (Longley et al. 2005, Worboys et al. 2001). An uncertain dataset contains both correct and incorrect objects. We represent spatial data uncertainty by attaching to each object a *confidence value* indicating its probability to be correct (Safra et al. 2007). We consider an object as *correct* when it represents correctly a real-world entity, and we consider it as *incorrect* otherwise. A user may be able test the correctness of an object by visiting the entity at the location of that object. In Example 1.1, a user may check the correctness of an object in the search result by visiting the relevant hotel and verifying that it is represented correctly.

When computing a route, different goals and constraints can be defined, such as minimizing the traveling length, limiting the route to be over roads of a certain type, etc. In this paper, we consider a route search where the aim is finding a route that starts at a given location and

traverses through as many correct objects as possible without exceeding a given distance. This problem is a generalization of the *Orienteering Problem* (OP) (Tsiligirides 1984).
Finding a solution to OP is a problem that cannot be computed efficiently. This is because OP is a generalization of TSP (Traveling Salesman Problem); hence, it is an NP-hard (nondeterministic polynomial-time hard) problem that unlikely to have a polynomial-time algorithm (Golden et al. 1987 in Feillet et al. 2005). Our main goal in this paper is to present heuristics to OP that are efficient and scalable. This will show that building a route-search system as a Web application is realistic.
For applications on the Web, it is crucial that an answer will be returned within a few seconds. The length of the computed route is not as important as the computation time. OP has been studied in several papers (Tsiligirides 1984, Chao el al. 1996, Fischetti 1998); however, these papers were concerned mainly with the problem of computing an approximation to the problem with a bound as tight as possible. Algorithms that work over large datasets (e.g., datasets with thousands of objects) and return an immediate answer were not proposed.
In this paper, we present four efficient route-search heuristics, one of which is a variation of the AAG algorithm that was introduced by Safra et al. (2007). We use a naive *Greedy* heuristic as a benchmark for measuring the performances of the other three heuristics. Differently from the greedy approach, our heuristics lead the constructed route towards clusters of objects as soon as possible, because clusters allow collecting many objects that are likely to be correct, from a small area.
In order to show that our heuristics are efficient and effective, we tested them on both synthetically-generated probabilistic datasets and real-world dataset. Our experiments illustrate the effect of different dataset properties on a route search, such as size, density and spatial distribution. The different methods are compared based on their time and memory efficiency, scalability and effectiveness (how many correct objects are on the route).
The structure of our paper is as follows. In Section 2, we introduce four new heuristics for OP. In Section 3, our heuristics are examined and compared. We conclude in Section 4.

2. ALGORITHMS

In this section, we present our heuristics and explain the differences between them. In our effort to increase scalability and efficiency, we used a spatial grid index (mesh) in the implementation of all our methods. Thus, we describe the methods assuming the existence of such an index. We use the following notation when presenting the algorithms. We denote by D the uncertain dataset. We denote by s the location where the route starts. The bound on the route-length, i.e., the maximal length allowed for the computed route, is denoted by $L_{\max}$.

2.1 The Greedy Algorithm

The greedy algorithm constructs a route iteratively by making the most profitable increase in each step. Let P_i be the path constructed in step i and let o_i be the last object of P_i. In step 0, P_i consists of the starting point s. In step i, the algorithm checks the set N of objects that are in D and are not already in P_i. It retrieves from N the object o' such that $distance((o_i, o')) / confidence(o') \leq distance((o_i, o'')) / confidence(o'')$ for any object o'' in N. If $length(P_i)$ +

distance((o_i, *o'*)) ≤ L_{max}, it adds the edge (o_i, *o'*) to P_i and continues to step *i*+1. Else, it returns P_i.
The greedy algorithm is simple and relatively efficient. No preprocessing is required and it has O($|D|^2$) time complexity, where $|D|$ is the size of *D*. The greedy algorithm is effective (i.e., computes a good Orienteering route) when the objects of *D* are uniformly distributed and their confidence values have a small variance, i.e., when all the confidence values are approximately equal. Intuitively, in such cases, the dataset is uniform in all directions. Hence, any direction chosen by the greedy algorithm is as good as any other direction, and the produced route will have an expected prize value close to the optimal (The expected prize value is the sum of the confidence values attached to the objects on the route). However, when the dataset is not uniform, the Greedy algorithm may not provide good results, as illustrated in Fig.1. In the example shown in Fig.1, the Greedy algorithm creates a route from the objects that are on the left side with respect to the starting point, while missing the cluster on the right side. Clusters are common in geographic datasets; hence, scenarios similar to the case depicted in Fig.1 frequently happen. To deal with this, our next heuristic examines pairs of edges in each iteration rather than examining a single edge as in the Greedy algorithm.

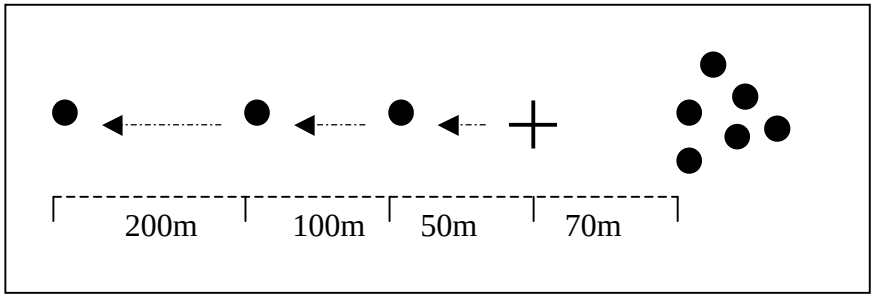


Fig.1 - A case where the greedy algorithm does not provide good results. The cross is the starting point of the route and the circles are the spatial objects of the dataset.

2.2 The Double-Greedy Algorithm (DG)

The Double-Greedy Algorithm (DG) is an improvement of the Greedy Algorithm that, intuitively, examines pairs of edges for deciding which node to add. Formally, in step *i*, the algorithm extends P_i by adding the object *o'* such that there exists *o''* for which *confidence*(*o'*)/*distance*((o_i,*o'*)) + *confidence*(*o''*)/*distance*((*o'*,*o''*)) ≥ *confidence*(*o**)/ *distance*((o_i,*o**)) + *confidence*(*o***)/*distance*((*o'*,*o***)) for any *o** and *o*** that are in *D* and are not in P_i (Note that also *o'* and *o''* are in *D* and are not in P_i).
Algorithm DG has time complexity O($|D|^3$). In order to increase efficiency, DG checks a pair of edges only when the following condition holds: α**distance*((o_{i-1},o_i)) ≤ *distance*((o_i,*o'*)), where α≥1 is a fixed factor. Intuitively this condition is satisfied when the next edge we consider to add to the route is much longer than its preceding. Frequently, this indicates that the route is leaving a cluster. When the route leaves a cluster, we want to detect this, and we want to direct the route to a new cluster.

In the case depicted in Fig. 1, DG will return a route that goes to the cluster; however, over the dataset of Fig. 2, DG will produce a route that does not lead to the cluster on the right

side. Thus, DG is not always a good heuristic. This motivates us to suggest additional heuristics.

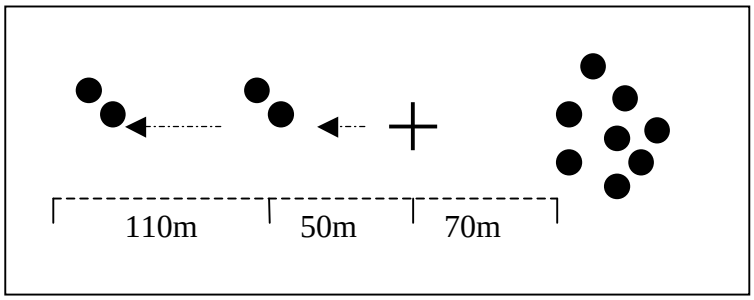


Fig.2 – An example where the route computed by DG does not lead directly to the cluster on the right.

2.3 The Adjacency-Aware Greedy Algorithm (AAG)

Clusters frequently appear in real-world datasets. For example, hotels are usually located near the coast or near tourist sites; restaurants are located in the city center, etc.
Given a dataset that contains clusters of objects, a good heuristic for constructing an OP route is to give precedence to objects that are in a cluster over objects that are not in a cluster. This is being done by the AAG method, which was first presented by Safra et al. (2007) and was designed for solving the *k-Route* problem. AAG starts by modeling the given dataset as a directed weighted graph where the objects of the dataset are the nodes and there is an edge between every pair of nodes. The weight on each directed edge is a combination of the distance between the objects and the confidence of the target node. Then, AAG computes for each node the probability of reaching this node in a random walk on the graph. Next, AAG replaces the confidence values on nodes by a combination of the confidence values and the random-walk probabilities. Finally, it applies the greedy algorithm using the new values.
Safra et al. (2007) showed that when computing a k-Route, AAG is superior to the Greedy for datasets that have clusters. The AAG improves the Greedy algorithm by giving a higher weight to objects that have many near neighbors, especially if the near neighbors have high confidence values. Although it was originally developed for solving the k-Route problem, using it for OP only requires changing the stopping condition in the last phase of the algorithm, when computing the route.
Fig.3 illustrates what happens when we apply AAG. Objects in a cluster receive a much larger weight than objects that are not in a cluster. When the cluster gets larger, the weight increases. So the route constructed over the dataset in Fig. 3 will lead directly to the cluster on the left side. This makes AAG better than the previous two algorithms. However, the route constructed by AAG can be improved by collecting the objects that are on the way to the cluster. That is, there can be objects that are not in a cluster but collecting them will have only a minor cost. The object between the starting point and the left cluster, in Fig. 3, is such an object. Making the route go via these objects can improve the result. The next heuristic improves AAG to construct a route that goes through such objects.

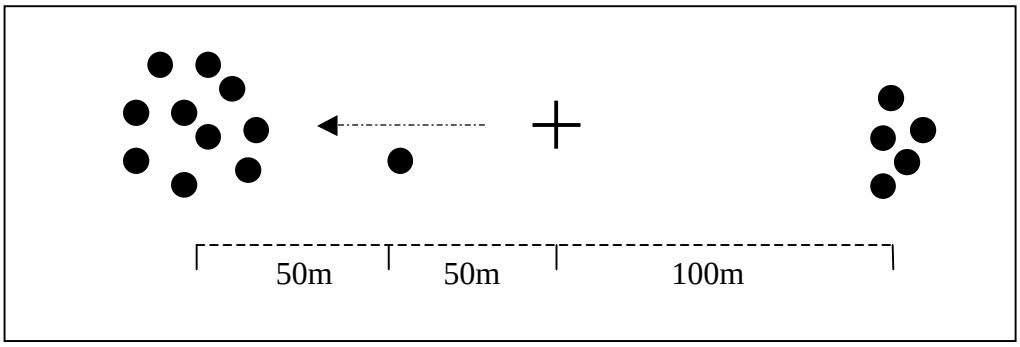


Fig.3 – An example where AAG can be improved. AAG constructs a route that goes directly to the cluster on the left without going through the objects that are on the way to the cluster.

2.4 The Adjacency-Aware Greedy Algorithm with Buffering (AAGB)

As explained in the previous section, AAG performs a greedy computation, using new weights that are computed according to the dataset topology. Now, our goal is to improve AAG by including in the route objects that are near the route of AAG. This will increase the collected prize at a slight cost in length. To do so, we start by a similar computation as in AAG, and for each edge in the route, we build a buffer. Objects that are inside the buffer are added to the route. In order to favor prize over cost, we do not want to increase the length of the route too much. Moreover, the shape of the buffer was chosen as a diamond as illustrated in Fig. 4 in order to subtle coarse turns which may occur inside a buffer crossing (Actually, an oval-shaped buffer would have been more correct mathematically, however, it is easier to manage a diamond-shaped buffer, and computations are more efficient when using a diamond buffer than when using an oval buffer).

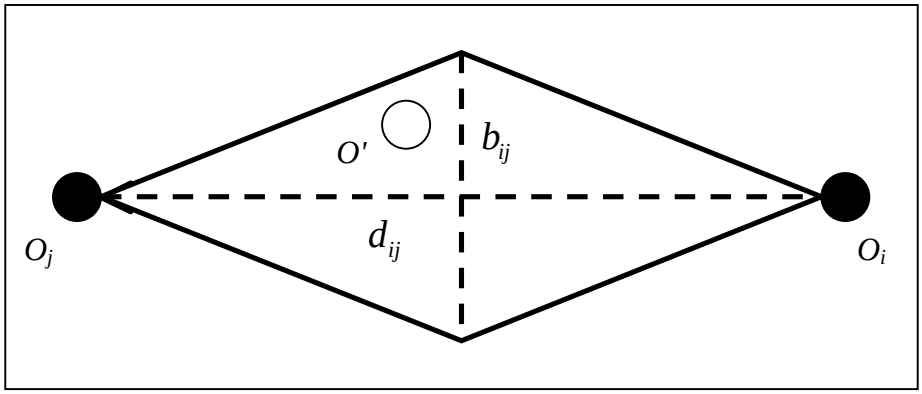


Fig.4 – Illustrating the AAGB diamond-shaped buffer.

AAGB starts by applying a pre-processing step similar to AAG (calculation of new weights). In addition, it finds the distance between every pair of objects in *D*, and it computes the mean of these distances. We denote this mean by $\hat{L}$. AAGB constructs the route greedily in the same way as AAG, but uses a buffer to add objects that are near the route constructed by AAG. The buffering is computed as follows. Suppose that we are in step *i* where the last

object so far is o_i and we increase the route by adding the object o_j. Let d_{ij} be the distance between o_i and o_j. Let b_{ij} be the width of the buffer. We compute the size of b_{ij} to guarantee that ΔL_{ij} = *distance* (o_i,o') + *distance* (o',o_j) - $d_{ij} \leq \hat{L}$. That is, the added distance by going to some object o' in the buffer (ΔL_{ij}) should not exceed the mean distance between objects in the dataset. This leads us to the following equations.

Equ.1 $\max(\Delta L_{ij}) = \sqrt{d_{ij}^2 + 4b_{ij}^2} - d_{ij}$

Forcing $\max(\Delta L_{ij})$ to be smaller than $\hat{L}$, yields the appropriate buffer size

Equ.2 $b_{ij} = 0.5\sqrt{\hat{L}\left(\hat{L} + 2d_{ij}\right)}$

After calculating the buffer, the set of objects contained in the buffer are returned and sorted according to their distance from the starting object. These objects are then added to the constructed route as long as $\Delta L_{ij} \leq \hat{L}$ and the total length of the route does not exceed $L_{\max}$.

3. EXPERIMENTS

In this section, we present the results of extensive experimentation over both synthetically generated probabilistic datasets and real world dataset. We illustrate the effect of different parameters such as the size of the dataset and the spatial distribution, on a route computation. Multiple synthetic datasets were generated for the tests, some of which will be presented. We have also examined our methods over real-world datasets. The real-world datasets were sampled by MAPA Company. They include comprehensive cover of Israel GIS layers. For both the synthetic and real world datasets, confidence values were chosen randomly according to a Gaussian distribution (normal distribution) with mean 0.7 and standard deviation 0.1.

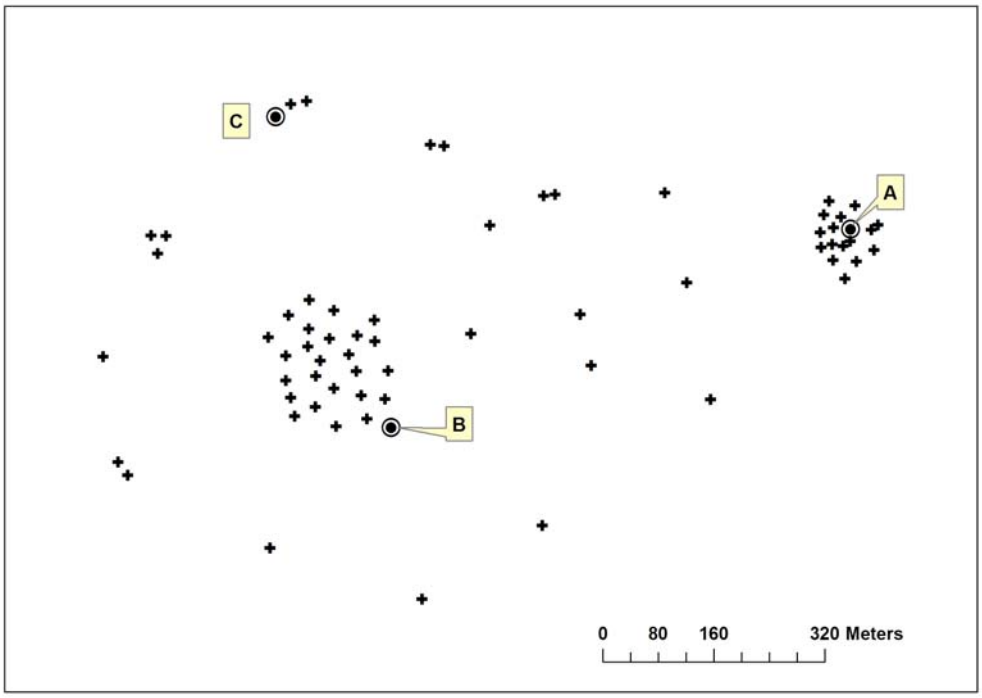


Fig.5 – Synthetic dataset. The three potential starting locations marked by indexed concentric circles (A, B and C).

3.1 Experiments with variable spatial distributions

We have performed extensive tests measuring the effect of different spatial scenarios on the route computation. In order to point out the differences between our heuristics, we will use the dataset presented in Fig.5. This dataset contains three potential starting locations marked by indexed concentric circles (A, B and C).
For each starting point, we have calculated the four heuristics paths results. Both numerical and visual results are presented below. Each OP traversal is restricted to 2000 meters.

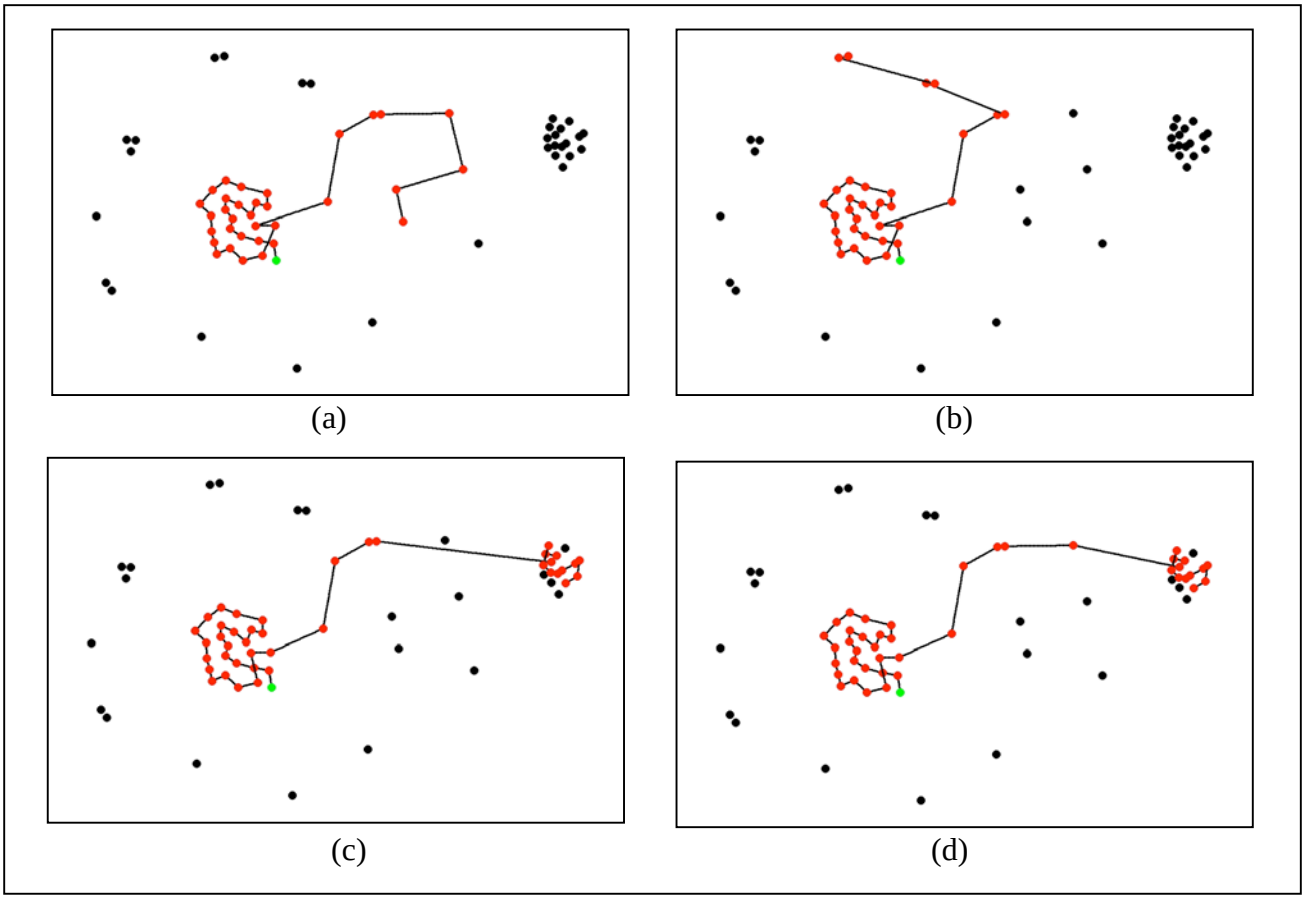
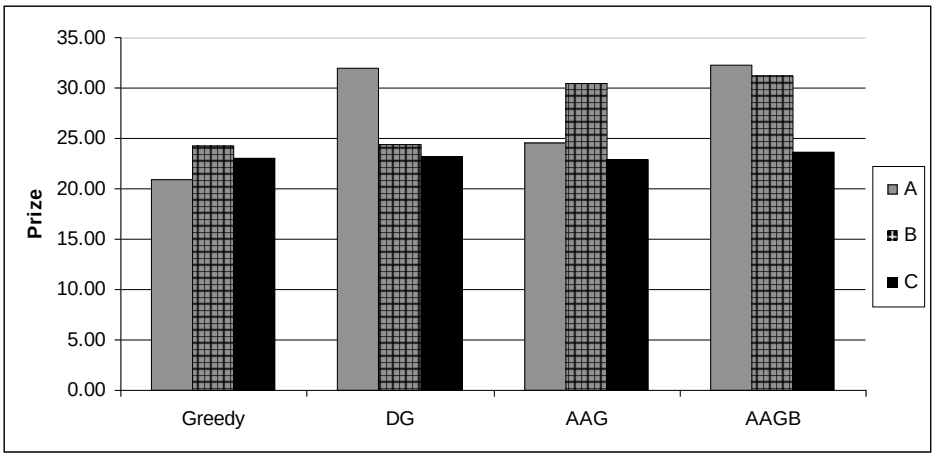


Fig.6 – Illustrating route search results for the different heuristics. The starting point is 'B' (see Fig.5). (a) The Greedy Algorithm, (b) DG, (c) AAG, and (d) AAGB.

In Fig.6, we can see the results of the four heuristics, starting at location B. It can be seen that both the Greedy and DG routes miss the second cluster, while AAG and AAGB go directly to it. Notice that AAGB, in comparison to AAG, visits another object on its way to the second cluster, presenting its ability to collect objects that are near the route of AAG.
The prizes collected by each heuristic (calculated as the sum of confidence values of the visited objects) for the three starting points A, B and C are shown in Tab.1.



Tab.1 – The prizes collected by each route for the start points A, B and C.

The results in Tab.1 help us compare the methods. Firstly, for point C, all the methods provided equally good routes. However, looking at in Fig.7 which shows the Greedy (a) and AAG (b) routes starting at C, we can see that if the distance was shorter then AAG would have been providing a better route than the Greedy algorithm because the Greedy travels around until reaching the cluster while AAG goes directly toward the cluster.

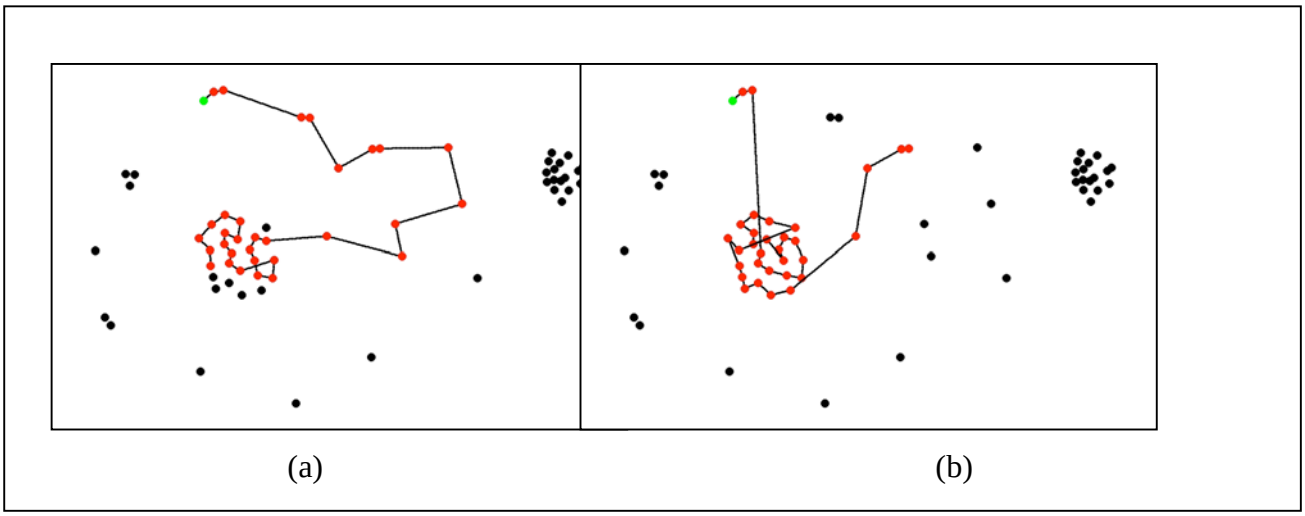


Fig.7 – The routes of the Greedy (a) and AAG (b) heuristics, starting at location C.

Secondly, for the routes that start at location A, DG provides exceptionally good results, almost as good as the results of AAGB. This case illustrates the advantage of DG over the, more naïve, Greedy algorithm. As shown in Fig.8, the Greedy algorithm deviates from its course to the big cluster, whereas DG, which examines pairs of nodes when leaving the smaller cluster, goes steadily to the cluster.

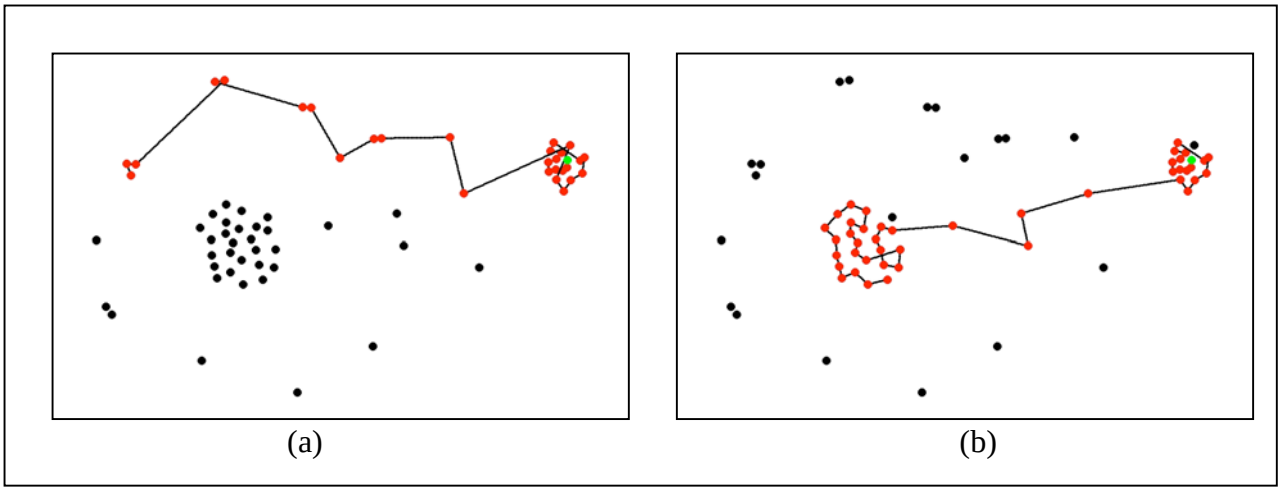


Fig.8 – Routes starting at Location A. (a) The result of the Greedy method, (b) The result of DG.

3.2 Time performance

We present now tests that examine the time performances of our methods. For these experiments, we used an HP laptop with a 1.66GHz processor (Intel T1300) and 500 MB of main memory.

Tab.2 shows the running times of the different methods, for different lengths of routes, over a dataset containing 4000 objects, where locations are uniformly distributed. It can be seen that the greedy algorithm is the most efficient, DG is almost as efficient as the greedy and the other two algorithms, i.e., AAG and AAGB, are less efficient and less scalable that the greedy and DG.

	2000[m]	**5000[m]**	**10000[m]**	**20000[m]**
Greedy	0.02[sec]	0.02[sec]	0.02[sec]	0.05[sec]
DG	0.02[sec]	0.02[sec]	0.03[sec]	0.05[sec]
AAG	0.06[sec]	0.16[sec]	0.27[sec]	0.56[sec]
AAGB	0.06[sec]	0.14[sec]	0.23[sec]	0.48[sec]

Tab.2 – The running time of the different methods.

At a glance, these results seem surprising. It is expected of AAG (and AAGB) to have similar running times as the other methods, because after the pre-processing step they perform a computation similar to the computation of the greedy algorithm. The difference lies in the new weights, being used in the AAG and AAGB that replace the original confidence values. These new weights express the probability of reaching a node in a random walk on the graph representing the dataset (Safra et al. 2007); therefore, the sum of these weights equals to one. This makes each weight relatively small. A normal NN (nearest neighbor) search of a grid index is a two-step process. First, a subset of neighbors is found by a containing bounding

box. Then, the NN is chosen among the objects discovered in the first step, by a sequential distance calculation. Dealing with probabilistic datasets changes the NN search. First, a normal NN is performed (using Euclidian distance). Then, the discovered NN determines (by its weighted distance now) the bounding box in which the weighted NN is bound, and finally a sequential search is performed over all bounded objects. This process approaches linearity when the objects weight is small and therefore the index efficiency is damaged.

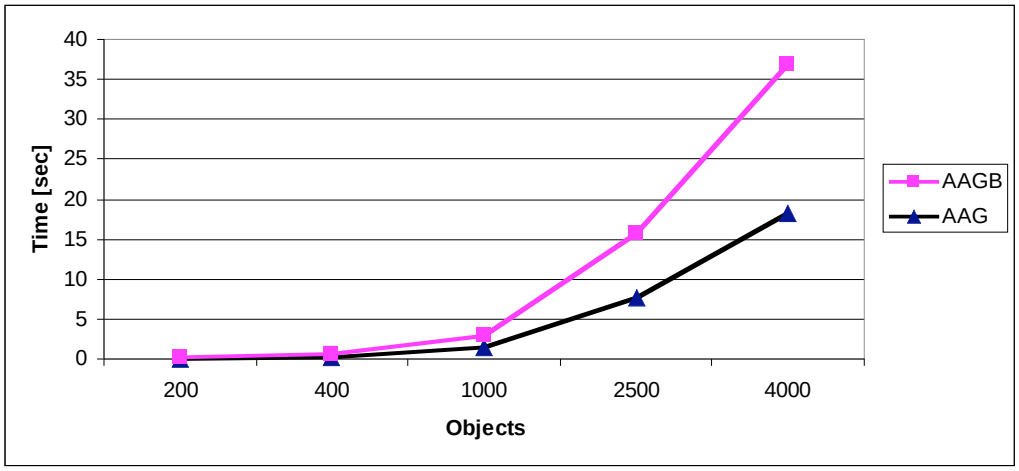


Fig.9 – The Pre-Processing Time For Aag And Aagb With Respect To Datasets Of Different Sizes.

Now, we will compare the pre-processing times of AAG and AAGB. The pre-processing times are depicted in Fig.9 as a function of the size of the dataset. It can be seen that AAGB has a pre-processing time that is about twice as large as the pre-processing time of AAG. This is due to the extra calculation of $\hat{L}$ in AAGB, which is of the same magnitude as the AAG pre-processing operation.

4. CONCLUSIONS

In this paper, we have investigated the Orienteering Problem. We presented four efficient and scalable heuristics for OP, namely Greedy, DG, AAG and AAGB. We tested our heuristics over different types of datasets to show that they are indeed efficient and scalable. The tests show that the Greedy heuristic is the most efficient and scalable among the four heuristics, but often it is less effective than the other methods, especially over datasets that have clusters. The DG heuristic is an improvement of the Greedy. It checks pairs of objects rather than a single object during the computation. Our experiments show that DG is slightly less efficient than Greedy; however it is more effective than Greedy. Yet, DG and Greedy both can miss nearby clusters when creating the route. In the AAG heuristic, for comparison, the route goes to a near cluster as soon as possible. To that end, AAG uses the topology of the dataset. Our tests show that AAG is more effective than DG and Greedy at the cost of a longer computation time. The AAGB heuristics is an improvement of AAG. It adds to the created route objects that are not in a cluster but can be added to the route at a small cost. Our tests

show that AAGB is the most effective heuristic but the least efficient among the four heuristics.
The methods we presented in this paper can provide an immediate answer to a route search over large datasets. This makes them suitable for applications where efficiency is crucial, e.g., Web applications or applications in a car navigation system. In future work we intend to examine ways to include these methods in a comprehensive route-search system.

REFERENCES

Chao I.-M., Golden B.L. & Wasil E.A. (1996). A fast and effective heuristic for the Orienteering Problem. *European Journal of Operational Research*, 88(3),475–489.

Feillet, D., Dejax, P. & Gendreau, M. (2005). Traveling salesman problems with profits, *Transportation science*, 39(2), 188–205.

Fischetti M., Salazar Gonz´alez J.J. & Toth P. (1998). Solving the Orienteering Problem through Branch-and-Cut, *INFORMS Journal on Computing*, 10(2),133–148.

Longley, P. A., Goodchild, M. F., Maguire, D. J. & Rhind, DW. (2005). *Geographic information systems and science* (2nd. ed), Chichester, UK.: John Wiley.

Safra, E., Kanza, Y., Dolev, N., Sagiv, Y. & Doytsher, Y. (2007). Computing a *k*-route over uncertain geographical data, In: *Proceedings of the10th International Symposium, SSTD*, Boston, USA.

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

Worboys, M. F. & Clementini, E. (2001). Integration of imperfect spatial information, *Journal of Visual Languages and Computing*, 12, 61-80.

BIOGRAPHICAL NOTES

Mr. Nir Dolev graduated with honor from the Technion - Israel Institute of Technology in Geodetic Engineering in 2007. He is currently a Ph.D. candidate of the department of Mapping and Geo-Information in the Technion.

Dr. Yaron Kanza has received a Ph.D. in Computer Science (2005) from the Hebrew University of Jerusalem. Previously, he received a B.Sc. in Mathematics and Computer Science, and M.Sc. in Computer Science (summa cum laude) from the Hebrew University of Jerusalem. He has been a post-doctoral fellow in the database group at the University of Toronto at the years 2004-2006. Since 2007, Dr. Kanza is a faculty staff member in the Computer-Science Department at the Technion.

Prof. Yerach Doytsher graduated from the Technion - Israel Institute of Technology in Civil Engineering in 1967. He received a M.Sc. (1972) and D.Sc. (1979) in Geodetic Engineering also from the Technion. Until 1995 he was involved in geodetic and mapping projects and consultation within the private and public sectors in Israel. Since 1996 he is a faculty staff member in Civil and Environmental Engineering at the Technion, and is currently the Dean of the Faculty of Architecture and Town Planning. He is also heading the Geodesy and Mapping Research Center at the Technion.

CONTACTS

Nir Dolev
Department of Mapping and Geo-Information Engineering
Faculty of Civil and Environmental Engineering
Technion – Israel Institute of Technology
Technion City
Haifa 32000
ISRAEL
Tel. +972-4-8292490
Email: nirdolev@tx.technion.ac.il

Dr. Yaron Kanza
Faculty of Computer Science
Technion – Israel Institute of Technology
Technion City
Haifa 32000
ISRAEL
Tel. +972-4-8294939
Email: kanza@cs.technion.ac.il

Prof. Yerach Doytsher
Department of Mapping and Geo-Information Engineering
Faculty of Civil and Environmental Engineering
Technion – Israel Institute of Technology
Technion City
Haifa 32000
ISRAEL
Tel. +972-4-8294001
Fax +972-4-8295641
Email: doytsher@technion.ac.il