Prograph Based Analysis of Single Source Shortest Path Problems with Few Distinct Positive Lengths

Biswajit Bhowmik
Assistant Professor
Dpt of Computer Science & Engineering
Bengal College of Engineering and Technology
Durgapur – 713 212
India
biswajitbhowmik@gmail.com

Sreyasi Nag Chowdhury
Dpt of Computer Science & Engineering
Bengal College of Engineering and Technology
Durgapur – 713 212
India
sreyasi_nc@yahoo.com

***Abstract* — In this paper we propose an experimental study model S^3P^2 of a fast fully dynamic programming algorithm design technique in finite directed graphs with few distinct nonnegative real edge weights. The Bellman-Ford's approach for shortest path problems has come out in various implementations. In this paper the approach once again is re-investigated with adjacency matrix selection in associate least running time. The model tests proposed algorithm against arbitrarily but positive valued weighted digraphs introducing notion of Pr*ograph* that speeds up finding the shortest path over previous implementations. Our experiments have established abstract results with the intention that the proposed algorithm can consistently dominate other existing algorithms for Single Source Shortest Path Problems. A comparison study is also shown among Dijkstra's algorithm, Bellman-Ford algorithm, and our algorithm.**

Keywords — shortest Path; negative weight cycle; weight matrix; variants; program-graph; SPL; dense network; scanned vertices; relaxation.

I. INTRODUCTION

Computation of *Shortest Path* (SP) is one of the most fundamental problems in graph theory. Both in operations research over and above theoretical computer science areas, the *Single Source Shortest Path Problem* $(SSSPP)$ is an extremely well-studied problem because of its broad applicability in a wide range of domains [1, 2]. The wide spectrum of its applications ranges from the routing problem in communication networks to robot motion planning, highway and power line engineering etc. Many optimization problems solved by dynamic programming or more complicated matrix searching techniques, such as the 0/1 knapsack problem, construction of optimal inscribed polygons, sequence alignment in molecular biology, length-limited Huffman coding etc, are expressed as shortest path problems. These also include scheduling problems such as critical path computation in PERT [3] charts. Moreover, the shortest-path problem as well has numerous variations such as the minimum weight problem, the quickest path problem etc. Our motivation for focusing on the algorithmic processes of these shortest path problems with few distinct edge lengths initiates from a problem that arises in social networks [1-7]. The *Shortest Path Problems* $(SPPs)$ have been, and still is, investigated by many researchers and mathematicians. With the rapid advancements and developments in communication, computer science and transportation systems, more variants of the $SPPs$ have appeared. Some of these are the traveling salesman problem, *K*-shortest paths, constrained shortest-path problem, multi-objective shortest path problem, network flow problems, and so forth including our key $SSSPP$ [2, 4, 8]. In this paper, we propose a model S^3P^2 that provides an algorithm similar to classical Bellman-Ford procedure that solves $SSSPP$. The model is represented with a digraph with few positive real edge weights introducing Pr*ogram Graph* or simply Pr*ograph* [9, 10] where each computation is performed at every node. The paper has been organized in the following sections.

- Problem Definition.
- Negative Weight Cycle.
- Representation of Weight Matrix.
- Mathematical Background.
- Shortest Path Variations.
- Proposed Work.
- Experimental Results.
- Comparison Study.
- Conclusion.

The Bellman-Ford's approach for shortest path problems has come into practice with different tools. The approach in this paper is re-investigated with adjacency matrix selection in associate least running time. The proposed model S^3P^2 on experiments has established abstract results with the intention that the proposed algorithm can consistently dominate other existing algorithms for the underlying problem. A comparison

study is shown among Dijkstra's algorithm, Bellman-Ford algorithm and our algorithm.

II. STATEMENT OF PROBLEM

Let us consider a weighted directed graph $G = (V, E, W)$ with vertex set V of size n, edge (or simply arc) set E of size m, and weight function $W : E \to R$ assigning a real valued weight (cost) to each edge in G. Then, a directed *Shortest Path Length* (SPL) from a given source (starting) vertex s to each other vertex v in G can be defined in (1) as:

$$SPL(s,v) = \begin{cases} \min(W(p) : s \xrightarrow{p} v), \text{ if } \exists \text{ a path from } s \text{ to } v \\ \infty \quad , \text{otherwise} \end{cases} \quad (1)$$

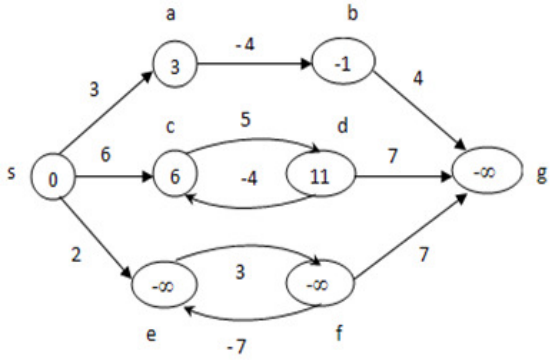
Here, $p = < v_1, v_2, ... v_k >$ is a *Shortest Path* (SP) from vertex v_1 to vertex v_k, $1 \le k \le n$. Therefore, SPL (also termed weight) of p is the sum of the weight of its constituent edges. Thus,

$$W(p) = \sum_{i=1}^{k-1} W(v_i, v_{i+1}) \quad (2)$$

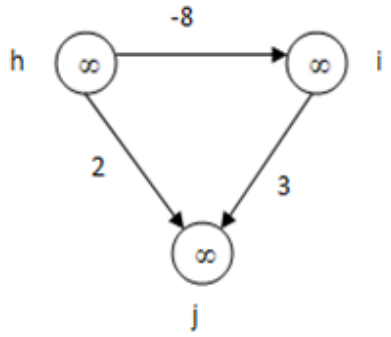
As a consequence, SPL from vertex s to v in G means any path p with weight, $W(p) = SPL(s,v)$ [4, 11, 12].

III. NEGATIVE WEIGHT CYCLE

We assume that all vertices are reachable from s otherwise unreachable vertices can be deleted from G in a linear-time at preprocessing steps. In some instances of $SSSPP$, there may be edges whose weights are negative. If the graph G has either no negative-weight edges, or no reachable negative-weight cycles from s then $\forall v \in V$, the $SPL(s,v)$ remains well defined during computation even if it has a negative value; otherwise $SPLs$ are not well defined. No path from s to a vertex on the cycle in this case can be a shortest path and a lesser-weight path can always be found that follows the proposed *shortest* path and then traverses the negative-weight cycle. If there is a negative-weight cycle on some path from vertex s to v, we define $SPL(s,v) = -\infty$. Illustration in Figure 1 exemplifies the effect of negative weights and negative-weight cycles on shortest path weights.



(a)



(b)

Fig. 1. Directed Weighted Graph with Negative Weight Cycle.

Within each vertex its shortest-path weight from source s is shown. Vertices e and f appear in a negative-weight cycle reachable from s, so they have shortest-path weights of $-\infty$. Because vertex g is reachable from a vertex whose shortest-path weight is $-\infty$, it, too, has a shortest-path weight of $-\infty$. Vertices h, i, and j are not reachable from s, and so their shortest-path weights are ∞, even though they lie on a negative-weight cycle [1, 8, 11, 13, 14].

IV. WEIGHT MATRIX REPRESENTATION

Edge weights $W(i,j)$ for the edges (i,j) can be interpreted as metrics (adjacency matrix, adjacency list etc) [8, 11, 15] other than distances. They are often used to represent time, cost, penalties, loss, or any other quantity that accumulates linearly along a path and that one wishes to minimize. In a weighted digraph G, W has been defined in (3) using adjacency matrix description of G as:

$$W(i,j) = \begin{cases} 0, & i = j \\ c, & \text{cost of the edge} \quad (i,j) \in E \\ \infty, & \text{otherwise} \end{cases} \quad (3)$$

Consider a digraph shown in Figure 2.

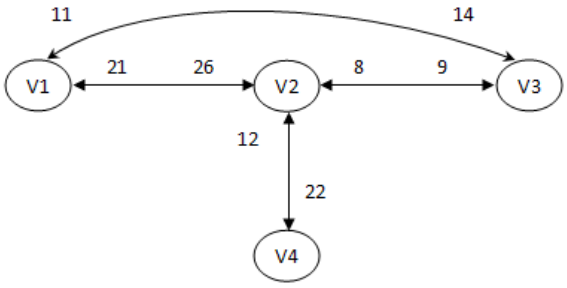


Fig. 2. Digraph without any self loop and negative weight

By definition in (3), weight matrix (also called cost matrix) would be constructed in Table I below.

TABLE I. ADJACENCY MATRIX OF THE GRAPH SHOWN IN FIGURE 2

W[i][j] =

Vi \ Vj	V1	V2	V3	V4
V1	0	26	14	∞
V2	21	0	9	22
V3	11	8	0	∞
V4	∞	12	∞	0

V. MATHEMATICAL INTERPRETATION

Shortest-paths algorithms typically rely on the property that a shortest path between two vertices contains other shortest paths within it.

Consider any i and j such that $s \le i \le j \le v$, let $p_{ij} = < v_1, v_2, ...v_j >$ is the sub-path of $SPL(i, j)$. Then, p_{ij} is a shortest path from vertex v_i to v_j. Let, $SPL^k[v]$ be SPL of vertex k from s under the constraint that SPL contains at most k edges. Obviously, $SPL^1[v] = W[s][v], \quad 1 \le v \le n$. Our goal is to compute $SPL^{k-1}[v], \quad \forall v \in V$ assuming this SPL contains at most $k-1$ edges and there are no cycles of negative length possible. Taking strength (advantages) of dynamic programming [4], the following observations [8, 11, 16] can be made.

- $SPL^k[v] = SPL^{k-1}[v]$, if SPL has at most k edges has no more than $k-1$ edges.
- $SPL^k[v] = \min\{SPL^{k-1}[v], \ \min_i\{SPL^{k-1}[v] + W[i][v]\}\}$, if some intermediate vertex i enters in SPL.

Both Figure 3 and Figure 4 [17, 18] in sections VI and VII respectively give an idea about the above discussion.

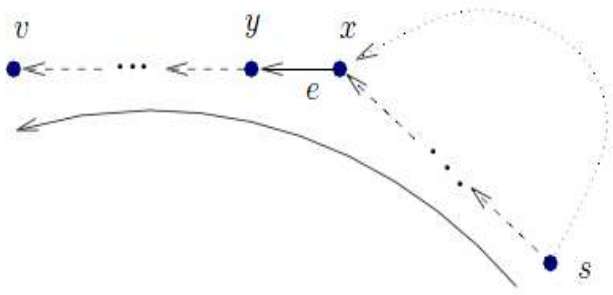


Fig. 3. How an edge enters the solution

Thus, the above observations could be a symbol of the following recurrence relation for SPL.

$$SPL^k[v] = \min\{SPL^{k-1}[v], \ \min\{SPL^{k-1} + W[i][v]\}\}, \quad 2 \le k \le n-1 \qquad (4)$$

VI. VARIANTS

A number of variations are possible depending on the type of network and costs involved, and source/destination pairs of vertices (nodes) for which we need solution [4, 8, 11, 19].

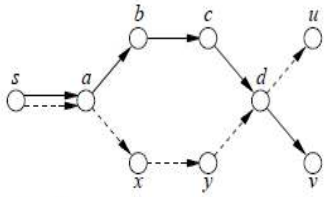
- *Cyclic or Acyclic problems* – Graph with at least one cycle. Otherwise, acyclic.
- *Non-negative or Negative distance problems* – If the distances (edge weights) are non-negative or if there is at least one negative distance.
- *Non-negative cyclic or Negative cyclic problems* – cyclic problems with non-negative length of all cycles or with at least one cycle has negative length.
- *Sparse or Dense network problem* – A network with m, number of edges, closer to n^2 (n is the number of nodes) is a dense network.
- *Single-source shortest path problem* – shortest paths from a source vertex s to all other vertices in the graph.
- *All-pairs shortest path problem* – shortest paths between every pair of vertices v, v' in the graph.
- *Single destination shortest path problem* – shortest paths from all vertices in the graph to a single destination vertex d.

Thus, a problem of finding a shortest path in many network as well as transportation related problems may arise as a main decision question or as a step in some situation.

VII. PROPOSED WORK

The literature on the $SSSPP$ is large, since computing shortest paths in a given graph (both directed and undirected) can be done in various ways. For example, consider Figure 4 [18]. It shows how an intermediate vertex $k \in V$ enters into a tentative solution of shortest path from a source vertex s to target vertex v. As one of the fundamental problems,

competent way for finding SPL is still an important field of research in computer science.



If $s \to a \to b \to c \to d \to v$ and $s \to a \to x \to y \to d \to u$ are both shortest paths, then $s \to a \to b \to c \to d \to u$ is also a shortest path.

Fig. 4. Various ways to compute shortest path from source

The proposed algorithm for S^3P^2 model is based on computation of cost of each vertex independently. The algorithm maintains a cost $SPL[v]$ for each vertex v tentatively, such that some path from s to v has total cost $SPL[v]$. As the algorithm proceeds, these costs will decrease in succession until for each vertex v, $SPL[v]$ is the cost of a minimum-cost path from s to v. Instead of normal termination, the proposed algorithm finishes execution at the moment when two successive stages of the algorithm, i.e. two consequent instances of the $\Pr ograph$ of the given G, have the same shortest path cost $SPL[v]$ for each vertex v. Initially, it is assigned that $SPL[s] = 0$ and $SPL[v] = \infty$ for every $v \neq s \in V$ [13]. The algorithm maintains n partitions at most of the n vertices. Each partition consists of each vertex v and its corresponding adjacent vertices. Each partition acts as respective node of our $\Pr ograph$ with node s as its root. Still, these n partitions can be sub-grouped into three states:

- *Unlabeled Vertices:* those with infinite provisional costs.
- *Labeled Vertices:* those with finite provisional cost whose minimum cost is so far unknown.
- *Scanned Vertices*: those whose minimum cost is known.

We start with s as labeled root node and all other vertices are as unlabeled. The *Prograph Based Shortest Path Algorithm* named as $\Pr ograph\_Shortest\_Path()$ described below executes comparable to Bellman-Ford approach to some extent.

A variety of methods and algorithms are available for the solution of $SSSPPs$ depending on the nature of specific problem [19]. Because of the nature of our problem, the algorithm suggested in this paper consists of the following steps and repeats till all vertices are scanned:

Algorithm: Prograph_Shortest_Path (G, E, V, W)

Where G = weighted connected graph
E = set edges in G.
V = set of vertices in G.
W = weight matrix (in adjacency matrix form).
Assumptions: Negative weighted edge (s) and self loop(s) are removed.

```
(1)  Initialize: p = q = 1
(2)  Set i = source_vertex = 1
(3)  Repeat for i = 1 to |E|
(4)    Source[p++] = source_vertex (Ei)   // keep source
                                          // vertices
(5)    End[q++] = terminal_vertex (Ei)    // keep terminal
                                          // vertices
(6)  M=1        // determines number of nodes in prograph
(7)  Repeat for j = 1 to |E|
(8)    If  i = Source(j)
(9)       Continue
(10)   Else
(11)      i= Source(j)
(12)      M++
(13) Set i = 1              // source vertex
(14) SPL[i] = 0 // source distance is 0
(15) k = 1
(16) X[k++] = Y[k++] = 0          // lists to check
                                  // termination condition
(17) Repeat for i = 2 to |V|
(18)   SPL[i] = ∞                 // initialize shortest path to
                                  // all vertices from source
                                  // with unknown value.
(19)   X[k++] = Y[k++] = 0
(20) Z = 1
(21) T = 0                        // termination condition
(22) N = 1                        // node 1 of prograph
(23) k = j = i = 1
(24) Repeat while  Source[j] = i through line 32
     // compute SPL from N to its adjacents
(25)   If End[j] = source_vertex
(26)      j++
(27)      Continue      // skip edge (Source(j), End(j))
(28)   SPL[End(j)] = min{SPL[End(j)], SPL[Source(j)]
                        + W(Source(j), End(j)}
(29)   Y[k++] =1
(30)   j++
(31) i = Source[j]
(32) N = N + 1
(33) If N ≤ M
(34)   Go to line 24.
(35) Repeat for  i = 1 to |Y|
(36)   If X[i] ≠ Y[i]
(37)      T = 1
(38)      Break
(39) Repeat for i = 1 to |Y|
(40)   X[i] = Y[i]
(41) If T = 1
(42)   Z++      // determines number of iterations
(43)   Go to line 21.
(44) Exit
```

Complexity Analysis: The running time complexity of the $\Pr ograph\_Shortest\_Path()$ algorithm could be analyzed in the following ways: (1) The number of edges E with non-negative weight can be found in Ω(|E|) time. (2) Lines 3 – 5 enqueue source and terminal vertices in two lists needed for *Relaxation* [8] and needs Ω(|E|) time. (3) Lines 6 – 12 compute number of nodes M which are to be in our $\Pr ograph$. This can also be computed in Ω(|E|) time. (4) Now, M ≤ |V|, before relaxation of edges, all the vertices are assumed to have unknown cost (shortest path reachable from source) with reachable path length of source vertex s is $Source[1]=0$. These initializations necessitate Ω(|V|) time. (5) Whenever two consecutive instances of the $\Pr ograph$ of the underlying problem are identical, relaxation is stopped. This checking is done Z (a constant) times. This Z naturally is reasonably a very small non-negative integer number. Generally Z is 2. (6) Reachable tentative path lengths from source vertex s to other vertices v are computed through lines 21 – 43. These relaxations will be continued till $X=Y$. Thus, overall running time complexity of the algorithm executes in Θ(Z|V||E|) time. Since Z is a constant and has numerical value 2 in general, thus the proposed algorithm runs in Θ(|V||E|) time.

VIII. EXPERIMENTAL RESULTS

In this section we will now look at an example of how our suggested algorithm works on a weighted digraph shown in Figure 5 to solve $SSSPP$ in optimal way with $\Pr ograph$.

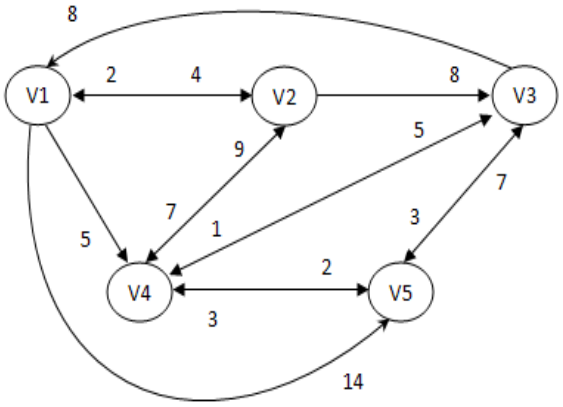


Fig. 5. Weighted Digraph for SPL computation

Now, the edge set E = {(V1,V2), (V1,V4), (V1,V5), (V2,V1), (V2,V3), (V2,V4), (V3,V1), (V3,V4), (V3,V5), (V4,V2), (V4,V3), (V4,V5), (V5,V3), (V5,V4)}.

When there are no cycles of negative length, we know that there is a shortest path between two vertices. The digraph in Figure 5 contains $n=|V|=5$ vertices. So, a SPL between any vertices has at most $n-1$ edges on it. The proposed algorithm during execution examines updating SPL on the cost matrix W constructed in Table II along with its reflection on the nodes of the $\Pr ograph$ worked out in sequence from Figure 7 through Figure 12.

TABLE II. ADJACENCY MATRIX OF THE GRAPH SHOWN IN FIGURE 5

W[i][j] =

Vi \ Vj	V1	V2	V3	V4	V5
V1	0	4	∞	5	14
V2	2	0	8	7	∞
V3	8	∞	0	1	3
V4	∞	9	5	0	2
V5	∞	∞	7	3	0

Suppose, $s=V1$, the first vertex (source of journey) is appeared in SPL. We follow the convention vertex $V\,i$ simply as vertex i. So, path length for s is $SPL[V1]=SPL[1]=0$. For simplicity, we need to have SPL from this s to remaining vertices. Typical initial configuration of the $\Pr ograph$ would look like as Figure 6.

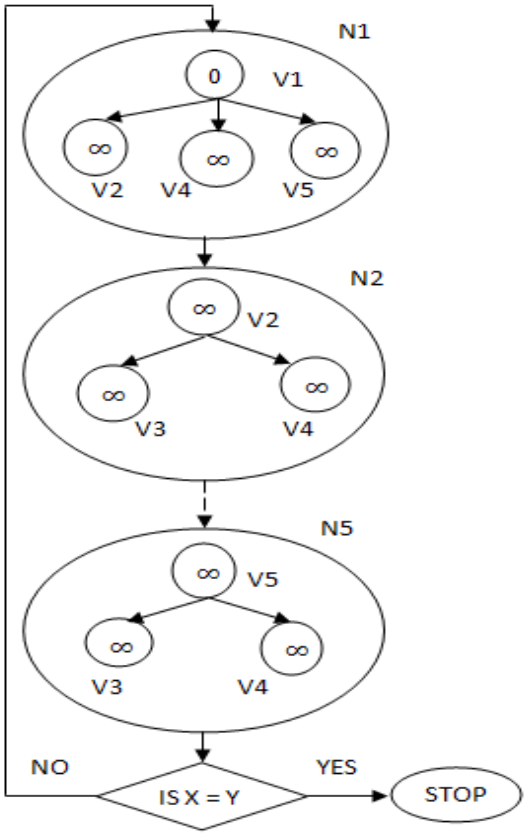


Fig. 6. Initial arrangement of Prograph for *SPL* from vertex 1

Node 1 consists of source vertex s and its adjacent vertices. Similarly, it is applicable to other nodes. However, each node each is related in top-down manner. Tentative path length value of a vertex in a node of the $\Pr ograph$ will be treated with its present value in next node if the vertex appears. But source vertex s would not be appeared as adjacent of the current

vertex in current node. Following iterations give details how to compute SPL from s through *Relaxation* [8] of edges in E.

Iteration I:

X[1… n] = Y[1… n] = {0}

Compute Node 1 (N1) relaxing edges with starting vertex 1 i.e. $V1$

SPL [2] = min {SPL [2], SPL [1] + W [1][2]}
= min {∞, 0 + 4}
= 4

Y[2] = 1

SPL [4] = min {SPL [4], SPL [1] + W [1][4]}
= min {∞, 0 + 5}
= 5

Y[3] = 1

SPL [5] = min {SPL [5], SPL [1] + W [1][5]}
= min {∞, 0 + 14}
= 14

Y[4] = 1

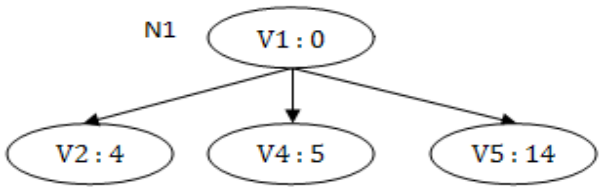


Fig. 7. Computation at Node 1

Relaxing edges with starting vertex 2 i.e. $V2$

SPL [3] = min {SPL [3], SPL [2] + W [2][3]}
= min {∞, 4 + 8}
= 12

Y[5] = 1

SPL [4] = min {SPL [4], SPL [2] + W [2][4]}
= min {5, 4 + 7}
= 5

Y[6] = 1

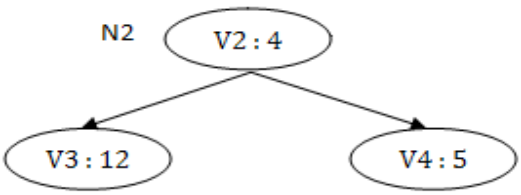


Fig. 8. Computation at Node 2

Relaxing edges with starting vertex 3 i.e. $V3$

SPL [4] = min {SPL [4], SPL [3] + W [3][4]}
= min {5, 12 + 1}
= 5

Y[7] = 1

SPL [5] = min {SPL [5], SPL [3] + W [3][5]}
= min {14, 12 + 3}
= 14

Y[8] = 1

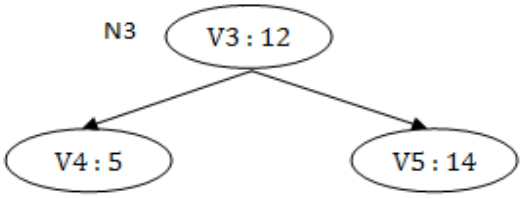


Fig. 9. Computation at Node 3

Relaxing edges with starting vertex 4 i.e. $V4$

SPL [2] = min {SPL [2], SPL [4] + W [4][2]}
= min {4, 5 + 9}
= 4

Y[9] = 1

SPL [3] = min {SPL [3], SPL [4] + W [4][3]}
= min {12, 5 + 5}
= 10

Y[10] = 1

SPL [5] = min {SPL [5], SPL [4] + W [4][5]}
= min {14, 5 + 2}
= 7

Y[11] = 1

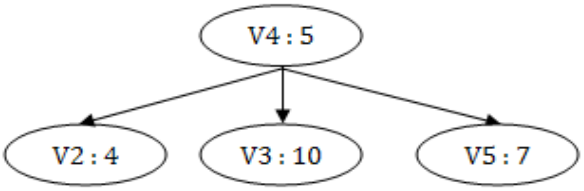


Fig. 10. Computation at Node 4

Relaxing edges with starting vertex 5 i.e. $V5$

SPL [3] = min {SPL [3], SPL [5] + W [5][3]}
= min {10, 7 + 7}
= 10

Y[12] = 1

SPL [4] = min {SPL [4], SPL [5] + W [5][4]}
= min {5, 7 + 3}
= 5

Y[13] = 1

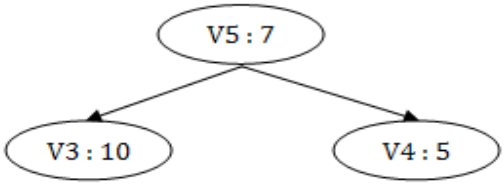


Fig. 11. Computation at Node 5

Now Y is compared with X. If both are same, no further visit of nodes in the $\Pr ograph$ takes place. Otherwise content of Y is transferred to X follow by start computing SPL from s again as the above procedure. In Iteration I it is just observed that $X \neq Y$. Computation needs again relaxation beginning with first edge (Source [1], End [1]). After Iteration I the $\Pr ograph$ would look like as illustrated in Figure 12.

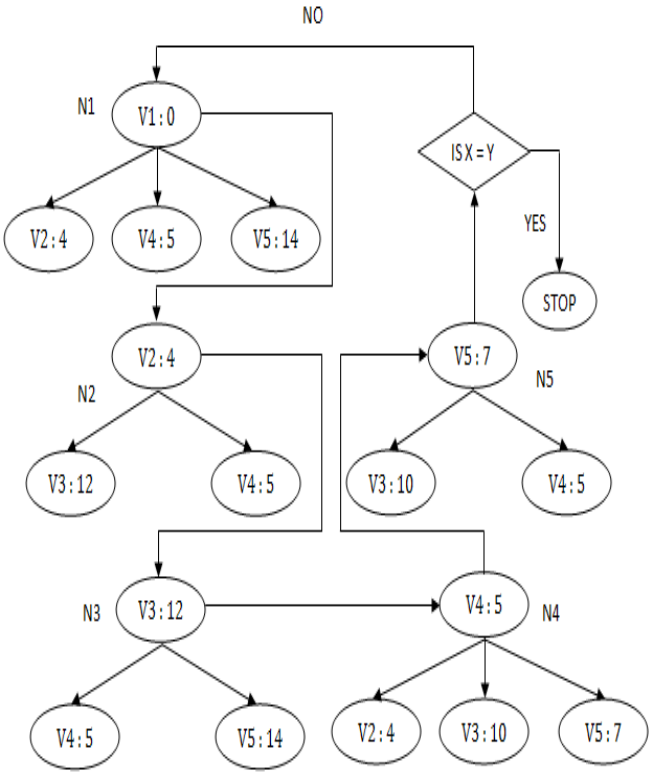


Fig. 12. Shortest Path Length from Vertex V1 to Others in Iteration I

Minimum path length in terms of edge cost from source vertex $V1$ to all remaining vertices is shown in association with starting vertex of every node Ni, $1 \le i \le |V|$. Proceeding in the same way (Z times) we would have our preferred solution. As $Z = 2$, the $\Pr ograph$ of the final solution would be same as portrayed in Figure 12 at the end of Iteration II, since no change does takes place. Thus, SPL from source vertex s to rest of the vertices can be viewed at a glance like as Figure 13.

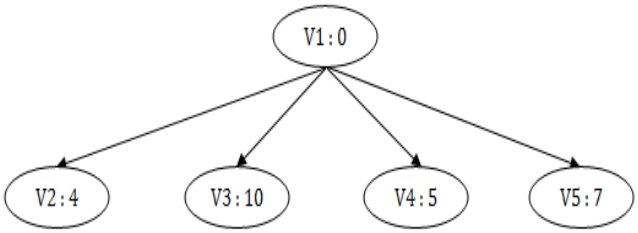


Fig. 13. Shortest Path Length from Vertex V1 to Others Fig. 5

Every node $(Ni:c)$ in Figure 13 represents a node Ni, $1 \le i \le |V|$ and its associated cost say c $(i.e. SPL)$ from s.

IX. COMPARISON STUDY

The S^3P^2 model could be solved easily with BFS [8, 16] graph traversal algorithm under a special case when all weights are 1 or shortest paths with unweighted graphs [12]. Theoretically, most efficient known algorithm for the problem was devised by Dijkstra in the year 1959 and since then it is known as the Dijkstra's algorithm [8, 16]. It is believed that first polynomial run time algorithm for *Shortest Path Problem* which is based on Greedy approach is this Dijkstra's algorithm. Dijkstra's original algorithm runs in $O(n^2)$ time. Later the algorithm has been implemented more efficiently in $O((m + n)\log n) \approx O(m \log n)$ time. However, this algorithm does not work on the digraphs with negative edge weights. Richard Bellman, Samuel End and Lester Ford Jr. solved the underlined problem with more powerful approach called Dynamic Programming [8, 11, 16]. And the algorithm they devised is popular as Bellman-Ford algorithm. It runs in $\Theta(n^3)$ time and $\Theta(mn)$ time whenever adjacency matrix and adjacency list are used correspondingly [16]. Although few non-negative edge weights have been considered in the proposed S^3P^2 model, it allows negative edge weights without any "*Loss of Generality*" [8, 11] of the Dynamic Programming Approach. Therefore, this algorithm which also runs in $\Theta(mn)$ time using adjacency matrix, should be considered preferable due to its open acceptability and powerfulness of Dynamic Programming [4] and superb easy understandable way to represent solution using $\Pr ograph$. The illustration computes a shortest-path tree similar to BFS tree.

X. CONCLUDING REMARKS

Our optimization technique is to detect vertices with exact distance labels from source vertex s. It should be noted that the proposed algorithm is competitive to algorithms used commonly in practice for shortest path computations. We implemented our approach without adjacency list representation of cost matrix and are able to estimate overall complexity in $\Theta(|V||E|)$ time. Two lists X and Y in this paper

have replaced the significance of the adjacency list. We showed that our algorithm has better time complexity over classical Bellman-Ford algorithm. Still no algorithm is known that is asymptotically faster for this *SSSPP* . Before we draw a concluding remark it should have to be brought up to date that our algorithm suggests and in addition yields exact result whenever negative weighted edges are allowed.

ACKNOWLEDGMENT

The first author would like to express heartily gratitude to his beloved Madhurima Mondal and her friends, B.Tech 3rd Year Students, Department of Computer Science & Engineering, Bengal College of Engineering and Technology, Durgapur, India for their useful notes with solved assignments without which the draft of this paper would not be completed in time. The author would also like to convey his thankfulness to Dr. S. Ranbir Singh, Assistant Professor, Department of Computer Science & Engineering, Indian Institute of Technology Guwahati for his valuable suggestions, and remarks.

REFERENCES

[1] James B. Orlin, Kamesh Madduri, K. Subramani, M. Williamson , "A faster algorithm for the single source shortest path problem with few distinct positive lengths", Journal of Discrete Algorithms, Elsevier, Vol. 3, No. 1, pp. 1-10, 2009

[2] Ammar W. Mohemmed, Nirod Chandra Sahoo, "Efficient Computation of Shortest Paths in Networks Using Particle Swarm Optimization and Noising Metaheuristics", Discrete Dynamics in Nature and Society, Vol. 2007, No. 4, pp. 1-25, 2007

[3] Rogger S. Pressman, Software Engineering – A Practioner's Approach, Mc-Graw Hill, 2001

[4] Biswajit Bhowmik, "Dynamic Programming – Its Principles, Applications, Strengths, and Limitations", International Journal of Engineering Science and Technology, Vol. 2, No. 9, pp. 4822 – 4826, 2010

[5] F. B. Zahn, C. E. Noon, "Shortest path algorithms: an evaluation using real road networks", Transportation Science, Vol. 32, No. 1, pp. 65–73, 1998

[6] G. Desaulniers, F. Soumis, "An efficient algorithm to find a shortest path for a car-like robot", IEEE Transactions on Robotics and Automation, Vol. 11, No. 6, pp. 819–828, 1995

[7] David Eppstein, "Finding the k Shortest Paths", 1997, available at http://www.ics.uci.edu/»eppstein/.

[8] Thomas H. Coremen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, PHI, 2008

[9] Biswajit Bhowmik, "Studies on Dimultigraph and Prograph Based Applications of Graph Theory in Computer Science", International Journal of Computer and Communication Technology, Vol. 1, No. 2, 3, 4, pp. 57-61, 2010

[10] N. Deo, Graph Theory with Applications to Engineering and Computer Science, PHI, 2002

[11] Biswajit Bhowmik, Design and Analysis of Algorithms, S. K. Kataria and Sons, 2011

[12] http://gabrielistrate.weebly.com

[13] Ravindra K. Ahuja, Kurt Mehlhorn, James B. Orlin, Robert E. Tarjan, "Faster Algorithms for the Shortest Path Problem", Journal of the Association for Computing Machinery, Vol. 37, No. 2, pp. 213-223, 1990

[14] M.H. Alsuwaiyel , Algorithms Design Techniques and Analysis, PHEI, 2003

[15] D. Samanta, Classic Data Structures, PHI, 2010

[16] Ellis Horowitz, Sartaj Sahani, Sanguthevar Rajasekaran, Fundamentals of Computer Algorithms, Golgotia, 1998

[17] Surender Baswana, Tobias Friedrich, Somenath Biswas, Piyush P. Kurur, Benjamin Doerr, Frank Neumann, "Computing Single Source Shortest Paths using Single-Objective Fitness Functions", FOGA'09, January 9–11, 2009

[18] http://theory.cs.uiuc.edu/~jeffe/teaching/algorithms/notes/13-sssp.pdf

[19] Farrukh Shehzad, Muhammad Akbar Ali Shah, "Evaluation of Shortest Paths in Road Network", Pakistan Journal of Commerce and Social Sciences Vol. 3, pp. 67 – 79, 2009

AUTHORS PROFILE

Biswajit Bhowmik is a very renowned faculty member currently with Bengal College of Engineering and Technology, Durgapur, India as Assistant Professor in the Department of Computer Science & Engineering. He is a member of different professional bodies such as IEEE, IACSIT, IAENG, PASS, IAOE, and UACEE. He is also member of some leading professional societies such as IEEE Computer Society, IEEE Communications Society, IAENG Society of Computer Science, IAENG Society of Software Engineering, IAENG Society of Wireless Networks, and IAENG Society of Artificial Intelligence. He is reviewer of several international journals such as IJCSIC, IJCSIS, JACSM, IJoAT, JWMC etc in the area of computer science. He has authored a book titled *Design and Analysis of Algorithms*. He has many publications in international journals including international conference proceedings on the subjects ranging from Algorithms Analysis, Graph Theory, Compiler Design, and Mobile Computing. In addition his area of interests includes Data Structures & Algorithms, Software Engineering, Computational Geometry, and Green Computing. He has guided several projects at under graduate level.

Sreyasi Nag Chowdhury is a 3rd Year UG (B.Tech) student in the Department of Computer Science & Engineering at Bengal College of Engineering and Technology, Durgapur, India.