International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-10, August 2019

A Backward Single Source Shortest Path Algorithm For Directed Graph

D. Jasmine Priskilla, K. Arulanandam

Abstract*: *A new directed backward variant of the Single Source Shortest Path algorithm was described in this paper. This algorithm accept that approaching adjacency list of the given graph vertex loads showed up in expanding request. The running time of forward, based strategy algorithm is the best aftereffect of O (n), which are the most ideal forward-backward SSSP consequences of Wilson et al. Moreover, the likelihood of the new algorithm additionally requires O (n) time.This is an equally improved version of exponentially and polynomial small probability derived by Wilson et al.

Keywords - shortest path, backward algorithm, directed graph, directed backward algorithm.

I. INTRODUCTION

The Single Source Shortest Path Algorithm solved by Bellman-Ford O (mn) time with the suspicion of all the edge weights have positive values. Dijkstra's [1] takes the time to solve this algorithm in O (m+nlogn). Thorup [2] found an algorithm with the runtime of O (m+n).

Thorup's algorithm and Dijkstra's algorithm are linear runtime and linear in the input-graph size. Spira's algorithm solves Single Source Shortest Problem by O ($n\log^2 n$) time. Later, this result was improved by many researchers like Takoka and Moffat [3] to O($n^2 \log n \log\log n$). And Bloniarz [4] improved to O(n^2 logn log*n). The directed backward SSSP algorithm is developed by us has runtime O (n) and it is implemented in two steps.

II. FORMULATION OF THE ALGORITHM

A. A New Directed Backward algorithm for Single Source Shortest Path

This algorithm produces the shortest way from z to y i.e, from destination vertex z to the source vertex y in a directed graph G = (V,E) with non-negative weight. Start by dist [z] = 0 while dist [y] = ∞ for every y ≠ z. At the point when Spira's algorithm [5] endeavors to enhance Dijkstra's algorithm, it considered the out-going edges of every vertex x in expanding request of weight. In any case, when we improve the Spira's algorithm with a backward algorithm idea we set the in-coming edges of every vertex z in the diagram are given is non-diminishing order of weight. The out-going edges of x are checked individually in Spira's algorithm. The algorithm discovers all vertices whose good ways from y is littler than dist[x] + cost [x,z^1] by sweeps out- going edge (x,z), where (x,z^1) is edge going before (x,z) in the adjacency list of x. This algorithm maintains the priority queue PQ to hold edges rather than vertices. The distance dist[x] from y to x is calculated based on the key value of edges (x,z) in PQ, and PQ= dist[x]+cost[x,z]. This directed backward algorithm examines the in-coming edges of z individually. The algorithm filters an in-coming edge (x,z) simply after it discovers all vertices whose distance to z is littler than dist[z]+cost[x,z], where (x^1,z) is the edge going before (x,z) in the adjacency list of z. To acquire this outcome, the priority queue PQ hold edge instead of vertices. The edge (x,z) is a key in PQ is dist[z]+cost[x,z], If it is accessible in PQ then the dist[z] is now set to distance from x to z, and the edge (x,z) is a key in PQ is dist[z]+cost[x,z] .

This algorithm keeps up a set Y⊆Z, where Z contains all vertices whose good ways from Z was at that point decided. At first Y={Z}. On the off chance that yϵY, the dist[y] is the good ways from z to y. On the off chance that y≠Y, at that point dist[y]= ∞. On the off chance that yϵY \{z}, at that point (path[y],y) is the last edge of a way of length dist[y] from z to y. At first set dist[z]=0, while dist[y]= ∞ for each y ≠z, and path[y]=null for each yϵZ.

This new algorithm begins by checking the in-coming edges (y,z) of z and embeddings it into the priority queue PQ with the key worth dist[z] +cost[y,z]= cost[y,z] from PQ in every cycle of the algorithm extricates an edge (x,z) with the littlest key dist[z]+cost[x,z], in the event that (x,z) isn't the toward the end in-coming edge of z, at that point the edge (x^1,z) that tails it in the adjacency list of z is embedded into PQ with key dist[z]+cost[x^1,z]. On the off chance that xϵY, at that point dist[z]+cost[x,z], the key of (x,z) is the distance to x, Thus, dist[x] is set to dist[z]+cost[x,z], PQ[x] is set to z, x is added to Y, and the first in-coming edge (w,x) of x, on the off chance that it is accessible in examined and embedded into PQ. This new algorithm additionally embeds an edge (x,z) into PQ regardless of whether xϵY or if PQ as of now contains an edge (x,z^1) with dist[z^1]+cost[x,z^1]<dist[z]+cost[x,z], when (x,z) is separated from PQ, the algorithm realizes that the time has come to filter the following in-coming edge of Z. The pseudo-code of the algorithm is given in Fig.1. Every vertex of zϵZ has a adjacency list In[z] of its in-coming edges arranged in rising request of weight. The perspective on In[z] is as a rundown of edges, every datum in In[z] is a vertex, the opposite end purpose of the edge that comes towards Z.

Manuscript published on 30 August 2019.
*****Correspondence Author(s)
D. Jasmine Priskilla, Ph.D Research Scholar, Research and Development Centre, Bharathiar University, Coimbatore, Tamilnadu, India.
Dr. K. Arulanandam, Head, Department of Computer Applications, GTM College, Gudiyattam, Tamilnadu, India.

The usage of this algorithm utilizes a function backward(z) that gets the following in-coming edge of the z and adds it, in the event that it exits, into PQ with key the dist[z]+cost[z,x]. The following in-coming edge is determined by calling before(In[z]). The function before(In[z]) restores the edge right now indicated and goes in backward the pointer to the past edge in the list, generally on the off chance that the list is unfilled, at that point before(In[z]) return invalid. Reset(In[z]) makes this pointer point to the primary edge of the list.

B. Proposed Algorithm for Directed Backward Single Source Shortest Path

Algorithm Shortest (G = (V,In,cost),z)

```
Y ← {z}; PQ ← ϕ
for all xϵZ do
{
dist [x] ← ∞
PQ[x] ← null
reset (In[x])
}
dist [z] ← 0
backward (z)
while Y ≠ Z and PQ ≠ ϕ do
{
(z,x) ← mindistance (path)
backward (z)
if (xϵY) Then
{ dist [x] ← dist[z] + cost[x,z]
PQ [x] ← z
Y ← Y U {x}
backward (x)
}
}
function backward(z)
{
x ← before (In[z])
if x ≠ null then
insert (PQ,(z,x), dist[z]+cost[x,z])
}
```

Fig.1 Shortest path algorithm.

C. Analysis of the Algorithm

Spira's algorithm demonstrates that the Single Source Shortest Problem utilizing the forward technique on κ_n (EXP(1)) can be tackled in O(n) time, But in the directed backward Single Source Shortest Problem algorithm the in-coming adjacency lists were utilized. We build up the new O(n) time directed backward Single Source Shortest Problem algorithm in two stages. In the initial step the algorithm looks at O (n) edge of the graph. In the subsequent advance, the algorithm is actualized to keep running in O (n) time. As the algorithm of Single Source Shortest Problem this algorithm additionally utilizes priority queue. During execution it takes O (n) time, and it needs to perform priority queue tasks expected in O (1) time. The distance of the primary (n/2) vertices are determined during the principal phase of the procedure. The relevance states of the algorithm is expressly checks in the second phase of the procedure. At the point when non in-appropriate edges are achieved, this algorithm quits checking of the in-coming adjacency lists.

Example 1.

The following Fig.2 shows the directed weight graph taken forward to prove the algorithm. Now, we have traced the shortest path vale from node S to node Z. This algorithm uses backward search for the shortest path. So the search begins from node Z and the In-coming node of Z is analyzed up to the initial value node S found.

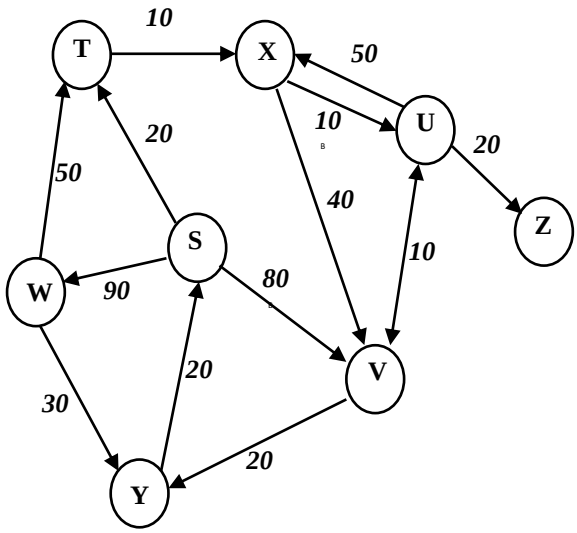

Fig.2 Directed weight graph G1.

After Initialization *dist* values changed for each process as in Table 1, and when node S selected as a new node to process, the searching operation is completed. From this example, initially the process starts from node Z and completed when it reaches the node S. The shortest value for S->Z is 60.

Table 1. Distance values between source and destination.

	S	T	U	V	W	X	Y	Z
1	∞	∞	20	∞	∞	∞	∞	∞
2	∞	∞	20	30	∞	30	∞	∞
3	110	∞	20	30	∞	30	∞	∞
4	110	40	20	30	∞	30	∞	∞
5	60	40	20	30	90	30	∞	∞

The path values are created as follows from the values in Table 1. Initially it starts from node Z and the In-coming path is from node U is shown in Table 2 as U->Z. From Table 2 we found that the path between node S-> Z is [S -> T -> X -> U ->Z] utilizing backward search method.

Table 2. Path between selected nodes

	S	T	U	V	W	X	Y	Z
Stage1	-	-	[U-Z]	-	-	-	-	-
Stage 2	-	-	[U-Z]	[V-U-Z]	-	[X-U-Z]	-	-
Stage 3	[S-V-U-Z]	-	[U-Z]	[V-U-Z]	-	[X-U-Z]	-	-
Stage 4	[S-V-U-Z]	[T-X-U-Z]	[U-Z]	[V-U-Z]	-	[X-U-Z]	-	-
Stage 5	[S-T-X-U-Z]	[T-X-U-Z]]	[U-Z]	[V-U-Z]	[W-T-X-U-Z]	[X-U-Z]	-	-

Example 2.

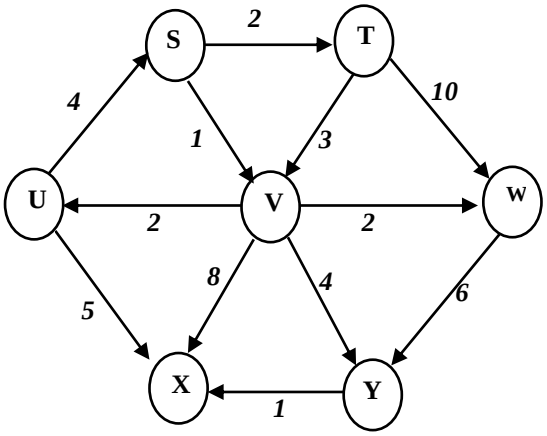

Fig.3 Directed weight graph G2

	Shortest Path Value		No. of comparison for selection of shortest path	
	Forward Method	Backward Method	Forward Method	Backward Method
V → S	6	6	2	2
V → T	8	8	6	3
V → U	2	2	1	1
V → W	2	2	2	1
V → X	5	5	4	3
V → Y	4	4	3	1

Table 3. The shortest distance values and number of comparisons between different pairs of the edges in Fig.3

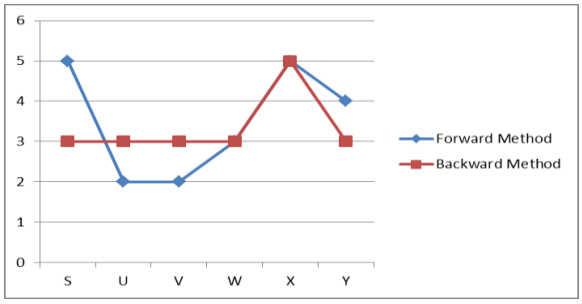

FIG.4 GRAPH FOR NUMBER OF COMPARISON IN SELECTION OF SHORTEST PATH BETWEEN FORWARD AND BACKWARD METHODS OF TABLE 3.

Table 4. The shortest distance values and number of the comparisons between different pairs of the edges in Fig.3

	Shortest Path Value		No. of comparison for selection of shortest path	
	Forward Method	Backward Method	Forward Method	Backward Method
T → S	9	9	5	3
T → U	5	5	2	3
T → V	3	3	2	3
T → W	5	5	3	3
T → X	8	8	5	5
T → Y	7	7	4	3

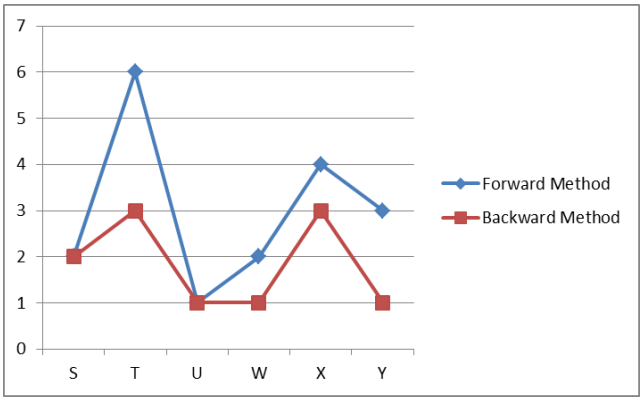

Fig.5 Graph for number of comparison in selection of the shortest path between forward and backward methods of Table 4.

I. VERIFYING THE SHORTEST PATH ALGORITHM

Before finding the algorithm solution, let us consider the following graphs G1 & G2 for the algorithm proof. This will enable us to exhibit the new shortest path algorithm and clarify how backward checking of edges have improved the effectiveness. From these graphs, we can conclude that both forward method and backward method take the nearest number of comparisons to produce the shortest path value. From this graph, we also conclude that in average cases backward search method takes less comparison compared to forward method.

Retrieval Number: J11350881019/19©BEIESP
DOI: 10.35940/ijitee.J1135.0881019
Journal Website: www.ijitee.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

A. A BACKWARD VERIFICATION ALGORITHM

The normal number of edges inspected by the backward check algorithm is kept running on Single Source Shortest Path of κ_n(EXP(1), is (1+O (1)) n log n. It isn't hard to check the backward algorithm by edges assessment, when the given tree is to be sure a tree of the most shortest paths, which are actually accessible in Spira's algorithm [6]. It filters the in-coming adjacency list of every vertex z, confirming the condition cost[x,z] ≥ dist[x] - dist[z], until it experiences an edge (z,x) in which cost[x,z] ≥ D - dist[z]. With this, the algorithm additionally utilizes conventional comparison-based priority queues of Williams [7] require O(log n) time. The straightforward adjustment of bucket based priority queue, and the part action of buckets into binary heaps just O(1) time per activity with high likelihood, and it permits executing Spira algorithm in O(n) time.

II. CONCLUSION

We introduced a new Single Source Shortest Path algorithm that works with backward method. The main concept of this algorithm rest on Spira's forward algorithm. From our implementations of the result analysis of the graphs we conclude that our backward algorithm also finds the same shortest path value in all possible paths between various sets of edges. From this, we conclude that this algorithm solves the Single Source Shortest Problem of complete directed graph with exponential edge weights, optimal and high probability in O (n) time.

REFERENCES

1. E. W. Dijkstra. "A note on two problems in connexion with graphs",1959, pp. 269-271.
2. Mikkel Thorup. "Undirected single-source shortest paths with positive integer weights in linear time", J. ACM, 1999, pp.362-394.
3. Tadao Takaoka and Alistair Moffat. "An O(n^2log n log log n) expected time algorithm for the all shortest distance problem". In Mathematical foundations of computer science, 1980 (Proc. Ninth Sympos., Rydzyna, 1980), volume 88 of Lecture Notes in Comput. Sci., Springer, Berlin, 1980, pp. 643-655.
4. Peter A. Bloniarz. "A shortest-path algorithm with expected time O(n^2log n log*n)", 1983, pp.588-600.
5. P. M. Spira. "A new algorithm for finding all shortest paths in a graph of positive arcs in average time O(n2log2n)", 1973, pp. 28-32.
6. David B.Wilson and Uri Zwick." A forward-backward single-source shortest paths algorithm", IEEE 54th Annual Symposium on Foundations of Computer Science, 2013, pp.707-716.
7. J.W.J. Williams. Algorithm 232: Heapsort. Communications of the ACM, 1964, pp.347-348.

AUTHORS PROFILE

Ms. D.JASMINE PRISKILLA pursued Bachelor of Science from University of Madras, in 1998. Master of Science from Bharathidasan University in 2000 and Master of Philosophy in 2002 from Mother Teresa University. She is currently working as Assistant Professor in Department of Computer Science and Computer Applications, Adhiyaman Arts and Science College for Women, Uthangarai, Tamilnadu, India. Her main research interest is in the area of Algorithms, Data Structures and Programming Languages. She has 19 years of teaching experience and 14 years of research experience.

Dr. K. ARULANANDAM is currently working as Assistant Professor & Head in the Department of Computer Science and Applicatuibs, Government Thirumagal Mills College, Gudiyattam. He has published several papers in reputed National and International jounals. He has 17 years experience in teaching and research experience.