Comparative Analysis of Algorithms for Single Source Shortest Path Problem

Mrs. Shweta Srivastava *shwetasrivastava21@gmail.com*
Computer Science & Engineering Department,
ABES Engineering College Ghaziabad.
India

Abstract

The single source shortest path problem is one of the most studied problem in algorithmic graph theory. Single Source Shortest Path is the problem in which we have to find shortest paths from a source vertex v to all other vertices in the graph. A number of algorithms have been proposed for this problem. Most of the algorithms for this problem have evolved around the Dijkstra's algorithm. In this paper, we are going to do comparative analysis of some of the algorithms to solve this problem.

The algorithms discussed in this paper are- Thorup's algorithm, augmented shortest path, adjacent node algorithm, a heuristic genetic algorithm, an improved faster version of the Dijkstra's algorithm and a graph partitioning based algorithm.

Keywords: Single Source Shortest Path Problem, Dijkstra, Thorup, Heuristic Genetic Algorithm, Adjacent Node Algorithm.

1. INTRODUCTION

The single source shortest path problem can be defined as: given a weighted graph (that is, a set *V* of vertices, a set *E* of edges, and a real-valued weight function *f*: $E \rightarrow \mathbf{R}$), and one element s of *V* (i.e. a distinguished source vertex), we have to find a path *P* from s to a *v* of *V* so that

$$\sum_{p \in P} f(p)$$

is minimal among all paths connecting *s* to *v* .
Refer figure 1.

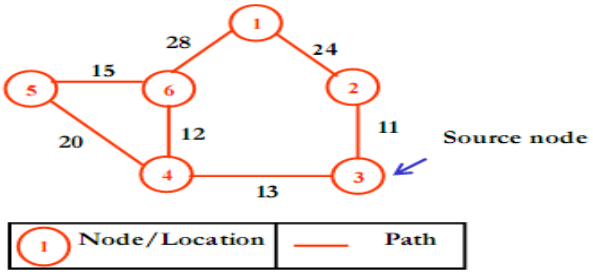

FIGURE 1: An Undirected Graph of 6 nodes and 7 edges

SSSP is applied in various areas such as:
1. Road Network
2. Computer Network
3. Web Mapping

4. Electronic Circuit Design
5. Geographical Information System (GIS)

In all six papers [1][2][3][4][5][6], the authors have given an improvement over the Dijkstra's algorithm.

2. BACKGROUND STUDY AND ANALYSIS

The Dijkstra's algorithm makes assumption that there is no negative-weight edges in the graph G (V, E) : w(u, v) >0 , $\forall$ (u, v) $\in$ E. The algorithms in [1][2][3][4][5][6] also follows this assumption. For simplicity, we use n = |V| and m = |E| and K distinct edge lengths.

Several algorithms have been proposed for this problem which is based on different design paradigms, improved data structure, parameterization and input restrictions. According to survey it is found that no algorithm based on the Dijkstra's algorithm has achieved the linear time complexity due to drawbacks of Dijkstra's algorithm. Dijkstra's algorithm maintains an adjacency matrix which consumes n*n space in the memory. When the number of nodes (n) is very large, it is difficult to apply Dijkstra's algorithm.

So, in paper [1], an algorithm has been proposed which modify Dijkstra's algorithm to overcome its bottlenecks. Dijkstra's algorithm visits the vertices corresponding to a sorting algorithm (in order of increasing d (v)). Since there is no linear ime algorithm for sorting problem. Unless the order of visiting vertices is not modified, a linear time complexity cannot be achieved. The performance of an algorithm for single source shortest path problem depends on the 3 attributes:

(1) Preprocessing time: time required to construct a search structure suitable for search.
(2) Space: storage used for constructing and representing the search structure.
(3) Search Time: time required to find shortest path from a query source s, using the search structure.

In year 2000 **Thorup** proposed a concept of components and using some complicated data structures which overcome the problem in Dijkstra's algorithm. There are 3 interesting features of the Thorup's algorithm [1]:
(i)It contains a minimum spanning tree algorithm as its sub procedure. To achieve the linear time complexity Thorup used a linear time MST algorithm.
(ii) Thorup's algorithm consists of 2 phases: a construction phase which constructs a data structure suitable for a shortest path search from the given query source s; a search phase of finding the shortest paths from s to all vertices using the data structure constructed in construction phase.
(iii) Construction phase in Thorup's algorithm is independent of the source, while data structures in previous algorithms heavily depend on the source.
Summary of the Thorup's Algorithm:
Step1. Construct an MST (M).
Step2. Construct a component tree (T) using MST.
Step3: Compute widths of buckets (B) to maintain the components.
Step4: Construct an interval tree (U) to store unvisited children.
Step5: Visit all components in T by using B and U. Also known as search phase.
The running time of each step is as follows: step1: O (m) time, step2, 4: O (n) time, step5: O (m+n) time.

In year 2000, a linear time algorithm for SSSP problem [4] was proposed called an improvement over

Augmented Shortest Path Algorithm.
They proposed this algorithm for situations where no edge is unreasonably larger than the other edge and the ratio of maximum and minimum weights of the edges (f) of the graph is not very large.

The algorithm converts the graph G into an augmented graph (Ga) in we replace every edge in the original graph by number of edges having equal weights and number of new edges for each edge in the graph is bounded. Then the shortest path tree is obtained. The major drawback in the ASP algorithm was that if some heavy weight edge is included in the shortest path tree ASP fails to perform well so an improvement is done in improved ASP algorithm [4].

```
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) ε E do
3.  inqueue [u,v] <- false
4.  edge [u,v] <- ∞
5. end do
6. d[s]<-0
7. nowmin=∞
8. nextmin=∞
9.  nowcount = 0
10. for all (s,u) ε  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. end do
18. nextcount = 0
19. while queue != empty do
20.   v,p <- serve()
21.   wv<- edge[p,v]-max(wmin , nowmin)
22.   if w <= 0 then
23.      if d[p]+w[p,v] <d[v] then
24.      d[v] <- d[p]+w[p,v]
25.      ⌈][v] <- p
26.      for all (v,u) ε 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(egde[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.   endif
47.   else
48.    if d[p]+w[p,v] <d[v] then
49.      enqueue(v,p)
50.      edge[p,v] <-wv
51.      if(wv   < nowmin)
52.         nextmin = wv
53.      endif
54.   endif
55. endif
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.

This new algorithm proposes that the distance order search will advance by the maximum of w(min) and the minimum weight in the queue. So, the problem of larger weight edge got removed in this new version. The new augmented shortest path algorithm takes O(mf/2 + n) time which is linear if f is not large. The space requirement is O(m+n). This algorithm proved to perform better than the bucket based algorithm.

In year 2006, an approach was proposed by the authors of [6] to speed up the Dijkstra's algorithm. An acceleration method called arc-flag is used to improve Dijkstra's algorithm. In this approach we follow a preprocessing of the graph to generate some additional information about the graph which is then used to speed up shortest path queries. In the preprocessing step graph is divided in the regions and checked whether an arc belongs to the shortest path in the given region. This preprocessing method is combined with an appropriate partitioning technique and bi-directed search which achieves an average speed up factor of more than 500 compared to the Dijkstra's algorithm on large networks. They tested different combinations of the arc-flag method with different partitioning technique.

They used A*, bi-directed search techniques and chosen bi- directed search because A* didn't improve the speed up factor. They considered Grid, kd, Tree or METIS as the base partitioning method and made 11 combinations of the searching, partitioning and preprocessing techniques. They applied the 11 different combinations on the German road network data. Kd trees and METIS yields the best speed- up. Bi-directed search proved to be better than the unidirected search and the two level partitioning was better than the single level partitioning. The preprocessing takes O (m(m+n+nlogn)) time. It increases for the dense graph.

In year 2007, **a heuristic genetic algorithm** [3] was proposed to achieve high performance. Their proposal starts with the initial population of candidate solution paths than a randomly generated one. HGA also uses a new heuristic order crossover (HOC) and mutation (HSM) to keep the limited search domain.
The components required to develop HGA requires chromosome coding, initialization, genetic crossover operator, genetic mutation operator, and parent selection & termination rules.

Summary of HGA:
Step1: Chromosome Coding Scheme and Initialization: Chromosome Coding Scheme:
The complete chromosome of a candidate is divided into node fields equal to the number of nodes in a network. Refer Figure 2.

3	3	0	3	2	11

FIGURE 2: Part of Chromosomal structure of a candidate path

This structure uses the node indices and the distance weight between 2 nodes.
N i0 = Previous (Ni) N i1 = Ni
N i2 = dist (Ni)
Previous (Ni) is same as the predecessor array and dist (Ni) is same as an array of best estimates of shortest path to each vertex in the Dijkstra's algorithm.

Initialization: First node of every candidate path is the source node. So each chromosome (s,s,0). Other entries are random nodes, covers all other nodes in the graph.
Step2: Parent Selection, HOC, HSM and Termination: Parent Selection:
Here, algorithm chosen for the selection is Tournament Selection Algorithm. The idea behind this is to pick a pair at random, compare their fitness and the fittest is selected.
To find the fitness value we need to know the objective function (path cost). Path cost = sum of (dist (Ni)) where i = 1 to n.
Using this the fitness function value is calculated as: Fitness (Chromosome) = $\frac{1}{\text{Path Cost}}$

HOC:
Here node fields are chosen as the cut points. The portion of the first parent between them is copied to the offspring, the rest of the offspring is selected from the second parent with the following conditions:
1. The source node fields remains at the first position in new generated offspring.
2. While taking other nodes from the second parent, all nodes should be included only once in the offspring.
And the offsprings are evaluated and added as the new solution path candidate.
HSM:
Two node fields are chosen randomly and swapped given that the source node is never mutated. And again new mutated chromosome is evaluated. HOC and HSM don't generate new edges in any candidate path , they just adjust the initially generated nodes into a legal minimum cost path
Termination:
The algorithm can be terminated when the number of generation crosses an upper bound specified by the algorithm. With an increased number of generations, HGA converges to the optimal solution.

In year 2009, an algorithm based on Dijkstra for huge data [5] was proposed. In the paper author has pointed out the drawbacks of the Dijkstra's algorithm and proposed an algorithm as **adjacent node algorithm** which an optimization over Dijkstra's algorithm. He proved that his algorithm can save lot of memory and is more suitable for graph with huge nodes. The adjacent node algorithm makes improvement by improving the method of creating the adjacency matrix. First the number of the maximum adjacent nodes r is found. Then the adjacency matrix of n*r is made which is much smaller that n*n matrix. One more judgement matrix is made of order m*r. The shortest path is found with the help of both adjacency and judgement matrix. In their experiment, this algorithm performed 6 times better than the Dijkstra's algorithm for the data size of 12000 nodes.

In year 2009, a faster algorithm [2] has been proposed for SSSP problem. They have proposed an efficient method for implementing the Dijkstra's algorithm with the same assumptions. In addition to it two more assumption is made that : Let L= {l1,l2,.....lk} be the set of distinct nonnegative edge weights given in an increasing order as part of the input stored as an array and the number of distinct edge lengths (k) is small. The author's solution is motivated by the "gossip" problem for social networks.

Two algorithms are proposed by the author in this paper:

1. Simple implementation of Dijkstra's algorithm that runs in O (m+nk) time.
2. Second algorithm is the modification of first algorithm by using binary heaps to speed up the FindMin() operation. Its running time is O(m log (nK/m)) if nK>2m.

Both the algorithms are identical to Dijkstra's algorithm. The difference is that it uses some additional data structures to carry out FindMin() operation. The algorithm is as follows:
Step1:
Function INITIALIZE()
1: S:={s}; T := V-{s}.
2: d(s):=0; pred(s):=.Φ
3: for (each vertex vε T) do
4: d(v)= ∞; pred(v)=. Φ

```
5: end for
6: for (t=1 to K) do
7:          Et(S):=. Φ.
8:          CurrentEdge(t):=NIL.
9: end for
10: for each edge(s, j) do
11:         Add(s, j) to the end of the list Et (S),where It =csj .
12:          if (CurrentEdge(t)=NIL) then
13:                   CurrentEdge(t):=(s, j)
14: end if
15: end for
16: for (t=1to K) do
17:         UPDATE(t)
18: end for
```

Step2:

```
Function NEW-DIJKSTRA()
1:  INITIALIZE ()
2:  while (T= Φ) do
3:          let r = argmin {f(t):1t K}.
4:          let (i, j) = CurrentEdge(r).
5:          d(j):=d(i) + lr ; pred(j):=i.
6:          S= S Ụ{j}; T :=T - {j}.
7:          for (each edge (j,k) ε E(j)) do
 8. Add the edge (j,k) to the end of the list Et(S),where It =cjk .
 9:            if (CurrentEdge (t) = NIL) then
10:                  CurrentEdge (t): = (j,k)
11:           end if
12:         end for
13:         for (t = 1to K) do
14:                UPDATE(t).
15:       end for
16: end while
```

Step3:

```
Function UPDATE(t)
1:Let (i, j) = CurrentEdge(t).
2: if (jεT) then
3:          f(t)=d(i)+cij
4:           return
5: end if
6: while ((j ε T) and (CurrentEdge(t).next != NIL)) do
7:          Let(i, j) = CurrentEdge(t).next.
8:          CurrentEdge(t)=(i, j).
9: end while
10: if (jT) then
11:         f(t)=d(i)+cij .
12: else
13:         Set CurrentEdge(t) to Φ.
14:         f(t)= ∞ .
15: end if
```

The initialization step takes O (n) time. The potential time taking operations are step 3 of New-Dijkstra and the Update procedure. In New-Dijkstra step3 takes O (k) per iteration of the while loop and O (nk) over all the iterations. Procedure Update is called O (nk) times and its total running time is O (m+nk). Iteration in which CurrentEdge (t) is not changed, running time is O (nk) and the iterations in which CurrentEdge(t) is changed, the running time is O(m).
So the total time taken by the algorithm is O (m+nk).

3. USEFULNESS OF ALGORITHMS IN VARIOUS APPLICATIONS

Thorup's algorithm [1] is much slower than the algorithms with the heaps as for the whole execution time is compared. It is very slow for SSSP due to the time of the construction of the data structures. Due to need of huge amount of memory and the complicated, large programs, Thorup's algorithm is not useful in practice today.

The algorithm in paper [2] works well for the graphs having smaller number of distinct edge lengths than the density of the graph.

The Heuristic Genetic algorithm [3] proved to be suitable for the network of different size and topology. HGA took reasonable CPU time to reach the exact solution and didn't variate much with increased input size.

The Augmented Shortest path algorithm [4] is suitable for the graphs with less value of f. According to author it is suitable for the road networks, electronic circuit designs etc.

The Adjacent Node algorithm in [5] is efficient for the huge data and takes less space. So it is suitable for the traffic analysis type of applications.

The algorithm based on partitioning of graph [6] although performed better than Dijkstra's algorithm for some cases but for large networks its performance is degraded than that of Dijkstra's.

4. CONCLUSIONS

In this paper it is tried to be explained- first, what are the different algorithms for the SSSP problem. Second, how do they perform in comparison of the Dijkstra's algorithm. Third, which algorithm is suitable for a particular application or situation.

5. REFERENCES

[1] Y. Asano, H. Imai, " Practical Efficiency of the Linear Time Algorithm for the Single Source Shortest Path problem", *Journal of the Operations Research, Society of Japan, Vol. 43, No. 4; 2000.*

[2] J. B. Orlin, K. Madduri, K. Subramani, M. Williamson, "A faster algorithm for the Single Source Shortest Path problem with few distinct positive lengths"*, Journal of Discrete Algorithm; 2009.*

[3] B. S. Hasan, M. A. Khamees, A. S. H. Mahmoud, "A Heuristic Genetic Algorithm for the Single Source Shortest Path Problem", *IEEE International Conference on Computer Systems and Applications; 2007.*

[4] P. P. Mitra, R. Hasan M. Kaykobad, "On Linear time algoritm for SSSP Problem", *ICCIT, 2000.*

[5] Zhang Fuhao, L. Jiping, "An algorithm of shortest path based on Dijkstra for huge data", *6th International Conference on Fuzzy Systems and Knowledge discovery, 2009.*

[6] R. H. Mohring and H. Schilling, "Partitioning Graphs to Speedup Dijkstra's Algorithm", *ACM Journal of Experimental Algorithmics, Vol. 11, Article No. 2.8, Pages 1-29, 2006.*