International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 7– No. 1, April 2014 – www.ijais.org

An Efficient Technique for Computing Shortest Path Tree in Dynamic Graphs

Neeraj Kumar Maurya
Lecturer TCET
IT department
Kandivli(E),Mumbai.400101 ,
India

R. R. Sedamkar, Ph. D
H.O.D. Computer Engineering Department
Thakur College Of Engineering And Technology
Mumbai 400 101
India

Abstract

This paper proposes an efficient technique for computing shortest path in dynamic graph. Which finds shortest path in a given graph which is static or intended to change its weight frequently. If that graph is static i.e. not changing its weight then SPT is being calculated once and that remains same. If graph is dynamic i.e. changing its weight then this technique finds new SPT with traversing minimum number of nodes or vertices. This technique extends a few state-of-the-art dynamic SPT algorithms to handle multiple edge weight updates, and find the SPT. A function based on the location of current node/ state is used to vary the cost of the goal node and the search is done with minimum the state space and exploring only affected nodes, by using these approaches problem is solved in minimum time. Based on experimental results on sample data set we propose to device an algorithm which efficiently handles different traffic conditions. The performance of this algorithm is measured on the basis of Graph size, number of changed edge (NCE). To evaluate the proposed dynamic algorithm, comparison is done with the well-known static Dijkstra algorithm. Where proposed algorithm's complexity is $O(b^d)$ in worst case O(E) in average case and O(1) in best case.

General Terms

Shortest Path Tree (SPT), Dynamic Tree, directed graph.

Keywords

Dynamic Dijkstra, Dynamic Graph, Graph, Directed Graph.

1. INTRODUCTION

Graph is a collection of nodes (vertices) and Edges. Directed Graph is a graph is one in which the direction is also given to all the edges, which shows that how one can travel from one node to another. If any edge E is having direction from Vertex V1 to Vertex V2, then one can travel from V1 to V2 but not vice versa. Weighted directed graph is a graph in which all the edges having their weight, i.e. travelling time or distance. Connected Graph is one in which there exist at least one path from any vertex to any other vertex.

Travelling time can be reduced by choosing shortest path from source to destination. But searching for shortest path is also a big concern, since for n number of vertices one will have to scan for all the vertices and their connectivity. For any connected graph there would be at least n-1 edges for n number of Vertices. For fully connected graph n(n-1)/2 edges are required. To find SPT complexity will be O(n2). This approach will give us the solution but is not efficient for large graph, and if that graph is dynamic i.e. where edges are changing its weight frequently then to compute entire SPT doesn't seem a good solution, since here system is recompiling that algorithm from the start vertex even that is not required.

Where the graph is changing its parameter like weight assigned to the edges is known as dynamic graph. Proposed paper is going to give an idea that how we can get SPT without exploring unaffected vertices again and how we can use heuristic information to cut down the cost.

2. RELATED WORK

2.1 Fully Dynamic Algorithm

Frigioni et al.[1] proposed a fully dynamic algorithm, for finding Shortest Path Tree in the given dynamic graph. In this algorithm each node has to maintain information of its parent node which restrict the number of nodes to be scanned each time when ever weight changes. It follows the same approach as Dijkstra's algorithm does but this is with minimum number of node search. According to author this algorithm is having less theoretical complexity but the data structures used are complex which makes the system inefficient, since its time consuming. This is shown experimentally in [2] by Frigioni that it is performing well.

2.2 BallString

Narváez et al. [3] propose an algorithm for SPT, where its finding SPT for the given graph and at the same time if any changes in weights occur it re-arranges themselves. And this is having a central idea of Ball and string model where some balls are tied together and any one of these is picked up and automatically all the balls are arranged with SPT i.e. all stretched strings shows travelling path between nodes. If any thread is shortened then all the affected nodes will set itself. In [4] one intelligent approach is proposed to recompute the SPT for the multiple weight changes in any dynamic graph. Here algorithms is called ball-string model since affected nodes are rearranging them in natural way with minimum complexity and in economical way, its really a great idea. This approach reduces unnecessary changes for any update in edges and do not process those parts which is not affected in any case i.e. after evaluating SPT again will give same results for those nodes.

In [5] this is shown that unfortunately this algorithm does not work correctly for multiple weight changes and updates.

2.3 DynamicSWSF-FP

Ramalingam and Reps in [5] propose a fully dynamic algorithm, DynamicSWSF-FP, Where they have given algorithm to update the SPT for any changes in edges. Here tree is used to compute minimum number of steps to derive any terminal strings from one or more non-terminal strings by using production rules. Here a graph is generated which can vary its edge weight. The main approach is given as follows: At any instant, a "right-hand side" (rhs) value, which maintains every vertex in the Graph(G) and denoted as rhs(v). this tracks the shortest distance which v gets by their all parents. dv is the shortest distance information for each vertex v in G, and one equality is represented as dv=rhs(v) before any input edge weight updates. As soon as edge weight is updated, DynamicSWSF-FP updates affected vertices, and it tries to access minimum number of edges and makes it equal to rhs(v) again.

A disadvantage of DynamicSWSF-FP is that it computes the rhs value too often, which leads to a high number of edge visits. In the same paper [6], the authors suggest some improvement on computing rhs values incrementally. The authors maintain a heap for each affected vertex. The improved algorithm is proven to be correct, but too many heaps may not be practical.

3. PROPOSED ALGORITHM

Here proposed system is dividing the graph in basically two parts one is which is not affected from the weight changes and other is affected. For this following algorithm is used.

In proposed algorithm, updation of the tree is divided into four parts.

If weight of any edge is increasing and that is not the part of SPT.

If weight of any edge is increasing and that is part of SPT.

If weight of any edge is decreasing and that is not part of SPT.

If weight of any edge is decreasing and that is part of SPT.

3.1 Dynamic Graph

All the above given cases may occur one by one or all together. For these mentioned cases we have implemented our algorithm and tested for multiple edge weight changes with all combinations.

Our focus is to just update only affected part of the tree. If we shall do that then the size of tree which is to be updated will de reduced drastically, and it shall reduce time required to compute SPT for new updated graph. If we talk about the time complexity to find SPT then, first time it will take same time as Dijkstra's SPT time, because at that time no SPT is calculated before and it will be calculated according to the Dijkstra's algorithm only. But for the next and so forth when any updation will occur then it shall visit only affected part and give new SPT for that graph.

The question which may arise is why the weight of the edges will increase? The reason is traffic or any other variable factor which changes or may change with time and usage, or it may be the failure of access of that link (edge), because of which one will have to select new path to travel. So the weight of the edges may increase or decrease.

For this type of approach some papers and research works are referred which are discussed in further sections. Some of them algorithms like Ball-and-String model and FMN algorithm are semi-dynamic(either working for increasing or decreasing weights) and some are not correct. Here the proposed algorithm is fully-dynamic algorithm and working with all conditions.

3.2 Used Datastructures and Algorithm

Our concern is to find new SPT whenever there is any change in the existing graph. And these changes may occur in some possibilities. Here in proposed algorithm the problem has been divided in four sub problems.

1. Weight of any edge is increasing and that edge is part of existing SPT.
2. Weight of any edge is increasing and that edge is not part of existing SPT.
3. Weight of any edge is decreasing and that edge is part of existing SPT.
4. Weight of any edge is decreasing and that edge is not part of existing SPT.

For all these, case sections are proposed separately, and only affected part of the graph is focused so that time could be minimized.

Variables and data structures which are used in algorithm are as follows:-

- **settledNodes** having set of vertex which are settled.
- **unSettledNodes** having set of vertices which are traversed but not connected to tree yet.
- **nonVisitedNodes** having set of vertices which are never visited and not connected to the graph having source.
- **predecessors** used to store vertices and their predecessor.
- **distance** is used to map distance of that node from source.

Methods used:-

- **removeSettled(Vertex)**: this method will be used to remove settled vertex from the settledNodes if that is affected by the weight change.
- **executeAfterChangeD(Vertex, int):** this method will be called when the weight of any edge is decreased and edge is in SPT by passing its destination vertex and difference in weight.
- **executeAfterChangeI(Vertex):** this method will be called when weight of any edge is increasing and that edge is present in SPT.
- **executeAfterChangeDecreaseNotConnected (Vertex):** this method will be called when weight of any edge is decreasing and that edge is not present in SPT. If weight of any edge is increasing and that edge is not present in SPT then just weight of edge will be updated.

Now we will look into the methods that what actually are they doing?

1. executeAfterChangeD(Vertex source, int diff)

1. Extract neighbour of that source node and store them in neighbor list
2. If the predecessor of that neighbor is this source

Then

just update the distance of that node by passed diff value.

Call

executeAfterChangeD(Vertex node, int diff)

Else

If that node is not successor of this source node in SPT

Then

Check the distance of node if it's decreasing

Then

Make that node unsettled. And start calculating SPT for that section

3. End.

2. executeAfterChangeI(Vertex source)

1. Make unsettle this source node and calculate its shortest distance
2. For all successive nodes do the same by calling and pass this current node as source **executeAfterChangeI(Vertex source).**

3.executeAfterChangeDecreaseNotConnected(Vertex source)

1. Find all parent node of this source node.
2. With all parent nodes check it's distance and select minimum one.
3. Change its predecessor to that parent node.
4. Now take all successors one by one and update their costs.
5. Stop.

4. EXPERIMENTAL PROOF AND EXAMPLES

4.1 Proofs

In experiment one randomly generated graph having given number of nodes and random number of edges is considered. And the SPT of that graph is found by using both the algorithms one which we have proposed and other is well known Dijkstra's algorithm.

Now have a look on all the cases one by one and analysis on time taken by proposed algorithm. Let us say weight of edge from node 0 to node 2 is updated to 2 from 6. Then in this case only encircled part of the tree will be traversed first, their distances from the source node will updated by 4 i.e. the difference in weight. In this case these nodes which are present in successor to that edge will not change their parent since they are already forming SPT and if weight is decreasing in that SPT then it is going to be remain same.

The proof can be given as follows.

1. **Case 1. If weight of any edge is increasing and that is not the part of SPT**

   As in Fig.2 two node are there which are present in SPT with cost c and c*. Here c+w > c* therefore node with cost c* is not attached in success of cost c.

   In such case any increment in w will not make any changes in SPTs.

   Lemma:- If edge increases its weight from w to w+d then since c+w > c* => c+w+d > c*, no need to scan any node further just update weight of that edge and stop.

2. **Case 2. If weight of any edge is increasing and that is part of SPT.**

   With the reference of Fig: 3 nodes n1 and n2 are two nodes present in SPT with different-different sub trees. In this case if cost c increases to c+d then all its successors will get unsettled i.e. need to check all the nodes present in that subtree.

   Lemma: c < c*+w and in case if c+d > c*+w because here d is always positive so it may occur or some where any other successive node it may be and at that time those nod will get unsettled will need to be attached to its some other parent nodes having minimum cost.

3. **Case 3. If weight of any edge is decreasing and that is not part of SPT**

   When weight of that edge decreases which is not present in SPT then in that case node which is successor to that edge is to be checked and if this cost is not less than previous cost of that node then we will just update edge weight and stop there only. And in case if this newly calculated cost is less than previous cost then that node will get unsettled and get its new parent this parent is the origin of that edge. And no need to do any other changes for further successive nodes. But those nodes which are successors of that subtree nodes but not present in that subtree for that SPT, may get attached to them since here it is possible to get other minimum cost path due to decrease in weights in predecessor's edge, so here we will have to check all those nodes also which are successors of those nodes which are present in that SPT subtree.

   Lemma:we shall consider situation in Fig.1. where n1, n2 and n3 are in same subtree and n2, n5 are in other subtree. It is sure that c+c* < c'+w that's why that is in other subtree. If weight w decreases by d then it may be the case that c+c* > c'+w-d and in that case n3 will change its parent node to n2. And n4 is having its minimum cost via n3 i.e. c+c*+c" is smaller than all other costs which are reachable to n4. Or we can say c" is already shortest path from n3 and edge from n2 to n3 is not present in SPT but after decreasing its weight it may give minimum cost but this is not going to affect subtree having root n3, so no need to check for its successors.

   Now taking case of node n5, assume there is any edge from n4 to n5 with weight w', previously n5 was not present as a successor of n4 in that subtree in which n4 is present but when w reduces by some weight d then it may be the case that c'+w-d+c"+w'< c'+c'*, and n5 will get attached to n4.

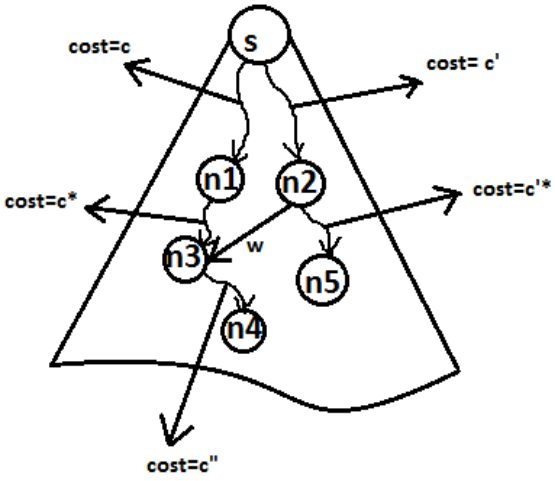

Fig 1: Edge from n2 to n3 is not in SPT and weight w decreases its weight by d.

4. Case 4. If weight of any edge is decreasing and that is part of SPT

If cost of any node from the source is c* and the edge at the destination of which this node is attached decreases its weight by d, then node having cost c* will have now c*-d which is again minimum.

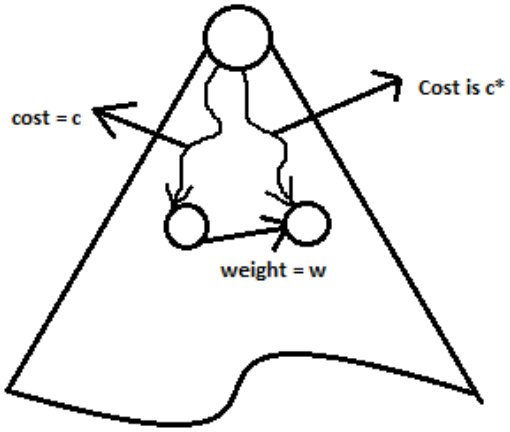

Fig 2: SPT having two subtrees with costs c and c* for two distinct nodes.

Before updation of cost=c*. and this is present in SPT i.e. c*< c+w. here c*decreases to c*-d.

c*< c+w => c*-d < c+w . it proves that nodes present in that sub tree will not change their parents, but nodes which are not present in that subtree may get updated minimum cost so they may change their parent as explained and proved in next paragraph.

Here n1 and n2 are two nodes of SPT but having different subtrees.Node n2 is not successor of n1 since c*+w > c, if c* decreases by d then c*-d+w < c may be the case, and in this case n2 will get unsettled and attached to n1 and all its successors will also be shifted and in case if c*-d+w > c then that successor node will remain at same position as it is having.

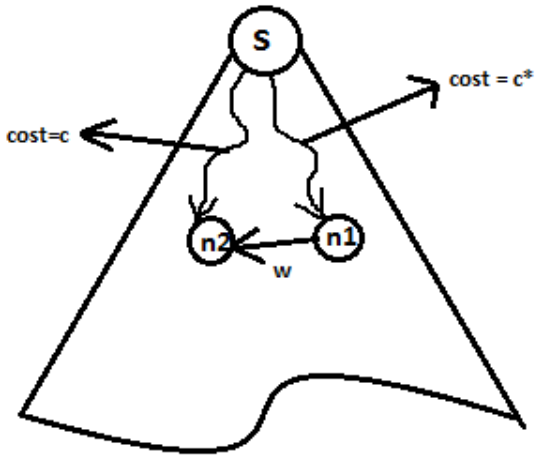

Fig 3: SPT having two subtrees with costs c and c* for two distinct nodes n2 and n1 respectively. Node n2 is also successor of n1 with cost w.

4.2 Factors Evaluated

Graph Size: This is number of nodes (vertices) and number of edges connecting them.

Number of edges updated: This is number of edges changing its value, either increasing or decreasing.

4.3 Experimental Results

As per the proposed algorithm here one small sample is taken as shown in Fig.4 having 19 vertices and 44 edges and explained its complexity while travelling and updating SPT when any edge or group of edges is updated.

The SPT generated of this graph is as shown in Fig.5 this is simple Dijkstra's algorithm since initially we have to compute SPT by that only but after this when ever weight will change then SPT is going to be calculated by proposed algorithm. Considering all the cases one by one.

Case 1. If weight of any edge is increasing and that is not the part of SPT

If weight of edge from 4 to 7 changes from 8 to 10which is not in SPT then in this case as discussed in case 1 of section IV. A. no node will be updated or traversed, just algorithm will visit all the edges to find that edge and update its weight, so it may have some time in order of O(1) in best case and O(E) in worst case and average case where E is number of edges present in that graph. And SPT is going to be remain unchanged where as Dijkstra's algorithm will take again $O(n^2)$ where n is the number of nodes in that graph.

	Dijkstra's Algorithm	Proposed Algorithm
No.Of vertices visited	$N*N=N^2$	0
No.Of edges visited	E	E
Time complexity	$O(N^2)$ in all cases	O(1) in best case O(E) in average and worst case

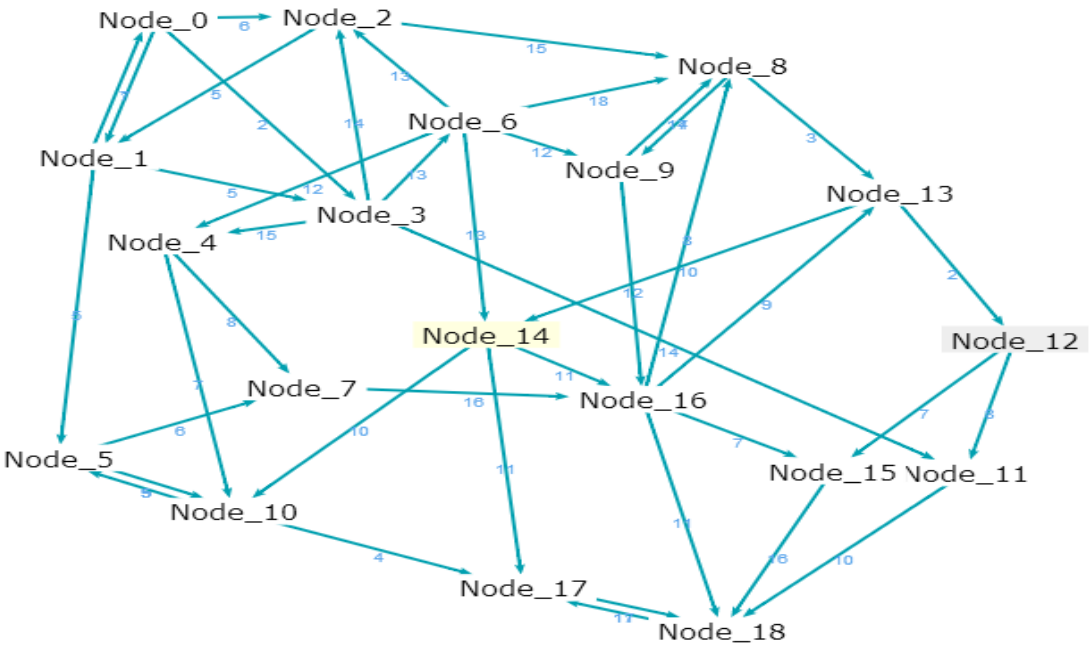

Fig 4: Small Graph taken to show correctness and working

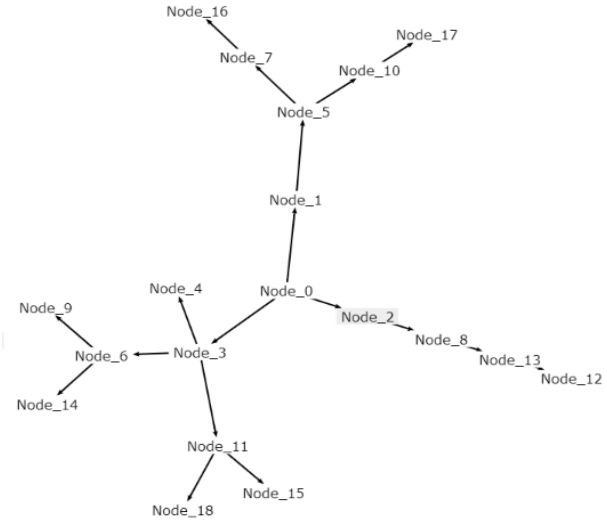

Fig 5: SPT for graph in Fig 4

Case 2. If weight of any edge is increasing and that is the part of SPT

To explain this case if weight of edge from vertex 2 to 8 changes from 15 to 50 then the vertices 8, 13, 12 which are in SPT shown in Fig 5 will get unsettled and may need to find their new parents shown in fig 4 , for this process proposed algorithm will visit nodes 2, 6, 9, 16, 8, 13 12 to make them settle down. But to find all these nodes algorithm will need to access all the Edges for 8, 13 and 12 which will take time in order of 3*E i.e. O(E), and time complexity to settle down vertices will be $O(d^b)$ where d is the depth of that sub-tree and b is the branching factor so complexity would be $O(|E|+|d^b|)$.

To show algorithms correctness updated SPT is shown in fig. 6.

In Graph shown in Fig. 4 be updated and total number of nodes traversed are 7, where as in Dijkstra's all 19 vertices will be traversed. Comparison matrix can be shown as follows.

	Dijkstra's Algorithm	Proposed Algorithm
No.Of vertices visited	$N*N=N^2$	b^d
No.Of edges visited	E	$b^d * E$
Time complexity	$O(N^2)$ in all cases	$O(b^d)$ in best , average and worst case

In the graph shown in Fig. 6 which is comparatively very small graph from practical and real life graph. This graph is having limited depth and nodes but in this also it is accessing only 7 out of 19 nodes so only 36.8% of the graph need to be traversed.

Case 3. If weight of any edge is decreasing and that is not part of SPT

If any edge which is not present in SPT let us say from vertex 3 to 2 changes its weight from 14 to 3 then, in first step only vertex 2 is needed to be checked if that is affected then need to change parent of 2 to 3, and rest of the vertex will remain as they are and the cost(distance from the source) of its sub-tree will be updated after that all other nodes will get checked which are successors of node 2, if they get affected then same process will be repeated for their successors also, until we get

further successors unchanged. This resultant SPT is shown in fig.7, to find all those vertices which are part of the subtree from that source vertex here in example it is vertex 2, algorithm will have to check all the vertices. There complexity would be O(E+N).

If we compare proposed algorithm's complexity with Dijkstra's algorithm then it would be as follows.

	Dijkstra's Algorithm	Proposed Algorithm
No.Of vertices visited	N*N=N^2	N
No.Of edges visited	E	E
Time complexity	$O(N^2)$ in all cases	O(E+N) in best , average and worst case

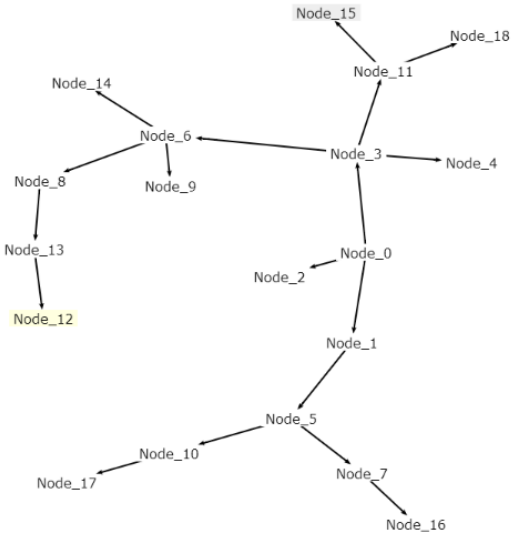

Fig 6: SPT after changing weight of edge from vertex 2 to vertex 8 from weight15 to 50.

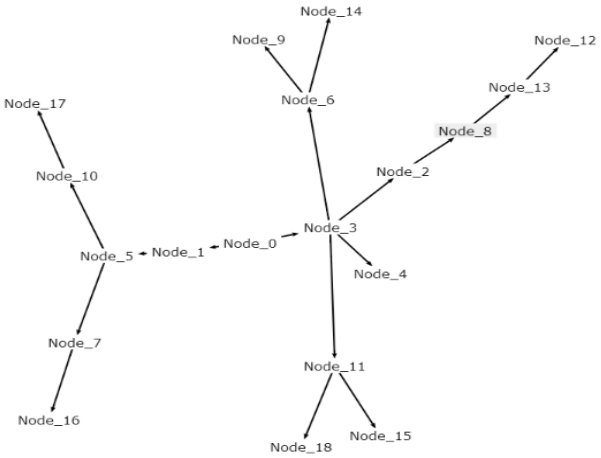

Fig 7: SPT after changing weight of edge from vertex 3 to vertex 2 from weight 14 to weight 3.

Case 4. If weight of any edge is decreasing and that is part of SPT.

This case is considered for those edges which are decreasing their weight and they are part of SPT, then proposed algorithm will just update edge weight and distance of vertices present in that sub-tree and halt but the structure of SPT will have same structure. No other changes will be done.

To find sub-tree algorithm will scan all the edges (E) and only those nodes which are in that sub-tree, so we can say that complexity is O(E).

	Dijkstra's Algorithm	Proposed Algorithm
No.Of vertices visited	N*N=N^2	Only subtree
No.Of edges visited	E	E
Time complexity	$O(N^2)$ in all cases	O(E) in best , average and worst case

Besides implementing our algorithm we have also implemented Dijkstra's SPT algorithm so that we could compare results of our algorithm and Dijkstra's algorithm when we update the graph with multiple edges. In this section we have discussed these results with multiple cases. Here we have generated a graph randomly where we have given number of vertices and edges are generated automatically to generate a Graph having random weights.

X-axis represents number of edges changed, and y-axis represents time taken to find new updated SPT by our proposed algorithm and Dijkstra's algorithm.

Graph Size	No. of changed edges	Time by Simple Dijkstra	Proposed Dynamic Algorithm
2000 nodes 5892 edges	1	1903	0
	2	1996	109
	3	1996	46
	4	2137	281
	5	1919	16
	6	2121	249
	7	1519	312
	8	2652	1310
	9	1342	94
	10	1420	156
	20	7301	6053
	30	3510	2262
	100	11731	10390

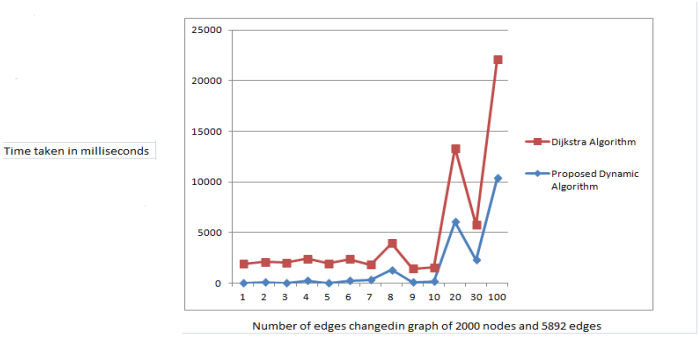

Fig 8: Comparative results with 2000 nodes and 5892 edges with mixed weight changes.

In Fig:4 the graph which is used is very dense and while changing any edge we get almost constant depth of that SPT tree, so here it may not show a high variation with different number of edges, but where this depth is high this will show a great difference in time.

5. CONCLUSION

For Dynamic Shortest Path computation some previous algorithms were there but either they were static or semi-dynamic or if fully then were fail to correctly process multiple edge weights. Proposed algorithm is easy to understand and correctly processing multiple weights. This algorithm is working efficiently for multiple weight change and even if all the conditions are given together as an input then also it is more efficient and computing SPT correctly.

As Compared with Dijkstra's algorithm this algorithm is showing less time complexity in order of O(E) in most of the cases and in some cases $O(b^d)$ where E is the number of edges present in that graph, d is depth of that affected sub-tree and b is branching factor of the graph, which shows tremendous time reduction for any dense and big network or graph.

Algorithm will show a great change and time reduction if graph will be dense and having high depth. Purpose of this study is to give time efficient algorithm for dynamic graphs where weights of the edges are changing frequently. Algorithm avoids those vertices to traverse which are not likely to be affected with any changes in edge weights. This proposed algorithm gives minimum complexity as compare to Dikstra's algorithm which has been already mentioned in previous topics with analysis.

6. ACKNOWLEDGEMENT

I take this opportunity to express my profound gratitude and deep regards to my guide (Professor R.R. Sedamkar) for his exemplary guidance, monitoring and constant encouragement throughout the course of this paper. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to my younger sister Ms. Shristi Maurya (Tata Consultancy Services), for the valuable help in implementation and moral support. I am grateful for their cooperation during the period of my assignment.

7. REFERENCES

[1] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni, "Fully Dynamic Algorithms for Maintaining Shortest Paths Trees," J. Algorithms, vol. 34, no. 2, pp. 251-281, 2000.

[2] D. Frigioni, M. Ioffreda, U. Nanni, and G. Pasquale, "Experimental Analysis of Dynamic Algorithms for the Single-Source Shortest- Path Problem," ACM J. Experimental Algorithms, vol. 3, p. 5, 1998.

[3] P. Narva´ez, K. Siu, and H. Tzeng, "New Dynamic Algorithms for Shortest Path Tree Computation," ACM Trans. Networking, vol. 8, no. 6, pp. 734-746, 2000.

[4] P. Narva´ez, K. Siu, and H. Tzeng, "New Dynamic SPT Algorithm Based on a Ball-and-String Model," ACM Trans. Networking, vol. 9, no. 6, pp. 706-718, 2001.

[5] Edward P.F. Chan and Yaya Yang,"Shortest Path Tree Computation in Dynamic Graph,"IEEE Transaction On Computers, vol 58. No.4., April 2009.

[6] G. Ramalingam and T.W. Reps, "An Incremental Algorithm for a Generalization of the Shortest-Path Problem," J. Algorithms, vol. 21, no. 2, pp. 267-305, 1996.

[7] E.W. Dijkstra, "A Note on Two Problems in Connection with Graphs," Numerical Math., vol. 1, pp. 269-271, 1959.

[8] B. Xiao, Q. Zhuge, and E.H.-M. Sha, "Efficient Algorithms forDynamic Update of Shortest Path Tree in Networking," J. Computers and Their Applications, vol. 11, no. 1, 2003.

[9] G. Ramalingam and T.W. Reps, "On the Computational Complexity of Dynamic Graph Problems," Theoretical Computer Science, vol. 158, nos. 1-2, pp. 233-277, 1996.

[10] G. Ausiello, G.F. Italiano, A. Marchetti-Spaccamela, and U. Nanni,"Incremental Algorithms for Minimal Length Paths," J. Algorithms,vol. 12, no. 4, pp. 615-638, 1991.