Available online at www.sciencedirect.com
ScienceDirect
Progress in Natural Science 18 (2008) 893–899
Progress in Natural Science
www.elsevier.com/locate/pnsc

A shortest path algorithm for moving objects in spatial network databases

Xiaolan Yin [a,b,*], Zhiming Ding [c], Jing Li [a]

[a] Technology Center of Software Engineering, Institute of Software, Chinese Academy of Science, P.O. Box 8718, Beijing 100080, China
[b] Graduate University, Chinese Academy of Sciences, Beijing 100049, China
[c] Technology Center of Basic Software, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China

Received 7 January 2008; received in revised form 11 January 2008; accepted 14 January 2008

Abstract

One of the most important kinds of queries in Spatial Network Databases (SNDB) to support location-based services (LBS) is the shortest path query. Given an object in a network, e.g. a location of a car on a road network, and a set of objects of interests, e.g. hotels, gas station, and car, the shortest path query returns the shortest path from the query object to interested objects. The studies of shortest path query have two kinds of ways, online processing and preprocessing. The studies of preprocessing suppose that the interest objects are static. This paper proposes a shortest path algorithm with a set of index structures to support the situation of moving objects. This algorithm can transform a dynamic problem to a static problem. In this paper we focus on road networks. However, our algorithms do not use any domain specific information, and therefore can be applied to any network. This algorithm's complexity is $O(k \log_2 i)$, and traditional Dijkstra's complexity is $O((i+k)^2)$.

Keywords: Moving object; Spatial network databases; Shortest path index; Shortest path

1. Introduction

With continual growing of wireless communication and developing of positioning technology, to answer highly efficient queries of many moving objects and the queries based on location-based services (LBS) is becoming more and more important. Furthermore, the shortest path query in Spatial Network Database is one of the most widely used services.

After the initiative work of Dijkstra, a lot of researches using many methods and algorithms to solve the question of the shortest path query have been conducted [1–13]. There are two fields in the research on the shortest path algorithm: online computation and precomputation. Online computation has been addressed, for example, in [14–17]. The best bound in precomputation appears in [18]. In [19–21], the second-best algorithm on the shortest path is studied. Previous work on exact algorithms with preprocessing includes those using geometric information [22,23], hierarchical decomposition [24–26], the notion of reach [27], and A* search combined with landmark distances [28,29]. In [30], the method is to compute the best path in a time interval according to the real time. This method can solve the dynamic change of the network and the moving objects, but it needs too many resources in each computation. Therefore, the computation time is too long when the network scale is bigger and it cannot provide the real-time result. In Ref. [31], it is discussed how to re-compute the path quickly if the network structure is changed when the shortest path is computed.

In this paper, we present the algorithm in the transportation network, but our algorithm does not use any information related to the network environment. Therefore, the

* Corresponding author. Tel.: +86 10 62661582 635.
E-mail address: xlyin@otcaix.iscas.ac.cn (X. Yin).

1002-0071/$ - see front matter © 2008 National Natural Science Foundation of China and Chinese Academy of Sciences. Published by Elsevier Limited and Science in China Press. All rights reserved.
doi:10.1016/j.pnsc.2008.01.031

algorithm would be able to be used in other network environments.

We now discuss some work related to precomputation below. There are two composite parts in these algorithms: preprocess algorithm to compute the auxiliary data, and query algorithm to compute the shortest path with the auxiliary data.

Gutman [27] defined Reach of a vertex. Simply put, Reach of a vertex is a number. The value of Reach is bigger when the vertex is in the middle of the shortest path, otherwise the value is smaller. Gutman showed how to modify the path from the origin to the destination using Reach of a vertex (upper limit) and the distance of a vertex (lower limit). He used Euclidean distance as lower limit, and said that the efficiency could be improved by using the query based on Euclidean A*.

Goldberg and Harrelson [28,29] raised that the efficiency could be improved based on the query of A* using landmark as lower limit. Therefore, they gave the ALT (A* search, landmarks, and triangle inequality) method. This method can improve the efficiency of Reach algorithm.

Sanders and Schultes [24,25] gave the Highway Hierarchy (HH) algorithm. They described this algorithm in the undigraph and presented how to expand this algorithm to the digraph. The realization and the experiment of this algorithm are based on the undigraph. We think that this algorithm would not lose efficiency in the digraph. Under this assumption, HH algorithm has the complexity of the fastest query speed, less used memory and reasonable preprocess.

There is different motive in Reach algorithm and HH algorithm. The former one is to reduce to a shortest path, and the latter one is to limit the algorithm into a smaller subgraph by using inner hierarchial paths. Even so, there is a correlation between these two algorithms.

The methods described above gave many ways to solve the query of the shortest path in a spacial network, but the destination objects are all assumed motionless in these methods. So far, there is no effective algorithm for the case that the origin and the destination are moving, for the old query algorithms cannot be used for moving query objects, especially the moving destination objects. However, the query of the shortest path for moving destination is a widely asked service, so it is a problem that needs to be solved quickly. To solve this problem we established a group of indexed structures to change the computation of the shortest path between two moving points into the computation of the shortest path between two motionless sides. The computation speed is improved, and the computation complexity is reduced greatly. In the transportation network, like that in the metropolis of Beijing and Shanghai, the shortest path can be given out in real time by using our method.

2. Network model

In Spatial Network Database, moving objects are limited by the network, so we need to modelize the network first. The network is modelized to a digraph $G = (R, J, C)$, in which J is a group of vertices. We assume that (J_i, J_j) is the single directed edge from J_i to J_j connecting any two different points J_i and J_j in J. J_i is the starting point of the edge, and J_j is the ending point of the edge. The edge does not pass any other vertices in J (if the directed edge passes any other vertex different from J_i and J_j, we suppose that (J_i, J_j) is an empty set). Then R stands for the set of all these single directed edges, and every element (J_i, J_j) in R corresponds to one and only one single directed edge. C is a weighted function, and every directed edge in R corresponds to a weight of a positive real number.

In the above digraph G, the path from origin edge to destination edge $p(u, v)$ consists of a group of edge series $(r_0, \ldots, r_k)$, in which $u = r_0$, $v = r_k$, $\forall i$, $r_i \in R(0 \leqslant i \leqslant k-1)$. The starting point of r_i is the same as the ending point of r_{i+1}, and $k+1$ is the number of edges of path $p(u, v)$. A path cannot include a loop, which is $\forall r_i,\ r_j \in p(u, v)$ $(i < j)$, then the starting point of r_i is not the same as the ending point of r_j. The weight of the path is the sum of weights of all edges except origin edge and destination edge, which is $\sum_{i=2}^{k-1} C(r_i)$. If there exists a path from u to v, we can say that v can be reached from u. The distance $d_R(u, v)$ between the edges u and v that can be reached is defined as the least weight among all the paths between these two edges, otherwise, the distance $d_R(u, v)$ would be unlimited. If $k = 1$ or 0, the distance $d_R(u, v)$ is 0. The shortest path $p_R(u, v)$ between edges u and v is defined as the path with the least weight, of which the least weight $= d_R(u, v)$.

Based on the above definition of the network modelization, we can define some data types to modelize the moving objects.

A graph point $GP(r, pos)$ stands for a moving object, where $r = (J_i, J_j) \in R$, and $pos \in [0, 1]$ stands for the relative value of the position (offset) of the moving object on the edge. If $pos = 0$, the moving object is on the vertex of J_i, and if $pos = 1$, it is on the vertex of J_j. We can use *dmgp* to digressively stand for the moving trail of the moving object. *dmgp* is a series and $dmgp = ((t_i, gp_i, vm_i))_{i=1}^{n}$ where t_i is the time point, gp_i is the graph point to describe the moving object at the time of t_i, vm_i is the velocity of the moving object at the time of t_i, and $(t_i, gp_i, vm_i) = \mu_i$ is called the moving unit at the order i of *dmgp*. We assume that the moving object moves in the average velocity of the path between two moving units; therefore, the location of the moving object can be computed through interpolation.

The shortest path between two graph points $gp_1(r_1, pos_1)$ and $gp_2(r_2, pos_2)$ is defined as $p_R(gp_1, gp_2) = p_R(r_1, r_2)$.

3. The shortest path algorithm

The network after the above modelization becomes a digraph with weight, and the moving object is modelized to a graph point. The graph point moves in the digraph. Because of the movement of the object, especially that of the destination object, the computation of the shortest path

between objects becomes more complex. We lower the complexity of the computation by building the index of the shortest path to change a continually dynamic problem to a static problem.

3.1. Index structure

3.1.1. Index of the shortest path

Considering that the moving object moves in the network, and the shortest path between two points in the network can be changed to the shortest path between two edges where the two points are, we can change the computation problem of two moving points to the computation problem of static edges.

First, the edges of the modelized graph can be mapped to the vertices, and the vertices can be mapped to the edges.

Second, running Dijkstra algorithm after n times modification to get the shortest path among all vertices in the changed graph. The algorithm is as follows:

1) Initialization. The starting point can be set as:
   - o $d_s = 0$, p_s is empty;
   - o All other points: $d_i = \infty$, $p_i = ?$;
   - o Mark the starting point as s. Now $k = s$, and all other points are not marked.
2) Check the weight from all marked point k to the unmarked point j that directly is connected to k, and set $d_j = \min[d_j, d_k + l_k]$, in which l_k is the weight of vertex k. If $k = s$, then $l_k = 0$.
3) Choose next point. From the unmarked points, choose the smallest i from d_j, where $d_i = \min[d_j$, all unmarked point $j]$. Point i is chosen as the point in the shortest path, and set as marked.
4) Find the point before point i. Find the point $j*$ among the marked points as the point before i, which is connected directly to point i, then set $i = j*$.
5) Mark point i. If all vertices are marked, the algorithm is derived completely. Otherwise, mark $k = i$, and turn to step 2 to continue.

Finally, arrange all the shortest path according to the order of serial number of edges. The data structure is illustrated in Fig. 1.

3.1.2. Index of moving objects

Since the moving trails of the moving objects can be described discretely, we can build the index on the moving trails of the moving objects to search the location of moving objects. Therefore, we are ready to change the shortest path between the points to the shortest path between the edges. The index structure is indicated in Fig. 2.

3.2. The shortest path algorithm

Initial time is t_0. There are n moving objects. Origin object is p_m $(1 \leqslant m \leqslant n)$. Destination object is p_k. After time Δt, the shortest path algorithm from p_m to p_k $(1 \leqslant k \leqslant n)$ is as follows:

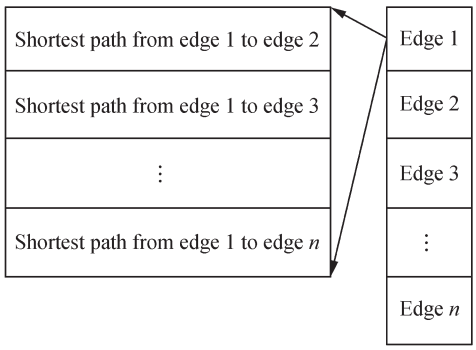

Fig. 1. Index structure of shortest path.

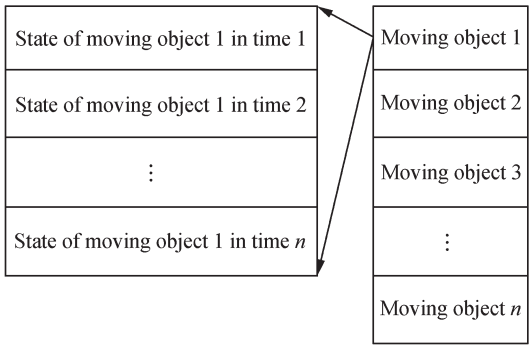

Fig. 2. Index structure of moving objects.

Input: Origin object p_m, destination object p_k, and time Δt.

Output: The shortest path.

1: $i = 1, j = n$
2: $l = [(i + j)/2]$
3: If $l = m$, then p_m is found. The search is finished. Otherwise, change the searching condition, continue to search.
4: $i = 1, j = n$
5: $l = [(i + j)/2]$
6: If $l = t_0 + \Delta t$, then find $gp_m(r_m, pos_m)$ of p_m at $t_0 + \Delta t$. The search is finished. Otherwise, change the searching condition, continue to search.
7: $i = 1, j = n$
8: $l = [(i + j)/2]$
9: If $l = k$, then find p_k. The search is finished. Otherwise, change the searching condition, continue to search.
10: $i = 1, j = n$
11: $l = [(i + j)/2]$
12: If $l = t_0 + \Delta t$, then find $gp_k(r_k, pos_k)$ of p_k at $t_0 + \Delta t$. The search is finished. Otherwise, change the searching condition, continue to search.
13: $i = 1, j = n$
14: $l = [(i + j)/2]$
15: If $l = m$, then find r_m. The search is finished. Otherwise, change the searching condition, continue to search.

16: $i = 1, j = n$
17: $l = [(i + j)/2]$
18: If $l = k$, then find the shortest path between r_m and r_k. The search is finished. Otherwise, change the searching condition, then continue to search.

According to what is described above, the main idea of the algorithm is to change the problem of computing the shortest path between two moving points to the problem of computing the shortest path between two static edges. To change the network to a digraph through the modelization and change of the network, it is easy to build the index for shortest path. Building the index of shortest path can change the shortest path between two points to the shortest path between two edges. The continuous query would be possible through the interpolation computation on the location of the moving objects after time Δt, through modelizing and treating digressively the moving objects.

The algorithm first determines the location of the moving object after time Δt through the index of the moving objects (1–12 above). Next, it gets the shortest path of the edges between the destination object and origin object, through searching the shortest path index of the edge where the origin object is (13–18 above).

3.3. Analysis of the computation complexity

Assume that there are i vertex, n edges and k interest objects in graph G. It would take $(\log_2 n) + 1$ times at most to compute by running 1–6 above, $(\log_2 n) + 1$ times at most to compute by running 7–12, and $(\log_2 n) + 1$ times at most to compute by running 13–18. So, it would take $3(\log_2 n) + 3$ times at most to compute with our algorithm. Therefore, the estimation of the time complexity of our algorithm is $O(\log_2 n)$. We assume that there is at most one edge between two vertices, then there are at most $(n*(n-1))/2$ edges in graph G. The time complexity is $O(\log_2 i)$ in terms of i. The time complexity is $O(k \log_2 i)$ when we need to find the shortest path between origin object and k interests objects. If we use traditional Dijkstra algorithm, 13–18 would use the traditional algorithm. The destination objects would be turned into vertices, and after the computation of the shortest path in Dijkstra algorithm, the estimation of the time complexity is $O((i + k)^2)$. Therefore, the estimation of the time complexity of our algorithm $O(k \log_2 i)$ is much less than that of Dijkstra algorithm $O((i + k)^2)$. In the space complexity, the space complexity of the shortest path index is $O(n^2)$, and the space complexity of moving object index is $O(n^2)$.

4. Analysis of the experiment

Below, we will compare the experimental results obtained using the shortest path index and the results using the shortest path query with Dijkstra algorithm.

4.1. Experimental data

To run the experiment, our data are the graphs generated randomly. The generation rule is as follows:

1) There are at most 5 and at least 1 directly connected vertices to every vertex. Among them, 1% vertices are directly connected to 1 or 5 vertices, respectively, 5% vertices are directly connected to 3 vertices, and the rest 93% vertices are directly connected to 4 vertices.
2) The weight of each edge is from 1 to 10. Among them, 1% edges have weight 1 or 10, respectively, 5% edges have weight 2 or 9, respectively, 8% edges have weight 3 or 8, respectively, 15% edges have weight 4 or 7, respectively, and 21% edges have weight 5 or 6, respectively.

Five graphs were generated randomly. They are with 1000 vertices and 3942 edges, 2000 vertices and 7862 edges, 3000 vertices and 11,778 edges, 5000 vertices and 19,634 edges, and 8000 vertices and 31,404 edges, respectively.

4.2. Experimental query

We run five groups of queries against each graph with different moving objects, respectively. The shortest paths are queried with origin object p_0 and 1, 5, 10, 20, and 50 moving interests objects, and each group is queried 100 times. The origin object and interests objects are selected randomly. Our measurement standard is the average time of each query.

4.3. Experimental result

The experimental results are illustrated by Table 1 and Figs. 3 and 4.

In each part of Fig. 3, the destination objects are the same but the number of the graph nodes increase.

In each part of Fig. 4, the number of the graph nodes is the same but the destination objects increases.

From Figs. 3 and 4 we can draw the following conclusions:

First, obviously, we can see that when the destination objects and graph nodes change, the result of using the shortest path index algorithm is better than that of using directly Dijkstra algorithm.

Table 1
The time of building index

Vertices	Time (s)
1000 vertices graph	14.5
2000 vertices graph	44.6
3000 vertices graph	81.3
4000 vertices graph	185
5000 vertices graph	462.3

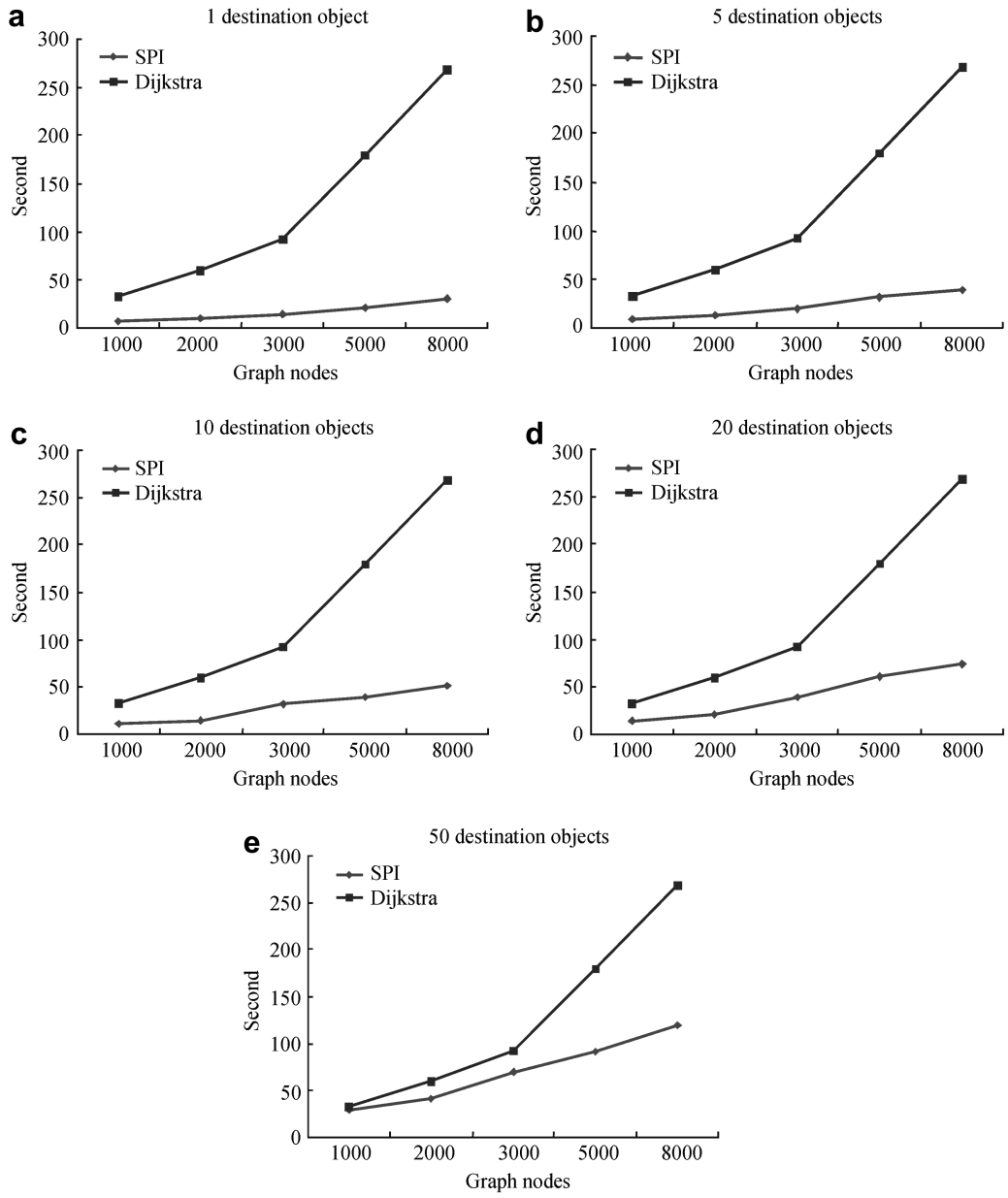

Fig. 3. The number of destination objects changed.

Second, in one same figure, as the number of the destination objects increases, the efficiency of Dijkstra algorithm keeps steady, but the efficiency of the shortest path index algorithm decreases. However, even when there are more destination objects, the efficiency of the shortest path index algorithm is higher than that of Dijkstra algorithm.

Finally, as the number of the graph nodes increases, the efficiency of both algorithms decreases, but the efficiency of the shortest path index algorithm is higher than that of Dijkstra algorithm. Even when there are fewer graph vertices, the efficiency of the shortest path index algorithm is higher.

5. Conclusions

In this study, we proposed, realized, and analyzed a shortest path query algorithm for moving objects in spatial network database. This method is a precomputation method. It stationizes the dynamic problem by building index. In the algorithm complexity and the experimental analysis, the performance of our method is better than that of the method using directly Dijkstra algorithm.

Compared with other query algorithms of the shortest path, there are the following specialities or initiatives in this paper:

(1) It solves the shortest path problem between moving objects.
(2) Through the algorithm of building the shortest path index, it changes the problem of computing the shortest path between two moving points to the problem of computing the shortest path between two motionless edges, therefore the complexity of the computation is lowered.

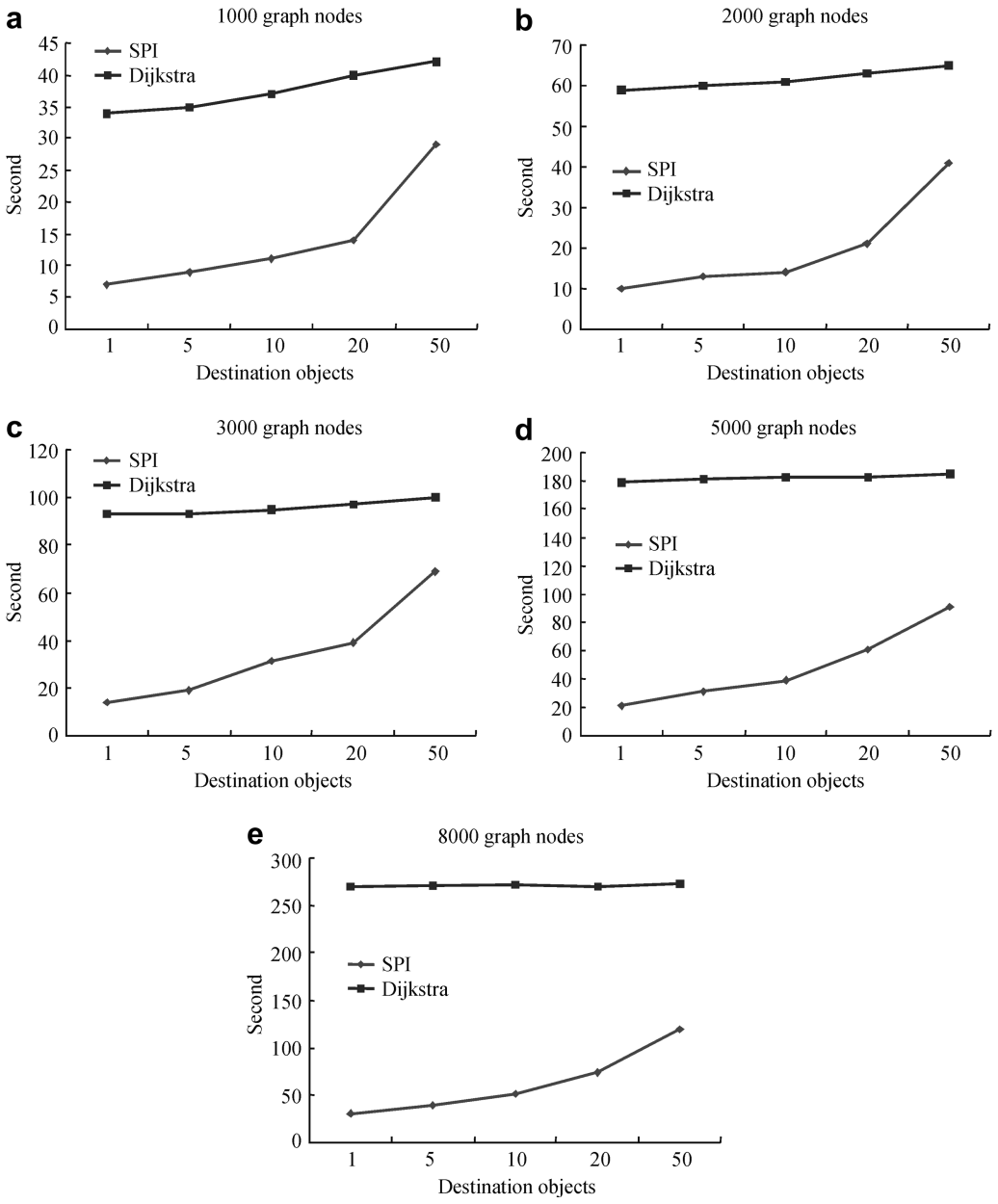

Fig. 4. The number of graph nodes changed.

(3) This algorithm does not depend on the speciality of the defined network, so it can be used in many other networks.

Acknowledgements

This work was supported by National Natural Science Foundation of China (Grant No. 60573164).

References

[1] Cherkassky BV, Goldberg AV, Radzik T. Shortest paths algorithms: theory and experimental evaluation. Math Prog 1996;73:129–74.
[2] Dantzig GB. Linear programming and extensions. Princeton: Princeton Univ. Press; 1962.
[3] Denardo EV, Fox BL. Shortest-route methods I. Reaching, pruning, and buckets. Oper Res 1979;27:161–86.
[4] Dijkstra EW. A note on two problems in connexion with graphs. Numer Math 1959;1:269–71.
[5] Fredman ML, Tarjan RE. Fibonacci heaps and their uses in improved network optimization algorithms. J Assoc Comput Mach 1987;34:596–615.
[6] Gallo G, Pallottino S. Shortest paths algorithms. Annal Oper Res 1988;13:3–79.
[7] Goldberg AV. A simple shortest path algorithm with linear average time. In: Proc. 9th ESA, lecture notes in computer science LNCS, vol. 2161, Springer; 2001, p. 230–41.
[8] Goldberg AV. Shortest path algorithms: engineering aspects. In: Proc. ESAAC '01, lecture notes in computer science. Springer; 2001.
[9] Goldberg AV, Silverstein C. Implementations of Dijkstra's algorithm based on multi-level buckets. In: Lecture notes in economics and mathematical systems 450 (refereed proceedings), Springer; 1997. p. 292–327.

[10] Jacob R, Marathe MV, Nagel K. A computational study of routing algorithms for realistic transportation networks. Oper Res 1962;10:476–99.
[11] Meyer U. Single-source shortest paths on arbitrary directed graphs in linear average time. In: Proc. 12th ACM-SIAM symposium on discrete algorithms, Washington, DC, USA; 2001. p. 797–806.
[12] Thorup M. Undirected single-source shortest paths with positive integer weights in linear time. J Assoc Comput Mach 1999;46:362–94.
[13] Zhan FB, Noon CE. Shortest path algorithms: an evaluation using real road networks. Transp Sci 1998;32:65–73.
[14] Ikeda T, Hsu MY, Imai H, et al. A fast algorithm for finding better routes by AI search techniques. In: Proc. vehicle navigation and information systems conference. IEEE, Yokohama, Japan; 1994.
[15] Pohl I. Bi-directional search. In: Machine intelligence, vol. 6. Edinburgh: Edinburgh University Press; 1971. p. 124–40.
[16] Sedgewick R, Vitter JS. Shortest paths in euclidean graphs. Algorithmica 1986;1:31–48.
[17] Zhan FB, Noon CE. A comparison between label-setting and label-correcting algorithms for computing one-to-one shortest paths. J Geograph Inf Decis Anal 2000;4(2):1–11.
[18] Fakcharoenphol J, Rao S. Planar graphs, negative weight edges, shortest paths, and near linear time. In: Proc. 42nd IEEE annual symposium on foundations of computer science, Las Vegas, Nevada, USA, 2001. p. 232–41.
[19] Cowen LJ, Wagner CG. Compact roundtrip routing in directed networks. In: Proc. symposium on principles of distributed computation, Portland, Oregon, USA, 2000. p. 51–9.
[20] Klein P. Preprocessing an undirected planar network to enable fast approximate distance queries. In: Proc 13th ACM-SIAM symposium on discrete algorithms, San Francisco, CA, USA, 2002. p. 820–7.
[21] Thorup M. Compact oracles for reachability and approximate distances in planar digraphs. In: Proc. 42nd IEEE annual symposium on foundations of computer science, Las Vegas, Nevada, USA, 2001. p. 242–51.
[22] Lauther U. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In: IfGI-prints 22. Institut fuer Geoinformatik, Universitaet Muenster (ISBN 3-936616-22-1); 2004. p. 219–30.
[23] Wagner D, Willhalm T. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In: European symposium on algorithms, Budapest, Hungary; 2003.
[24] Sanders P, Schultes D. Fast and exact shortest path queries using highway hierarchies. In: Proc. 13th annual European symposium algorithms, Palma de Mallorca, Spain; 2005.
[25] Schultes D. Fast and exact shortest path queries using highway hierarchies. Master's thesis. Department of Computer Science, Universitat des Saarlandes, Germany; 2005.
[26] Schulz F, Wagner D, Weihe K. Using multi-level graphs for timetable information. In: Proc 4th international workshop on algorithm engineering and experiments, LNCS, Springer, San Francisco, CA, USA, 2002. p. 43–59.
[27] Gutman R. Reach-based routing: a new approach to shortest path algorithms optimized for road networks. In: Proc 6th international workshop on algorithm engineering and experiments, New Orleans, Louisiana, USA, 2004. p. 100–11.
[28] Goldberg AV, Harrelson C. Computing the shortest path: A* search meets graph theory. In: Proc. 16th ACM-SIAM symposium on discrete algorithms, Vancouver, BC, Canada, 2005. p. 156–65.
[29] Goldberg AV, Werneck RF. Computing point-to-point shortest paths from external memory. In: Proc. 7th international workshop on algorithm engineering and experiments, SIAM, Vancouver, BC, Canada, 2005. p. 26–40.
[30] Moshe BA, Michel B, Bottom J, et al. Development of a route guidance generation system for real-time application. In: Proceedings of the 8th IFAC Symposium on Transportation, Chania, Greece; 1997.
[31] Hershberger J, Suri S, Bhosle A. On the difficulty of some shortest path problems. ACM Trans Algor 2007;3(1):5–15.