Applied Mathematics & Information Sciences
An International Journal

A new algorithm for shortest path problem in large-scale graph

Li Xiao[1,2] *and Lixue Chen*[1] *and Jingzhong Xiao*[2]

[1] School of Computer Science, Southwest Petroleum University, Chengdu 637001, P.R.China
[2] School of Computer Science and Technology, Southwest University for Nationalities, Chengdu 610041, P.R.China

Received: Jul. 8, 2011; Revised Oct. 4, 2011; Accepted Oct. 6, 2011
Published online: 1 Sep. 2012

Abstract: The shortest path problem is one of the basic problems in graph theory, which attracted a lot of attention of many scholars. However, with the continuous development of intelligent transportation, communications systems, many complex network structures with large-scale nature occurred, which have a larger amount of data and algorithm execution efficiency requirement, compared with the traditional shortest path problems. It first research and analyze the traditional serial A * algorithm in this article, the defect of the A * algorithm is proposed to improve. The optimization algorithm opposed in this article is named single-source algorithm. The new algorithm have lower time-complexity and more efficient processing in large-scale map compared with the A * algorithm, which take into account a variety of methods, including data preprocessing, improving the search ways, as well as the evaluation function and the internal data structure. The conclusion of the study is verified by simulation.

Keywords: A* algorithm, the shortest path, single-source algorithm, preprocessing.

1. Introduction

The shortest path problem is one of the basic problems in graph theory. According to the network features; it can be divided into two categories: static shortest path problems and dynamic shortest path problem. For single-source single-intersection of the static network, one of the most widely used is the Dijkstra algorithm. However, the A * algorithm searches the shortest path by using heuristic search, which can find the shortest path in a shorter time compared to the Dijkstra algorithm [1].

With the development of science and technology, the shortest path problem is applied in many areas frequently, the complexity of the applications have become more complicated, which demands more sophisticated algorithm for the shortest path problem.

In intelligent transportation systems, the maps with one million nodes have been very common. The number of the node in the map is increasing. Meanwhile, with the increase in the number of intelligent transportation system users, it demands less time in finding the shortest path. So, it requires a more efficient shortest path algorithm to solve all the problems listed above. There are similar problems in communication systems, the selection of data transmission path, which also requires that the algorithm can fast and efficient in large-scale networks. For different application environments and scenes, the A* algorithm must also be a modest change and improvement. The A* algorithm is analyzed in many aspects, including the complexity of algorithm theory, the actual operating efficiency, and storage space requirements. This article proposed an improved algorithm for the bottleneck of the A* algorithm, it named the single-source algorithm, which can solve the shortest path problem better in the large-scale map.

2. Algotithm theoretical basis

A * algorithm which is a subset of the heuristic search algorithm is an increase to the general heuristic search in constraints [2], in order to find the shortest path more efficiently. The data structure of A * algorithm contains an open table and a close table. The open table is used to record the candidate vertices in the next step; the close table is used to record the vertices that do not need to deal with, the close table also record the vertices contained in

* Corresponding author: e-mail: x267@163.com

the shortest paths which have been found. The A * algorithm has the following characteristics Compared to the Dijkstra [3] algorithm, (assuming that s is the source vertex, t is the intersection vertex):

The first is that the A* algorithm have an evaluation function: F (x) = G (x) + H (x). Where F (x) is the evaluated value of the vertex x, G (x) is the actual cost from vertex s to x, H (x) is an evaluated cost from vertex x to t. G (x) is always the shortest length from s to x In the general heuristic search, H (x) can take any value. The A * algorithm is different from the general heuristic search algorithm, assume that H '(x) is the evaluation function from x to t in the A * algorithm, dist (x, t) is the length of the shortest path from x to t, then x, H '(x) dist (x, t), the closer H' (x) to dist (x, t) is, the higher the search efficiency is.

The second is that the A* algorithm have open table and close table, the two tables are used to select the vertices required in the next step as well as to determine whether the vertices should be processed.

Detailed analysis of the efficiency of the A * algorithm will be given in this section.

First of all, it is the analysis of the number of visit nodes. Assume that the number of vertices in the network is fixed at 300, the experimental data obtained by changing the number of edges in Figure 1.

As shown in Fig 1, that the number of vertices visited by A* is almost equal to the number of vertices visited by Dijkstra when the number of edges is 400; the number of vertices visited by A* is significant less than the number of vertices visited by Dijkstra when the number of edges is 600, meanwhile the execute time of the two algorithms are almost equal.

When the number of edges is increasing, the number of vertices visited by the A* algorithm began to be less than the number of vertices visited by the Dijkstra algorithm.

In addition, when the network is relatively dense, the growth of running time A * is far lower than the growth rate of the edges. When the number of vertices are continue to increase, the running time will be proportional to the number of edges. As shown in Figure 2.

It can be seen from Figure 2, the number of edges are 10 | V|, | V | * sqrt (| V |),and | V | 2/10 respectively, the rate of increase of the running time is below the rate of the increase of edges from the vertical. The increase rate of the number of vertices is proportional to the increase rate of the running time from the horizontal.

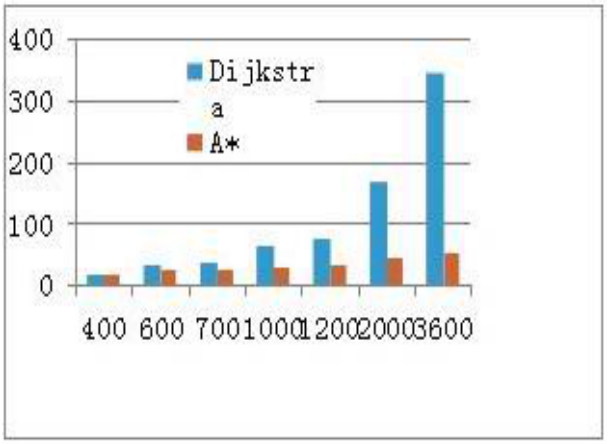

Figure 1 The comparison of edges and vertices in two algorithms.

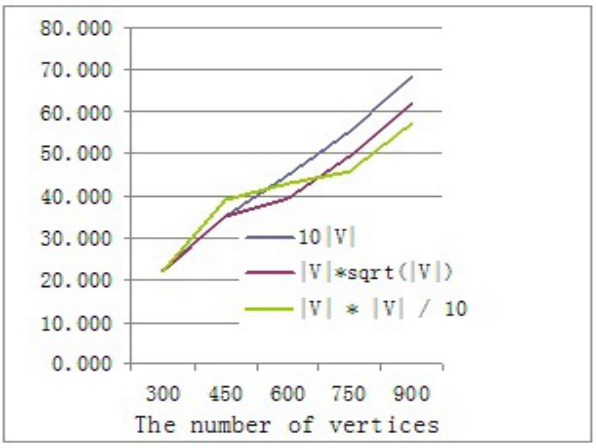

Figure 2 The relationship in runtime and vertices and edges in A* algorithm.

3. Bidirectional st algorithm

Based on the analysis above, A new shortest path algorithm is proposed in this paper, which is Bidirectional ST algorithm. The ST algorithm can be optimized in four aspects against A *:

data preprocessing, improving the way of search,improving the evaluation function, improving the structure of internal data.

The common method for data preprocessing is pretreatment on the network. Figure1 shows that when the network is relatively sparse, the number of vertices visited by ST algorithm is similar to that number in the dense network. There is another situation that the network may be not connected In addition to the different network-intensity. When the network is not connected, if calculating the shortest path between two vertices which are in different connected components, which will lead ST algorithm to traverse all the vertices in a connected component, Then the number of vertices visited by ST algorithm will increase greatly. According to this case, the networks need to be preprocessed. This pre-processing needs to use some additional storage spaces. In undirected graph, the flooding

method can be used to mark the diagram. The pseudo code of this pre-processing is as follow:

Array flag [1 ... n] = -1
cnt: = 0
For all vertices
Do
If flag [u] -1
Processing the next vertex
Else
cnt: = cnt + 1
For each vertex v, perform depth-first search from u
flag [v]: = cnt
Done

In a directed graph, the tarjan algorithm can be used to calculate the strongly connected components.

It first needs to determine whether flag[u] is equal to flag[v] during the query of the shortest paths between vertices (u,v) after the pretreatment. If they are not equal, then there is no path between the two vertices. Otherwise, if they are equal, it will call A * algorithm. The time complexity of tarjan algorithm and the flooding algorithm is equal to O (max (| V |*| E |)) [4].

The search ways in ST algorithm can be optimized properly. In general, the algorithm will start from the initial vertex to the end vertex during the search process. Similar to the BFS algorithm, ST algorithm algorithm can search in bi-direction [5,6]. The main problem of the bi-directional search is focused in the design of the terminal condition and in the design of evaluation function. It is the first that the design of h () function. Considering the bi-directional search, so it should design two h () functions of hs () and ht () for the two starting vertices respectively. It is the second that the design of the terminal condition of search, A simple way is that does not deal with the vertex u if u have visited in the reverse search when the search from the starting point access the vertex u. To calculate the value of dist (s, u) + dist (u, t), and compare the value with the length of the shortest path, and then to update the shortest path. The detailed information is specified in Figure 3 and Figure 4.

Figure 3 is the initial graph. Figure 4 represent the vertices visited after one step is performed in both of the forward and reverse search. In the next step, the forward search will search from vertex 2 to vertex 4. The vertex 4 will be ignored because it has visited in reverse search. Instead, the value of dist (1,4) + dist (4,7) will be recorded as length of the shortest path. The next vertex should be visited is vertex 5 in the forward search. The vertex 5 is also facing the same situation as vertex 4 in the reverse search. The value of dist (1, 5) + dist (5, 7) will be compared with the value of dist (1, 4) + dist (4, 7) recorded previously, and then, select the optimum value to be the length of shortest path. Should be noted that, it does not need to cross-search in bi-direction during the search process, it needs only to select open table that has less vertices every time. This allows bi-directional search share the search task. Although the method can apply better evaluation function in bi-directional search, the disadvantages

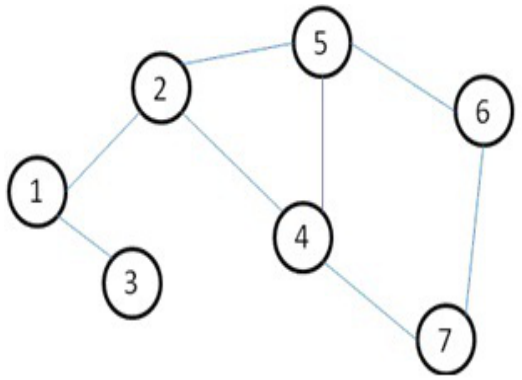

Figure 3 initial graph.

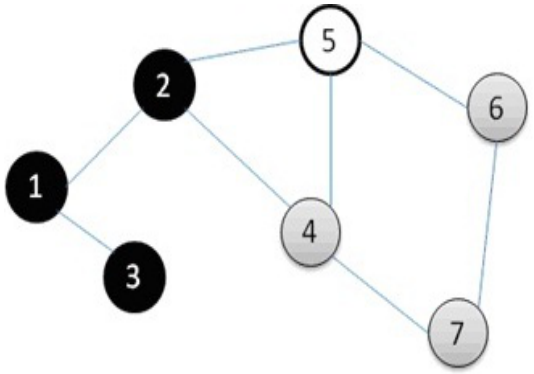

Figure 4 The example of Bi-directional search.

are obvious. The algorithm can not stop immediately, but continue to process, when the searches in both directions meet. This method shares the search task, and reduce by half the scale of the whole network, so the final theoretical complexity is O ((|V | + | E |) * log (| V| / 2)). However,due to the poor terminal condition, it can not guarantee that the process will stop immediately after finding the optimal distance. It needs appending some assessments to select the optimum value in the actual use of the process.

4. Improve the evaluation function

Considering the design of the evaluation function, the closer the value of f () and the actual shortest distance is, the fewer the points visited by the search process is. There are three methods to solve this problem: first, a better evaluation function will be designed; second, using the variant of the evaluation function; third, using multiple valuation functions.

The first method is too difficult to design. Considering the second method, assume the evaluation function is

F(x)=G(x)+wH(x), where w 1, which function can reduce the runtime compared with the traditional evaluation function in A*, but this evaluation function can not guarantee the ultimate solution is the optimum, the only thing It can guarantee is that the ultimate solution is not more than w times than the optimal solution. Other functions such as: f(x)=wg(x) + (1-w)h(x), f(x) = g(x) + wh1(x) + (1-w)h2(x), can guarantee to find the optimal solution. These functions play important roles in particular situations. For example, to find the shortest path of two vertices in the communication networks, which needs to take many factors into account such as: the shortest length of the lines, the signal attenuation during passing each vertex, in addition, as little as possible of the vertices visited is also one of the goals.

Consider the third method, set an evaluation function h = h1, h2, h3, h4 ...., for each vertex x, to calculate the value of each function in h corresponding to the vertex x, and then select the largest one among that. Although the overhead that visits each vertex is increased to some extent, it can effectively reduce the total number of vertices visited, and which is not difficult to design, the approach can eventually get the shortest path. For example, in the design of h () previously, which uses only the geometric distance between two vertices. If it uses the geometry of the triangle inequality, it will obtain a more accurate value [7]. For any three vertices u, v, w, which are connected to each other, there are dist (u, v) - dist (v, w) dist (u, w).Selecting some reference vertices to construct a reference set v1, v2, v3 .. vn, then calculating the shortest distance of the vertex to each reference vertex, as well as the shortest distance from each reference vertex to the terminal vertex t. In the search process of A *, select h (u) = max | the dist (u, v1) - dist (v1, t)|,| dist (u, v2) - dist (v2, t)| ... | the dist (u, vn) - dist (vn, t) | in the calculation of the value of h (u) corresponding to vertex u, this makes the errors between h () and the actual shortest distance to be reduced to a certain extent, thus a more accurate choice of vertex visited next step is made. Additional storage space required for this method is associated with the size of the selected reference vertices set. The additional storage space is O (n * | V |) if n basis vertices are selected. The time overhead will be divided into two sections. One is the overhead of the initialization process, which needs to calculate the distances of all vertices to the reference vertices, the time complexity is O (n * (| V | + | E|) * log | V |); The other is the calculation of h () in the search process, , the time complexity in search of the shortest distance becomes to be (n* (| V | + | E |) * log | V |). This method is mainly applied to the static networks, which needs to improve if applying it to the dynamic networks.

The heap data structure is used in A* algorithm. Although it is possible to keep the balance of the heap by inserting new vertices in empty position, which still have drawbacks. If there are three consecutive delete operations to the situation shown in Figure 5, the heap will become the one shown in Figure5:

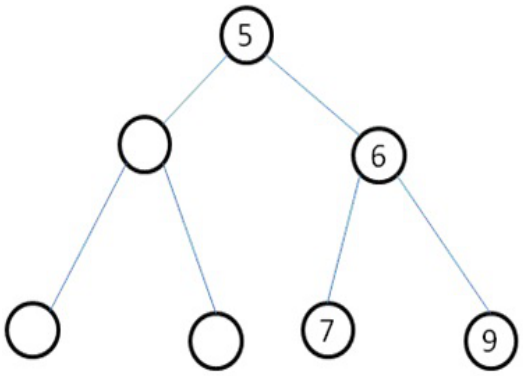

Figure 5 The examples of a heap.

Therefore, in the processing of the ST algorithm, some special situations will occur, such as that a heap is out of balance and degraded, which will decrease the query efficiency of heap greatly. The use of some other data structure can improve this situation, such as some self-balancing trees like splay, AVL, which can maintain balance of the heap after delete operation by the rotation of the entire tree.

The efficiency of Priority queue which is acted by binomial heap can still be further improved. The main idea is the introduction of the Fibonacci heap [8]. Fibonacci heap has excellent efficiency when inserting vertex, it can be done by only one step, it does not need to update the entire structure after insertion operation. ST algorithm adjusts the priority queue of internal elements, it reduces the weight of the vertex, the complexity of such operation is O (log n) operated by Binomial heap or balanced tree. Same as the insertion operation of vertex, the Fibonacci heap can do such operation by only one step. However, the Fibonacci heap is mostly discussed at the theoretical level, because of the huge additional overhead in concrete realization. If there is no targeted hardware support, the actual operating efficiency will even be lower than the binomial heap. According to the high theoretical value, the concrete realization of the Fibonacci heap with complexity closed to the theoretical value will occur in the near future, so in this paper, Fibonacci heap is described as an implementation of priority queue of ST algorithm.

Addition to the Fibonacci heap, pairing heap [9] is a data structure which combined the features of self-adjusting tree and heap. Pairing heap can complete the insertion of vertex by only one step. Different from the Fibonacci heap and Spaly heap, pairing heap is more complicated when adjusting the internal vertex weights, the complexity of which is a relatively broad boundary instead of a specific value. In spite of this, the complexity of Pairing heap is still far less than the complexity of binomial heap in adjusting operation.

The Table 1 lists the theoretical complexity of all types of data structures mentioned above.

Table 1 Time complexity of different data structures

Operation	Binomial heap	Splay	Fibonacci heap	Pairing heap
Insert	O(logn)	O(logn)	O(1)	O(1).
Delete Minimum vertex	O(logn)	O(logn)	O(logn)	O(logn).
Merge	O(nlogn)	O(logn)	O(1)	O(1).
Adjust vertex	O(logn)	O(logn)	O(1)	(loglogn).

Table.1. Time complexity of different data structures

It can be seen from Table 1, the theoretical complexity of binomial heaps and Splay is the same, considering the balance problem of the whole heap in practical application, the actual performance of Splay will be better than binomial heap. In theory, the Fibonacci heap is most suitable for the implementation of the priority queue of ST algorithm. However, its excessive complexity makes the Pairing heap more dominant in practical applications. Pairing heap is quite suitable to be used to improve the efficiency of the implementation of ST algorithm, because of its good theoretical complexity and practical performance. After the application of these data structures to the ST algorithm algorithm, the ultimate complexity of ST algorithm is changed, the binomial heaps and Splay makes complexity of ST algorithm becomes O ((| V | + | E |) log | V |), the Fibonacci heap makes the final complexity of ST algorithm to O (| E | + | V | log | V |).And the final complexity of ST algorithm which adopts the Pairing heap is O (| E | + | V | log | V |).

5. Experimental results

All time data in Table 2 are the sum of time to calculate the shortest path between all pairs of vertices in the network. The first improvement is about the initialization in ST algorithm, which can avoid the worst case of traversal of all edges in network when the network is not connected. The improvement will be more effect in sparse networks.

Table 2 The Efficiency comparison before and after preprocessing

the number of edgethe number of Vertices	Running time(no preprocessed)	Running time(preprocessed)	The ratio of performance improvement
(500,800)	12.175	10.345	17.69
(500,1000)	11.719	11.012	6.42
(500,1500)	10.226	9.878	3.52
(700,1000)	34.682	28.481	21.77
(700,1400)	31.637	28.261	11.94
(700,2100)	31.404	31.119	0.91
(700,21000)	24.832	24.805	1.08

Table.2. The Efficiency comparison before and after preprocessing

When the network is sparse, the connectivity of the network is not very good. The improved performance of the preprocessing is more obvious. When the network is dense, almost all vertices are connected. The improved performance of the preprocessing is not obvious.

Due to the change of the terminal conditions in Bidirectional search algorithm, the efficiency of bidirectional search is not good as the one-way search when it is running on a single processor, as shown in table 3. However, through the use of two processors, the search of different directions are processed parallel, the running time of which is less than the time of one-way search running on a single processor. The bi-directional search can search in two different spaces at the same time, so the two search trees can meet faster, a high speedup can be get.

Considering the efficiency of bi-directional search on a single processor, because the bi-directional search will obtain the shortest path after several renewal of the value of current shortest path, Table 4 lists the number of edges during the traversal and the times of renewal of the shortest path.

Intensive networks, bi-directional search can get the value of the shortest path at the moment the two search trees are meeting. However, the number of edges visited in the bi-directional search is growing, which mainly because that the path selected by the heuristic functions of the two directions are different from the one-way search. The huge number of edges visited is the main reason for the slow-speed of bi-directional search on a single processor. Comparing the number of edges visited and the current solution and the length of the shortest path, the conclusion can be seen distinctly from the data listed in Table 4. It can get the value of the shortest path after 395 edges visited on average. One-way ST algorithm can obtain the optimal solution after 298.635 edges visited on average. Fig 6 lists the number of edges visited in a bidirectional ST algorithm when it obtains the optimal solution at 1x, 1.025x, 1.05x, where x is the value of the shortest path.

It can be seen from Figure 6 that the bi-directional ST algorithm should continue to search for a period of time to determine the optimal solution after contained the shortest path. In the process to obtain optimal solution, the number of edges visited is almost the same in both the bi-directional ST algorithm and the one-way ST algorithm. When the higher quality of the solution is not required, the number of edges in the traversal tends to decrease in bidirectional ST algorithm, the rate of descent decreases constantly. The efficiency of bidirectional ST algorithm is poor than one-way ST algorithm when they obtain the shortest path. One reason for that is the change of the terminal condition led to the increase in the number of edges in the traversal. Other reason is the additional overhead led by the renewal of current solution.

Table 3 The experimental results by Dual-core CPU and single-core cpu

(the number of edge and Vertices	The Searching time of st using single-core CPU(s)	The Searching time of Bidirectional st using single-core CPU(s)	The Searching time of Bidirectional st using Dual-core CPU(s)
300,600	2.425	3.186	1.585
300,1000	2.236	2.93	1.593
300,5196	1.906	3.21	1.583
300,30000	1.8408	3.556	1.73
500,5000	8.096	14.145	7.01
500,11180	8.19	11.917	6.291
500,50000	8.16	14.852	7.83

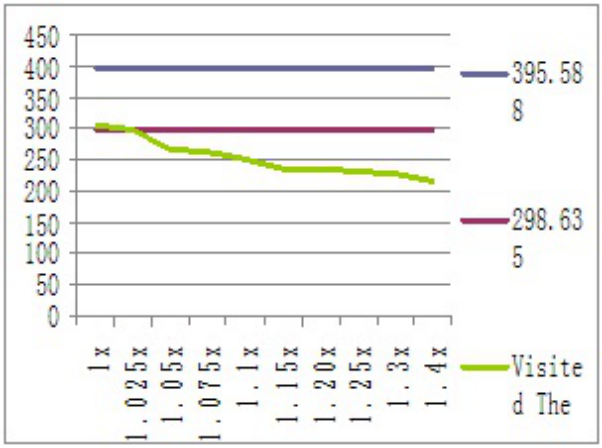

Figure 6 The number of edges visited by ST algorithm.

Table 4 Traverse the number of edges of different networks

(the number of edge and Vertices)	The Average number of edges	The Average number of updates	Proportion The number of edges	visited by bidirectional st
300,600	328.097	11.71279	3.570	314.793
300,1000	339.536	8.7624	2.580	317.515
300,5196	395.588	3.002544	0.759	298.635
300,30000	517.456	1.088622	0.210	298.337

Table 5 The experimental results of the different valuation functions

the number of edge and Vertices	0	0.2	0.4	0.6	0.8	1.0
300,1000	5.384	4.378	0.774	0.373	0.344	0.33
300,5196	12.487	12.148	3.057	0.729	0.701	0.66
300,10000	18.312	17.577	6.603	0.874	0.935	0.926
300,20000	27.218	26.719	11.123	1.25	1.383	1.247
300,30000	34.73	35.516	15.265	1.509	1.649	1.518

According to the conclusions above, the bi-directional ST algorithm can be further optimized. The approach of optimization is to constrain the number of renewals of the current solution. It can be seen from Table 5, the number of renewals is not much in general, better constraints can make the bidirectional ST algorithm to obtain the shortest path in most cases. For example, the network described as (300, 30000) in Table 5, assume that the number of renewals of the current solution is constrained to be less than 2, which can obtain the shortest path in the vast majority of query, a small part of the query can get relatively optimal solution. The performance can also slightly increase. The performance of improvement is more significant after the parallelization.

Considering the improvement of the evaluation function, this article describes how the evaluation function influences the efficiency of ST algorithm by the analysis of proximity the level between the evaluation function and the optimal value.

The variable data of the efficiency of bidirectional ST algorithm are listed in Table 5, where the value of the evaluation function is 0, 0.2, 0.4,0.8,1.0 of the length of the shortest path respectively .It can be seen from Table 5, the closer the value of evaluation function to the length of the shortest path is , the higher the overall efficiency of the bidirectional ST algorithm is .In some networks, the value of evaluation function is closer to the length of the shortest path while the efficiency becomes lower, probably due to some special vertices in the networks.

6. Conclusion

The A * algorithm for the shortest path problem was analyzed in this article. According to the defects of the A * algorithm, a new improved algorithm of A* algorithm named bi-directional ST algorithm was proposed. The simulation results show the higher efficiency of bidirectional ST algorithm in large-scale networks. The time complexity of the improved bidirectional ST algorithm is linear.

Acknowledgement

This work was supported in part by the Fundamental Research Funds for the Central Universities, Southwest Uni-

versity for Nationalities under Grant Nos. 12NZYTD14, 12NZYQN22,11NPT02 and 11NZYBS09.

References

[1] Hart, P. E. Nilsson, N. J. Raphael, IEEE Transactions on Systems Science and Cybernetics SSC **100**, 4 (1968).
[2] Aho , Hopcroft, Ullman ,The Design and Analysis of Computer Algorithms, 103 , (Pearson Education, India, 1974).
[3] Thomas, H.Cormen,Charles, E.Leiserson,Ronald, L.Rivest, Clifford, Stein,Introduction to Algorithms,24 (China Machine Press,Beijing, 2009).
[4] Tarjan, R. E, SIAM Journal on Computing **146**, 1 (1972).
[5] Nicholson, T.A.J, the computer journal **275**, 9 (1996).
[6] de Champeaux, Journal of the ACM **22**, 30 (1983).
[7] Goldberg, A.V. and Harrelson, C, Proc.16th Annual ACM-SIAM Symposium on Discrete Algorithms, 156 (2005).
[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, Introduction to Algorithms, 476 (MIT Press & McGraw-Hill, New York, 2001.).
[9] Fredman, Michael L.Sedgewick, Robert; Sleator, Daniel D.; Tarjan, Robert E, Algorithmica **111**, 1 (1986).

First Author received the BS degree in computer science from Southwest Petroleum University in 2010. She is presently employed as teacher at School of Computer Science and Technology, Southwest University for Nationalities. Her research interests are in the areas of network architecture,Algorithm Design and Analysis, and information systems.She is an active researcher coupled with the vast (10 years) teaching experience. She has published more than 30 research articles in reputed international journals of computer sciences.