

On Linear time algorithm for SSSP Problem

Pradipta Prometheus Mitra
Ragib Hasan
M. Kaykobad

Department of Computer Science & Engineering,
 Bangladesh University of Engineering & Technology,
 Dhaka-1000, Bangladesh

email: shmitra@yahoo.com, ragibhasan@yahoo.com, mkbd1234@yahoo.com

Abstract

The shortest path problem is one of the most studied problems in computer science. In a previous paper[7], we suggested an algorithm(the Augmented Shortest Path Algorithm) for the solution of the single source shortest path problem in $O(mk+n)$ time, where k , the ratio of maximum and minimum weights of the edges of the graph, is bounded by a constant, m is the number of edges and n is the number of vertices. In this paper, we propose an improvement.

1. Introduction

The single source shortest path is one of the oldest classical problems in algorithm theory. Given a positively weighted directed graph G with a source vertex s , this problem asks for finding the shortest path from s to all other vertices. Almost all developments concerning this problem have evolved around the famous Dijkstra's algorithm [2, 3], presented first in 1959. The original version of this algorithm ran in $O(n^2+m)$ time[3]. The complexity has since been reduced to $O(m+n \log n)$ using Fibonacci heaps (Fredman and Tarjan, 1987)[4]. Various improvements have since been proposed [4, 5]. However, all these improvements have been limited to special classes of graphs. The current paper proposes to present a linear time algorithm for positively weighted directed graphs. While Thorup[6] developed a linear time algorithm for integral weights on undirected graphs only, our algorithm is valid for all real positive weights for directed graphs. Moreover, as [1] shows, the algorithm doesn't perform well in practical simulations. Our algorithm, as will be shown, is viable in theory as well as practice.

The assumptions of this algorithm are applicable to several real life problems such as:

- road networks
- Computer network i.e. LANs
- Internet
- Data communication networks
- Electronic Circuit Design

In these cases, no edge is unreasonably larger than the other edges, and therefore, the constant k is not very large. Also, in such graphs, the number of edges is not much greater than the number of vertices, so the space requirement is also low. So, the application of the proposed algorithm will produce the shortest path in linear time. This ensures the applicability of the present algorithm.

2. The previous algorithm: in a nutshell:

The intuition of the algorithm proposed in [7] is that the original graph can be transformed into a new augmented graph that is easier to handle. We know that a bfs like algorithm can find the shortest path of a graph that has equal weights in linear time. If we can replace every edge in the original graph by number of edges having equal weights and the number of new edges for each edge in the original graph is bounded, we can find the shortest path tree in linear time as well.

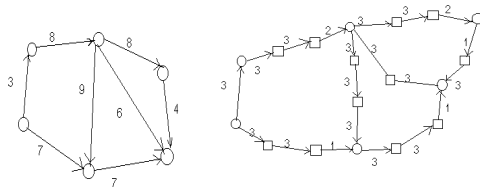


Figure 1: A graph G (left) and its augmented graph G_A (right)

3. The Problem:

The algorithm presented in [7] relied heavily on the constant k to achieve it's linear running time, which as referred before is the ratio of weights of the maximum and minimum weights in the graph. However, large values of k may cause the running time to deteriorate. We have shown in [7] that the edges that do not appear in the shortest path tree do not affect the running time even if they

have large weights. But an edge that appears in the shortest path tree will effect the running time. For example, lets use take the graph in Fig. 1.

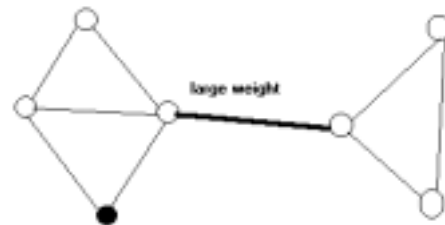


Fig. 2: Effect of large weights on the ASP Algorithm

In Fig.1 if we start computing the shortest path tree from the darkened node, the large edge (bold) will act as a severe bottleneck. We will progress by only w_{min} at each step, though the queue holds lengths much greater than w_{min} . In general, if the shortest path tree contains a few edges with massive weights, the ASP algorithm will fail to perform well.

4. The Motivation of the Algorithm

The main observation in this respect was that if edges with huge weights exist, they will eventually drive out other edges from the queue. In such a situation the queue will consist of entries much larger than the minimum weight w_{min} . It will be wasteful to advance the distance order search only by amount w_{min} .

The new algorithm proposes that the distance order search will advance by the maximum of w_{min} and the minimum weight in the queue(up to the current distance order search). Thus we eliminate the problem involved with edges with large weights.

5. The Algorithm

We describe the algorithm in the following.

New Augmented Shortest Path (G, s)

1. Find the minimum edge weight w_{min} in $O(m)$ time from the adjacency list.
2. for all $(u,v) \in E$ do
3. $inqueue[u,v] \leftarrow false$
4. $edge[u,v] \leftarrow \infty$
5. enddo
6. $d[s] \leftarrow 0$
7. $nowmin = \infty$
8. $nextmin = \infty$
9. $nowcount = 0$
10. for all $(s,u) \in E$ do
11. $enqueue(u,s)$

```

12.   inqueue[s,u] ← true
13.   edge[s,u] ← w[s,u]
14.   if(w[s,u]<nowmin)
15.       nowmin = w[s,u]
16.   nowcount = nowcount+1
17.   enddo
18.   nextcount = 0
19.   while queue ≠ empty do
20.       v,p ← serve()
21.       wv ← edge[p,v]-max(wmin, nowmin)
22.       if wv ≤ 0 then
23.           if d[p]+w[p,v] < d[v] then
24.               d[v] ← d[p]+w[p,v]
25.               π[v] ← p
26.               for all (v,u) ∈ E do
27.                   if inqueue[v,u] = false then
28.                       if d[v]+w[v,u] < d[u] then
29.                           enqueue(u,v)
30.                           inqueue[v,u] ← true
31.                           edge[v,u] ← w[v,u]+wv
32.                           nextcount = nextcount + 1
33.                           if(edge[v,u] < nextmin)
34.                               nextmin = edge[v,u]
35.                       endif
36.                   endif
37.               else
38.                   edge[v,u] ← min(edge[v,u],w[v,u]
39.                       + wv)
40.                   if(edge[v, u] < nextmin)
41.                       nextmin = edge[v,u]
42.                   endif
43.               endif
44.           enddo
45.       endif
46.   else
47.       if d[p]+w[p,v] < d[v] then
48.           enqueue(v,p)
49.           edge[p,v] ← wv
50.           if(wv < nowmin)
51.               nextmin = wv
52.           endif
53.       endif
54.   endif
55.   enddo
56.   nowcount = nowcount - 1
57.   if(nowcount = 0)
58.       nowcount = nextcount
59.       nowmin = nextmin
60.       nextmin = ∞
61.       nextcount = 0
62.   enddo
63.   Return the shortest path.

```

6. Analysis

Time complexity

The performance of the initial algorithm, was $O(mk/2 + n)$ time which is linear if k is not unreasonably large [7]. However, with large values of k , the algorithm can become prohibitively slow. The new proposed improvement, though doesn't introduce any order improvements, bounds the maximum possible number of distance order searches (which is n). This ensures that the algorithm will run in reasonable time even if k is large.

Space Complexity

The graph is represented using the adjacency list implementation. For a graph with n vertices and m edges, the space requirement is $O(m+n)$.

In the worst case, the space requirement for the Queue is equal to the maximum number of edges in the augmented graph, which is $O(m)$.

7. An Analysis on the behavior of the distance order queue:

Definition: A queue entry is residual if the weight of the entry $< w_{min}$.

Let, the graph contains n nodes, m edges and $k = w_{max}/w_{min}$.

On the average, the weights of the edges will be distributed as follows:

| Group | Characteristic | No.of edges | No. of times in the queue |
|-------|--|----------------|---------------------------|
| 1 | $w_{min} < w \leq 2 \times w_{min}$ | $m' = m/(k-1)$ | 2 |
| 2 | $2 \times w_{min} < w \leq 3 \times w_{min}$ | $m' = m/(k-1)$ | 3 |
| | | | |
| k-1 | $(k-1) \times w_{min} < w \leq k \times w_{min}$ | $m' = m/(k-1)$ | k |

The edges in the queue, except possibly for the time, have weights $\geq w_{min}$. So, the number of queue entries in each group that have weights $\geq w_{min}$ and $< w_{min}$ can be counted respectively as:

| Group | Characteristic | Entries with weights $\geq w_{min}$ | Entries with weights $< w_{min}$ |
|-------|--|-------------------------------------|----------------------------------|
| 1 | $w_{min} < w \leq 2 \times w_{min}$ | m' | m' |
| 2 | $2 \times w_{min} < w \leq 3 \times w_{min}$ | $2 \times m'$ | m' |
| | | | |
| k-1 | $(k-1) \times w_{min} < w \leq k \times w_{min}$ | $(k-1) \times m'$ | m' |
| Total | | $m' \times k(k-1)/2$ | $m' \times (k-1)$ |

Hence, probability the a queue entry is residual is $p = 2/k$.

Now, let us consider the shortest path tree. The depth of the tree may vary from 1 to $n-1$. We take the average $n/2$ to simplify analysis.

The average weight of an edge is $w_{min} \times (k+1)/2$

Clearly the maximum distance order search will be $n \times (k+1)/2$.

The total number of queue entries $m' \times k(k+1)/2 \approx m \times (k+1)/2$

\therefore , number of entries in the one distance order search = $2m/n$.

Assuming binomial distribution,

Average no. of residual edges in a distance order search = $2/k \times 2m/n = 4m/nk$.

Probability that there are no residual edges in a distance order search (and thus by choosing w_{min} we are losing) becomes (after some manipulation): $(1-2/k)^{2mn}$

Comments:

1. This analysis gives the approximate probability that we lose in the original algorithm by choosing w_{min} as the next amount to move. However, this analysis does not predict the behavior of the new algorithm.
2. The analysis is performed by taking average parameters in many cases. It may not reflect the behavior of the algorithm in many representative cases.

8. Performance Test

We compared the three algorithms for various values of k . The implementation was in the C++ programming language. The compiler used was Borland C++ Builder 3.0 for Windows 98 running on a Pentium III 500Mhz machine with 64MBs of main memory.

In the following table, the times for 30 runs for each algorithm are presented.

| N o. | Ratio w_{max}/w_{min} K | No. of Vertice s N | No. of Edges M | Time for Dijkstra T _D (sec) | Time for ASP T _{ASP} (sec) | Time for NewASP T _{NewASP} (sec) |
|------|---------------------------|--------------------|----------------|--|-------------------------------------|---|
| 1 | 800 | 200 | 39000 | 0.156 | 0.134 | 0.121 |
| 2 | 100 | 2000 | 5000 | 6.89 | 5.745 | 5.355 |
| 3 | 40 | 2000 | 30000 | 8.50 | 5.46 | 5.32 |
| 4 | 40 | 2000 | 40000 | 8.68 | 5.46 | 5.34 |
| 5 | 140 | 2000 | 30000 | 8.17 | 5.57 | 5.46 |
| 6 | 140 | 2000 | 40000 | 8.59 | 5.61 | 5.47 |
| 7 | 240 | 2000 | 30000 | 8.22 | 5.69 | 5.57 |
| 8 | 240 | 2000 | 40000 | 8.57 | 5.70 | 5.67 |
| 9 | 340 | 2000 | 30000 | 8.24 | 5.79 | 5.70 |
| 10 | 340 | 2000 | 40000 | 9.28 | 5.78 | 5.70 |
| 11 | 540 | 2000 | 30000 | 8.26 | 5.99 | 5.89 |
| 12 | 540 | 2000 | 40000 | 8.70 | 5.95 | 5.86 |

We also simulated the behavior of the two ASP algorithms to find out the percentage of the distance order search that used the minimum value in the queue instead of the global minimum.

The following table represents these results.

| No. | Ratio w_{max}/w_{min} K | No. of Vertices N | No. of Edges M | Ratio R1 (%) | Ratio R2 (%) |
|-----|---------------------------|-------------------|----------------|--------------|--------------|
| 1 | 800 | 200 | 39000 | 22.0 | 30.0 |
| 2 | 100 | 2000 | 5000 | 0.22 | 10.0 |
| 3 | 40 | 2000 | 30000 | 1.60 | 37.5 |
| 4 | 40 | 2000 | 40000 | 4.90 | 60.0 |
| 5 | 140 | 2000 | 30000 | 0.40 | 15.0 |
| 6 | 140 | 2000 | 40000 | 0.60 | 26.7 |
| 7 | 240 | 2000 | 30000 | 0.63 | 16.7 |
| 8 | 240 | 2000 | 40000 | 1.60 | 26.7 |
| 9 | 340 | 2000 | 30000 | 0.38 | 14.0 |
| 10 | 340 | 2000 | 40000 | 1.40 | 18.2 |
| 11 | 540 | 2000 | 30000 | 0.40 | 15.2 |
| 12 | 540 | 2000 | 40000 | 0.35 | 8.0 |

Note:

R1: The ratio of the number of queue entries that used the queue minimum and number of entries that used the global minimum.

R2: The ratio of the number of distance order searches that used the queue minimum and number of distance order searches that used the global minimum.

Comments

1. From the experimental values, it can be observed that the Augmented_Shortest_Path algorithm outperforms Dijkstra's algorithm, under the conditions specified above.
2. The experimental cases presented are for random graphs. If graphs with edges with extremely large weights were tested, even better performance improvements may have been observed.
3. As an aside, researchers have shown that bucket-based algorithms often perform poorly in practical cases[1].

9. Conclusion

A few remarks may be made on possible future work.

1. The analysis presented here can be performed in a more generic setup. The analysis does not agree very closely to the presented experimental data. Both theoretical study and more elaborate performance tests may be performed to develop a more thorough understanding of the algorithm.
2. The behavior of the new algorithm also requires analysis. Experimental results seem to show that the distance orders

containing residual edges are much larger in size than the one that don't. Researchers can try to find out the reason for such behavior.

Reference

1. **Asano, Y., Imai H.:** *Practical Efficiency of the Linear Time Algorithm for the Single Source Shortest Path Problem* : The journal of the operations research society of Japan.
2. **Cormen, T.H., Leiserson, C.E. and Rivest, R.L.:** *Introduction to Algorithms* : MIT Press, 1990.
3. **Dijkstra, E. W.:** *A note on two problems in connection with graphs.* Numer. Math. 1, 269-271. 1959.
4. **Fredman, M.L. and Tarjan, R.E.:** *Fibonacci heaps and their uses in improved network optimization algorithms.* J.ACM 34, 3(July), 596-615. 1987.
5. **Moriya, E and Tsugane, K.:** *Optimally Fast Shortest Path Algorithms for Some Classes of Graphs.* Inter. J. Computer Math. Vol. 70, pp.297-317 1998.
6. **Thorup, Mikkel:** *Undirected Single Source Shortest Paths with Positive Integer Weights in Linear Time.* J.ACM 46, 3(May), 362-394, 1999.
7. **Mitra, P.P, Hasan, R and Kaykobad, M:** *A Linear time algorithm for single source shortest path problem.* ICCIT '2000.