SPFA를 기반으로 개선된 벨만-포드 알고리듬

진호* · 서희종**

An improved Bellman-Ford algorithm based on SPFA

Hao Chen* · Hee-Jong Suh**

요 약

이 논문에서 SPFA(shortest path faster algorithm)을 사용해서 기존의 벨만-포드(Bellman-Ford)을 개선한 효율적인 알고리듬을 제안한다. 벨만-포드 알고리듬은 딕스트라(Dijkstra) 알고리듬과 다르게 부(-)인 가중치를 갖는 그래프에서 사용할 수 있다. SPFA 알고리듬은 한 대기열을 이용하여 노드를 저장한다. 그래서 중복을 피할 수 있다. 벨만-포드 알고리듬은 시간을 더 사용하여 노드 표를 업데이트를 시킨다. 이 개산 알고리듬에서는 인접 리스트를 이용하여 표의 각 노드를 저장한다. 한 대기열을 통하여 데이트를 저장한다. 개선 방법에서는 새로운 점에 계속 relaxation을 통하여 최적 패스를 얻을 수 있다. 딕스트라 알고리듬과 SPFA 알고리듬과 개선된 알고리듬의 성능을 비교하기 위해서 시뮬레이션을 하였다. 실험 결과에서 랜덤(random) 그래프에서 개선된 알고리듬, SPFA 알고리듬과 딕스트라 알고리듬은 효율이 비슷했었는데, 격자형 지도에서 개선 알고리듬의 효율이 더 높았었다. 처리시간에서 개선된 알고리듬은 SPFA 알고리듬 보다 3분의 2를 감소시켰다.

ABSTRACT

In this paper, we proposed an efficient algorithm based on SPFA(shortest path faster algorithm), which is an improved the Bellman-Ford algorithm. The Bellman-Ford algorithm can be used on graphs with negative edge weights unlike Dijkstra's algorithm. And SPFA algorithm used a queue to store the nodes, to avoid redundancy, though the Bellman-Ford algorithm takes a long time to update the nodes table. In this improved algorithm, an adjacency list is also used to store each vertex of the graph, applying dynamic optimal approach. And a queue is used to store the data. The improved algorithm can find the optimal path by continuous relaxation operation to the new node. Simulations to compare the efficiencies for Dijkstra's algorithm, SPFA algorithm and improved Bellman-Ford were taken. The result shows that Dijkstra's algorithm, SPFA algorithm have almost same efficiency on the random graphs, the improved algorithm, although the improved algorithm is not desirable, on grid maps the proposed algorithm is very efficient. The proposed algorithm has reduced two-third times processing time than SPFA algorithm.

키워드

SPFA, Bellman-Ford, Shortest path problem, Routing
SPFA, 벨만-포드, 최단 경로 문제, 라우팅

Ⅰ. Introduction

The shortest path search in a graph is studying for long time to minimize the sum of the weights of its constituent edges, and has been very important in computer, communication and transportation network. etc, as the shortest path routing. In a packet switching network, the primary function

* 전남대학교 전자통신공학과(chenhaock@hotmail.com)
** 전남대학교 전자통신공학과(hjsuh@jnu.ac.kr)
접수일자 : 2012. 06. 21
심사(수정)일자 : 2012. 07. 26
게재확정일자 : 2012. 08. 09

is to accept packets from a source station and deliver them to a destination station through the path. Routing consists of two basic tasks. The first task is to collect the state information and keep it up-to-date. The second task is to find a satisfactory path for a new connection based on the collected information[1]. Most least cost routing algorithms used in the networks and Internet are variation of one of two algorithms, known as Dijkstra's algorithm and Bellman-Ford algorithm[2].

The Bellman-Ford algorithm solves the shortest path problem, together with the Dijkstra's algorithm. This algorithm is less restrictive condition on the graph, compared to Dijkstra's algorithm. And, the time complexity of the Bellman-Ford is $O(VE)$, and the time complexity of Dijkstra's algorithm is $O(V^2)$, where V and E are the number of vertices and edges respectively[3]. To improve solving this problem, the SPFA (shortest path faster algorithm) had been proposed, but it was an improved Bellman-Ford algorithm[4]. Its time complexity is $O(kE)$, where k is a constant. In sparse graph, the time complexity of SPFA algorithm is far less $O(V^2)$. But this algorithm requires more time in order to move the new nodes.

In this paper, a relaxation method is used for the new node, and the implementation of algorithm has been continuously iterated. And an optimized SPFA algorithm is proposed. The result of the experiment shows that the proposed algorithm has better performance, than SPFA algorithm and Dijkstra's algorithm.

In Chapter Ⅱ, SPFA algorithm is explained. In Chapter Ⅲ there is an optimized SPFA algorithm by improving the search method. In Chapter Ⅳ, there are the comparisons of the proposed algorithm with SPFA and Dijkstra's algorithm by experiments. Chapter Ⅴ provides conclusions. Next is references.

Ⅱ. Related Algorithm

In a digraph $G=(V,E)$, a set L is used to denote the adjacency list of the. The weight of each edge $l(v,k)$ is used to establish the adjacency list L.

An array set D store the current value of the shortest path from the source node to the remaining nodes. Each element of the array D is initialized to maximum value. After running SPFA algorithm, the set D output the shortest path value of each node.

The graph for SPFA algorithm is expressed as an adjacency list, and to search the shortest path a dynamic optimal approach is applied. A FIFO queue is used to store the vertex to be optimized. When optimization, a node w is removed from the queue, and a current path $D[w]$ with the node w is used to optimize the value of path $D[j]$ with another node j. If adjusted, the value of the $D[j]$ will be getting smaller, the node j will be added to the queue to be further optimized. Repeatedly the nodes from the queue is removed to optimize the current shortest path. Until the queue is empty, re-optimization will be done though searching a unnecessary shortest path. At that time, array D has stored the value of the shortest path from the source node to the other nodes.

SPFA algorithm:

for each v in V do

begin

for each $k \in L[v]$ do read $(l(v,k))$;

Read the weight of each edge to the adjacency list.

$$QM[v] := 0;$$

Initialize each vertex whether the flag array into the queue.

$$D[v] := MAX;$$

Initialize the shortest path array to the maximum

value.

end;

$$Queue \leftarrow v_0;$$

$$QM[v_0] := 1;$$

The source node v_0 into the queue.

$$D[v_0] := 0;$$

From the source node to itself, the value is zero.

while Queue not empty do

begin

$$w \leftarrow Queue;$$

Removed a node w from the queue.

$$QM[w] := 0;$$

After a node w is removed, its flag array instead of zero. It means the node w is not in the queue.

for each $j \in L[w]$ do

if $D[j] > D[w] + l(w,j)$ then

begin

$$D[j] := D[w] + l(w,j)$$

Judge the path from the node w to the node j is shorter than the original path, optimize the path with node j.

if $QM[j] = 0$ then

begin

$$Queue \leftarrow j;$$

$$QM[j] := 1;$$

When the node j is node in the queue, the flag of $QM[j]$ instead of one. It means is the node j is in the queue.

end

end

end;

for each v in V do

begin

write $D[v]$;

end;

Optimization is complete, the array D is stored the shortest path that from the source node to each node. The result will be output.

end.

End of the algorithm.

The time complexity of SPFA algorithm is $O(kE)$, where k is a constant. First in initialization of SPFA algorithm the weight of each edge is read. In this case there will be a time complexity is $O(E)$. The sum of out-degree of the vertexes V is the edges E. For one node, the average out-degree is $\frac{E}{V}$. Execution time is $O\left(\frac{E}{V}\right)$. Set the number of vertices that is added the queue m, the average of each node is added the queue once, $m = n$; the average of each node is added to the queue twice, $m = 2n$. m is a constant, it has noting to do with E and V. So the time complexity of SPFA algorithm is $T = O(kE)$.

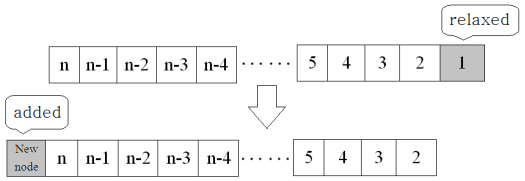

Fig. 1 Search method of SPFA algorithm

SPFA algorithm uses a FIFO queue to store the nodes. A node is removed from the first of the queue. The node is relaxed successfully, a new node will be added at the end of the queue to be further optimized. Repeatedly the node is removed to optimize the current shortest path, until the queue is empty. Fig.1 shows search method of SPFA algorithm.

III. Improving the Search Method

The data structure of the proposed algorithm also uses adjacency list as SPFA algorithm. In our algorithm, an adjacency list structure is used significantly to save memory space and reduce sea-rching time[5].

An important property is the triangle inequality. Set $G=(V,E)$ is a digraph with weights. a node s is the source node, and for any node t, $d(s,t)$ is the shortest path that from s to t. For all edges $(u,v)\in E$, $d(s,v)\leq d(s,u)+w(u,v)$ is a sufficient and necessary condition for $d(s,t)$ for the shortest path from s to t.

The proposed algorithm uses relaxation operation. First set $f(u)$ is the shortest path of node u. Assignment $f(s)=0$, $f(u)=+\infty(u\neq s)$. Then,

For $(u,v)\in E$, Relax (u,v)

If $f(v)>f(u)+w(u,v)$, $f(v)=f(u)+w(u,v)$

It means that if each edge do not satisfies the triangle inequality, the destination node will be re-assigned.

The essential of relaxation operation is an search constantly looking for conflicting between current state and target state. Adjustment of status is done until no conflicting. After that time, the optimal state is reached.

If the Bellman-Ford algorithm is used, actually only one node is updated with each iteration, and other loops are invalid. The Bellman-Ford algorithm is inefficiency, because there are a lot of unnecessary operations. It is only when the node is updated in the last iteration that will be useful for current iteration. The proposed algorithm takes advantage of storage, using the queue optimization. Its key is to store useful state only.

When the queue extent a new node, it is added at the end of the queue. The iteration is continued. This process required more time in order to move the new nodes. In order to relax the new node, an implementation of the algorithm can have an iterated.

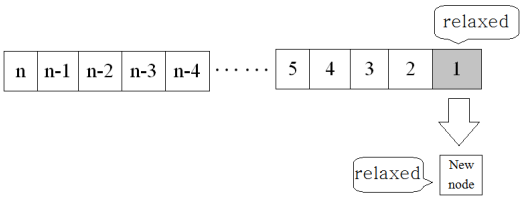

Fig. 2 Improving the search method of SPFA algorithm.

Fig. 2. shows improving the search method of SPFA algorithm. The method that begin at the first node. By the improved algorithm, the node 1 is first removed from the queue. After the node is relaxed successfully, and a new node will be searched directly. All nodes are relaxed, until the queue is empty.

Ⅳ. Results of Experiment

To verify our idea, experiments were taken on large amounts of data. First, the experiment of random maps. Dijkstra's algorithm solves the graphs with only non-negative edge weights. So in the case of negative edge weights, there is a comparison of the proposed algorithm with SPFA. In the case of non-negative edge weights, there is a comparison of the proposed algorithm with SPFA and Dijkstra's algorithms.

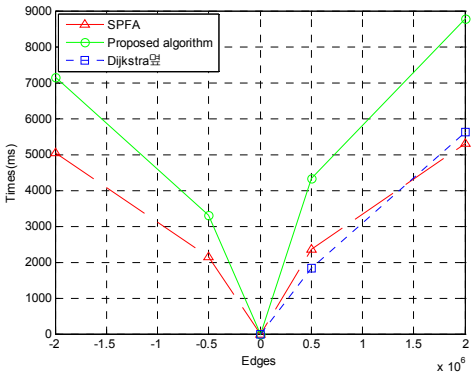

Fig. 3 Comparison of three algorithms (SPFA algorithm, the proposed algorithm and Dijkstra's algorithm) on random data.

As shown in Fig. 3, when the size of the graphs module is randomly generated, the proposed algorithm is not desirable, because it will keep iterated if a graph is great depth. It does too many times to iterate. SPFA and Dijkstra's algorithms have almost same efficiency.

Grid map is widely used because it is easy to build and maintain without definite geometric parameters.[6] Keeping iterated is characteristic of the proposed algorithm. Consequently, it is used on grid maps.

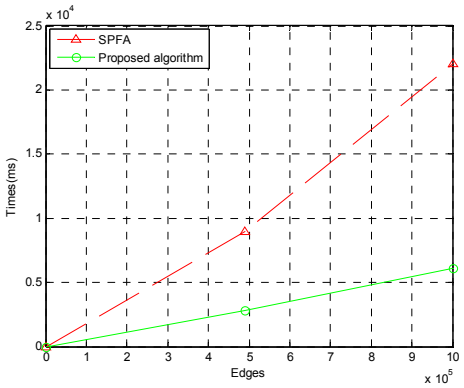

Fig. 4 Comparison of two algorithms (SPFA algorithm, the proposed algorithm) on grid maps.

In Fig. 4, we can see clearly the relationship between the proposed and SPFA algorithms. In both case, SPFA algorithm takes 31048 milliseconds, the proposed algorithm only takes 8898 milliseconds that reduced two-third in time consumption. We used Intel P7350 2.00 GHz, window 7(32 bits).

Ⅴ. Conclusions

The experimental results show that SPFA algorithm and Dijkstra′ algorithm have almost same efficiency on the random graphs(Fig. 3). But on grid maps the proposed algorithm is very efficient. As shown in Fig. 4, SPFA algorithm takes 31048 milliseconds, the proposed algorithm only takes 8898 milliseconds that reduced two-third in time consumption. So the proposed algorithm is more effective on grid maps.

We could see that on grid maps the proposed algorithm is very efficient, and the proposed algorithm than SPFA algorithm reduced two-third in time consumption by Matlab. This may broaden the development space for the proposed algorithm in the future.

References

[1] 서희종, "실시간 네트워크에서 개선된 분산 QoS 알고리듬, "한국전자통신학회논문지," 7권, 1호, pp. 53-60, 2012.
[2] William. Stallings, "Data and Computer Communications Eighth edition," Pearson Education International, pp. 332-347, 2007.
[3] George T. Heineman, "Algorithm in a Nutshell," O'Reilly Media, pp. 322, Oct. 2008.
[4] Duan. Fangding, "A Faster Algorithm for Shortest Path-SPFA," Journal of Southwest JIAOTONG University, Vol. 29, No. 6, pp. 207-212, Apr. 1994.
[5] Gao. Yang, "An Improved Shortest Route Algorithm in Vehicle Navigation System," International Conference on Advanced Computer Theory and Engineering (ICACTE), Vol. 2, pp. 363-366, Aug. 1996.
[6] Wang. Kun, "Simultaneous Localization and Map Building Based on Improved Particle Filter in Grid Map," International Conference on Electronic and Mechanical Engineering and Information Technology, Vol. 2, pp. 963-966, Aug. 2011.

저자 소개

진호(Hao Chen)

2010년 복경 석유화공학원 통신공학과 졸업(공학사)
현재 전남대학교 대학원 전자통신공학과 (석사과정)
※ 관심분야 : 라우팅, 코딩

서희종(Hee-Jong Suh)

1975년 한국항공대학교 항공통신공학과 졸업(공학사)
1996년 중앙대학교 대학원 전자공학과 졸업(공학박사)
1980년~2006년 여수대학교 전자통신공학과 교수
2006년~현재 전남대학교 전자통신공학과 교수
※ 관심분야 : 컴퓨터 네트워크, 인터넷통신, 위성통신