Journal of Theoretical and Applied Information Technology
10th June 2015. Vol.76. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

IMPROVEMENT OF SHORTEST-PATH ALGORITHMS USING SUBGRAPHS' HEURISTICS

[1]FAISAL KHAMAYSEH, [2]NABIL ARMAN

[1]Asstt Prof., Department of Information Technology, Palestine Polytechnic University

[2]Prof., Department of Computer Science & Engineering, Palestine Polytechnic University

[2] Corresponding Author

E-mail: [1]faisal@ppu.edu, [2]narman@ppu.edu

ABSTRACT

Given a graph G=(V,E,w), where V and E are finite set of vertices and edges respectively, is a directed weighted graph with weights denoted by w(e)>0 for each edge $e \in$E. P(s,t) is the shortest path between the given vertices <s> and <t> containing the least sum of edge weights on the path from <s> to <t>. Properties of the graph representation, using different matrix structures to represent the graph in normal flow and reverse representations, are considered. Based on these structures, a new algorithm determines the candidate subgraphs and prunes every subgraph that is either unreachable from the given source vertex <s> or does not lead to the given destination<t>, benefiting from the rich information inherent in the matrix and reverse matrix structure representations of the graph. The experiments were conducted using our heuristic and the conventional shortest path finding, namely Dijkstra's algorithm. Practical results are given showing considerable improvements of the proposed algorithm in performance. This improves the shortest path algorithm significantly.

Keywords: *Shortest Path, Pruning, Graph Algorithms, Candidate Subgraphs, Heuristics.*

1. INTRODUCTION

Shortest path finding problems are the most encountered problems in graph algorithms and communication network applications. Since finding shortest paths over network topology is demanding and expensive, it is worthy to consider various techniques and heuristics that can help in improving the existing algorithms. The most well-known algorithm for finding a single-source shortest path is Dijkstra's algorithm [4]. There are many attempts to improve the functionality of shortest path algorithms using different assumptions and graph representations [1]-[3] and [6]-[9]. The main scope of this paper is to introduce some heuristics to improve the performance of shortest path finding. This study addresses the value of the graph representation in both forms –the normal and the reverse matrix representations - to improve the performance of the shortest path algorithm.

The first section of this paper describes an existing technique of graph representation and how this technique works on path existence in directed unweighted graphs [1]. The second section of the paper represents the techniques of finding shortest paths in directed weighted graphs with some enhancements on some current methods. The remaining sections present our algorithm and its improvements. This work presents a new improved variation of finding single source-destination shortest path by focusing on candidate sub graphs.

Let G=(V,E,w) be a directed graph, where V is a set of vertices, E is a set of edges and w is the weight function, where w(e)>0 for each edge $e \in E$. Let each edge e has a non-negative weight. Assume <s> and <t> are given vertices where <s> and <t>$\in V$, <s> is the source vertex and <t> is the destination. The single pair source-destination shortest path is to find the path with the minimum cost sum of edges from source <s> to destination <t>.

Finding the shortest path varies in time complexity upon the constraints to be applied. Such examples are finding the single-source shortest path, single-source shortest path with the possibility of negative weights, k-shortest paths, single-pair using heuristics, all-pairs shortest paths, etc. These assumptions and constraints may require applying simple minimum spanning tree procedures to effectively find the shortest path, while other assumptions may require advanced algorithms such as Dijkstra's algorithm. Some variations and improvements based on tree structures have been

Journal of Theoretical and Applied Information Technology
10th June 2015. Vol.76. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

presented in the literature. Example of such variations is the running time based on Fibonacci-heap min-priority queue which is $O(|V|log|V|+|E|)$ assuming that w(e) is a nonnegative weight [4].

Recently, [10] attempted to improve shortest path algorithm based on search strategy by introducing a constraint function with weighted values. Reference [12] ignores the large number of irrelevant nodes during shortest path finding. Some researchers have focused more on overcoming the network structure rather than the algorithm itself. Reference [11] presented an algorithm to find the shortest path through graph partitioning. They took an advantage of road network features to improve the search. The main feature is the possibility of partitioning the graph into a set of components or clusters. They focused on simplifying the detailed graph by clustering nodes that are near each other. In the final generated graph, the search is conducted near the start of the destination of the path and among the components on the transit edges.

2. DATA STRUCTURES

The normal and reverse representations require two matrices with maximum $|V|^2$ elements to in order to represent the graph G containing |V| non-repeatable vertices [1]. For efficient implementation and to save storage, the matrix can also be represented as a linear array with |E| entries. Fig. 1 depicts graph G=(V,E,w) which is represented in the matrix structure and linear array as shown in Fig. 2 and Fig. 3 respectively. These representations were used in developing parallel algorithms for the generalized same generation queries in deductive databases [3].

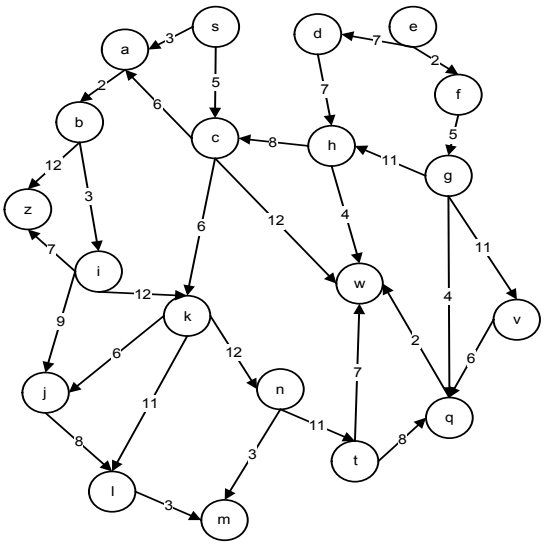

Figure 1: Directed Wighted Graph G=(V,E,W)

-	0	1	2	3	4	5	6	7	8
0	s	a	b	Z					
1				I	0,3				
2					j	l	m		
3					k	2,4			
4						2,5			
5						n	2,6		
6							t	w	
7								q	6,7
8		c	0,1						
9			3,4						
10			6,7						
11	d	h	8,1						
12			6,7						
13	e	11,0							
14		f	g	11,1					
15				7,7					
16				v	7,7				

Figure 2: Graphmatrix Representation

The main advantage of the graph matrix representation shown Fig. 2 is that it stores all paths from each node to all reachable nodes in the graph G. This representation introduces a set of benefits. Checking path existence and path links takes linear time. This matrix also shows all graph roots and paths' ends in reference to a given source vertex s. This means that any unreachable vertex from a given source is shown in the first column (the source column). Each path is represented in as a depth first search traversal order while common parts of the paths are stored only once using array indexing to avoid sub path duplications. For example, if paths $p_1=<v_1,v_2,v_i,\ldots,v_{n-1},v_n>$ and $p_2=<v_1,v_2,..,v_i,..,v_m>$ are present in the graph, then p_2 is stored in the next row of p1 starting from the column (i+1) representing the rest of the p2 as $<v_{i+1}, v_{i+2}, \ldots>$ with empty i+1 entries. Moreover, the vertex is represented only once in direct form. This leads to store the distinct subpaths by storing the first-visited nodes and recording their coordinates while subsequent subpaths that are shared in more than one path are represented by storing the coordinates' pointer to the first common revisited node. The basic advantage of this efficient graph-

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

matrix representation is to avoid duplicate representations of common subpaths.

Node	reference	Node	Reference
0,0	s	**8,1**	c
0,1	a	**8,2**	0,1
0,2	b	**9,2**	3,4
0,3	z	**10,2**	6,7
1,3	i	**11,0**	d
1,4	0,3	**11,1**	h
2,4	j	**11,2**	8,1
2,5	l	**12,2**	6,7
2,6	m	**13,0**	e
3,4	k	**13,1**	11,0
3,5	2,4	**14,1**	f
4,5	2,5	**14,2**	g
5,5	n	**14,3**	11,1
5,6	2,6	**15,3**	7,7
6,6	t	**16,3**	v
6,7	w	**16,4**	7,7
7,7	q		
7,8	6,7		

Figure 3: Linear Array Representation

The reverse matrix structure represents the graph with leaves stored first. The advantage of this representation is that it stores all paths that can reach the node from the source nodes. The reverse matrix can be constructed in the same way the main matrix is constructed. The paths p1=<v1,v2,v3,v4,v5> and p2=<v1,v2,v7,v8> that are linearly retrieved from reverse matrix indicate that the sources v5 and v8 reach the destination vertex v1.

The sizes of both used graph matrices may differ in the way of representing paths. The reason is that some parts of some represented paths may be visited earlier and being referenced later, while these nodes may explicitly appear in consecutive row entries or consecutive row and column entries if first visited. For example; in the main matrix representation, as shown in Fig. 2; if we attempt to visit the two subgraphs of source <s> in a way that subgraph rooted with <c>comes before the one rooted with <a>, then the whole part rooted with <k> in entry (3,4) is left-shifted two columns because it is previously visited as successor of node <c>. Another observation is that the number of entries in both matrices must be the same, since these entries represent the nodes and edges of the main graph.

3. WEIGHTS REPRESENTATION

The straight forward representation of the graph is to modify the matrix in Fig. 2 by adding the weights, accumulative weight and the previously utilized predecessor vertex in the same structure.

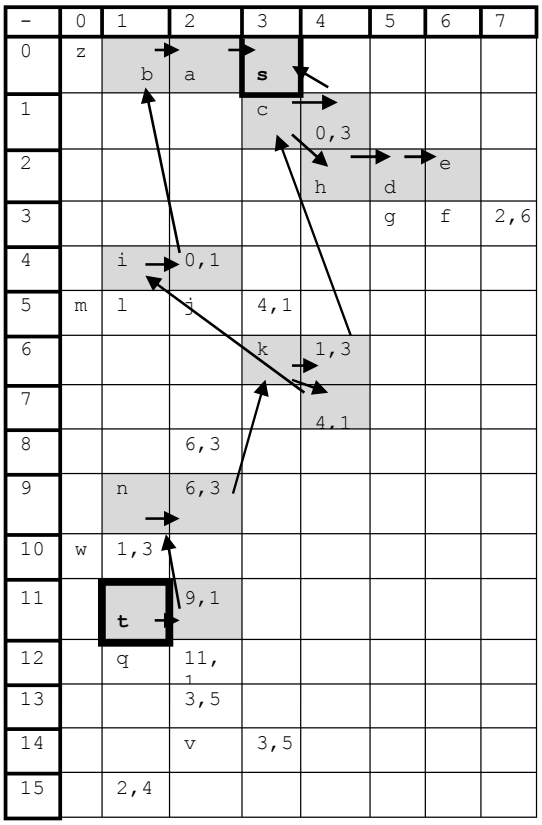

Figure 4: Reverse Matrix Representation

The structure (Vertex, Dist, Pred node) is used as a matrix entry representation as shown in Fig. 5. The advantage of this matrix is to be used while finding the shortest path down to the current vertex by adding either the current weight or by adding the weight of the new Pred node. This is possible by going through the paths stored in the weighted graph matrix represented in Fig. 5 taking into account the candidate vertices produced by marking the set of vertices V' which is accomplished by going through the reverse matrix starting from the destination vertex <t> as shown in marked (shaded) entries in Fig. 4. In this case, the distance of the path (dist) is to be updated to the minimum sum of

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Nodes	0	1	2	3	4	5	6	7	8
0	s -	a 3	b 5	z					
		s	a						
1				i 8	0,3				
				b					
2					j	l	m		
3					K 11	2,4			
					c				
4						2,5			
5						n 23	2,6		
						k			
6							t 34	w	
							n		
7								q	6,7
8		c 5	0,1						
		s							
9			3,4						
10			6,7						
11	d	h	8,1						
12			6,7						
13	e	11,0							
14		f	g	11,1					
15				7,7					
16				v	7,7				

Figure 5: Weighted Graph Matrix Representation.

weights based on the new weight corresponding to the new vertex. This ends up with the minimum sum of weights from source vertex to current vertex via the previously selected one which is named Pred Node in this procedure.

The algorithm always adds the smallest weighted candidate edge to the shortest tree keeping the shortest path being calculated from the vertices of candidate subgraph G'. As an example, the procedure goes only through the candidate marked entries as illustrated in Fig. 5 looking for the shortest path. Other nodes are excluded by the algorithm as these nodes do not lead to the destination.

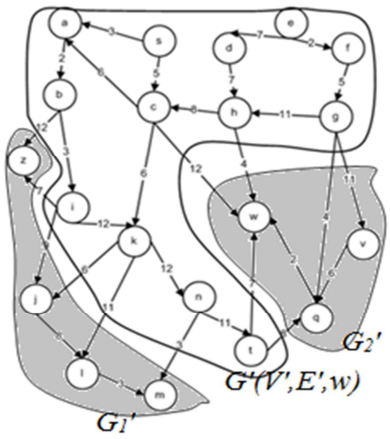

Figure 6: Candidate Subgraph G' (V',E',w)

Journal of Theoretical and Applied Information Technology
10th June 2015. Vol.76. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. PROPOSED ALGORITHM

The advantage of shortest-path algorithms and corresponding structure is apparent when the algorithm is applied on large graphs such as huge network nodes. It is not practical to use traditional algorithms to find the shortest path. It is worthy to minimize the graph and exclude the parts that do not lead to destination vertex. This optimized technique may exclude huge parts of the graph and hence saves the cost and improves the performance of the graph. The technique is summarized as:

1) Construct the matrix to represent the graph with inner structure that includes the Vertex, Dist, and Pred Node. Dist[v] maintains the minimum distance to <*v*> via Pred Node.
2) Construct the Reverse Matrix to represent the graph rooted with destinations.
3) Traverse the graph G starting by the given destination to mark all candidate nodes in the main matrix representation. This is possible using Reverse Matrix marking all candidate nodes as shown in Fig. 6. This is also possible in different ways as preferred by the programmer, e.g., copying the candidate nodes to a different reduced matrix, having a mark flag in the node structure, or by changing the weights of the excluded nodes to infinity in the main graph matrix. The preferred way is to have a 0/1 flag in a corresponding coordinate linear array representation.
4) After marking the candidate subgraph in the main matrix, and starting from the given source *s*, the algorithm adds all neighbor edges by visiting all nodes listed in the next column (breadth fashion) of the current node (*vertex*). In this case, we always accumulate the subpath weight by adding the current vertex weight to accumulated path weight (*dist*). Whenever we read the coordinates (*i,j*) of any vertex, it means that we revisit the node using another edge *e* with new weight w(e). In this case we directly jump to coordinates' pointer (*i,j*) in the main graph matrix as represented in Fig. 2 and compare the new weights and hence we keep the minimum path distance with updated predecessor nodes.

Keeping the predecessor vertices enables us to trace back the shortest path form the current vertex to the source vertex. Starting from this source vertex, the destination vertex is reached with no guarantee that the search proceeds directly to this destination. In some cases, the search will explore irrelevant parts of the graph.

Vertex No	vertex	Coordinates	Mark
0	s	<0,0>	~~0~~ 1
1	a	<0,1>	~~0~~ 1
2	b	<0,2>	~~0~~ 1
3	z	<0,3>	0
4	i	<1,3>	~~0~~ 1
5	j	<2,4>	0
6	l	<2,5>	0
7	m	<2,6>	0
8	k	<3,4>	~~0~~ 1
9	n	<5,5>	~~0~~ 1
10	t	<6,6>	~~0~~ 1
11	w	<6,7>	0
12	q	<7,7>	0
13	c	<8,1>	~~0~~ 1
14	d	<11,0>	~~0~~ 1
15	h	<11,1>	~~0~~ 1
16	e	<13,0>	~~0~~ 1
17	f	<14,1>	~~0~~ 1
18	g	<14,2>	~~0~~ 1
19	v	<16,3>	0

Figure 7: MarkMatrix

These steps assure that the shortest path works only on candidate subgraphs that form the shortest tree with leaves are possible destinations. Having one destination in account, the reverse visits minimize the candidate tree and excludes all parts that do not lead to the given destination. Hence, the algorithm ignores all unmarked tree nodes and unreachable parts as shown in shaded excluded subgraphs*G1′* and *G2′* in Fig. 8.This improved technique leads to considerable saving in real networks, such as communication and routs networks.

The final shortest path tree is then formed from the marked nodes within the indicated nodes in candidate space. The minimized shortest tree is shown in Fig. 9 as an overlap between the possible shortest tree and the candidate subgraph. The advantage of these candidate subgraphs is to

Journal of Theoretical and Applied Information Technology
10th June 2015. Vol.76. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

minimize the graph to as small as possible around the shortest path especially when source and destination vertices are specifically indicated. More precisely, we save the effort of the algorithm in finding the shortest paths in irrelevant graph parts.

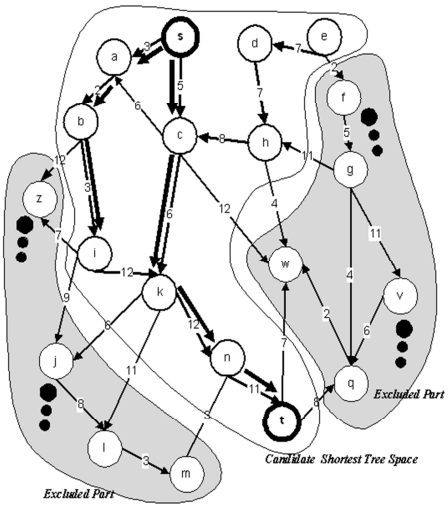

Figure 8: Candidate Shortest Tree Space

Algorithm Shortest Path Using Candidates mainly uses the matrices Reverse Matrix, Weighted Graph Matrix, and Mark Matrix in Figures 4, 5, and 7 respectively. The Reverse Matrix is generated from original unweighted directed Graph Matrix representation in Fig. 2. The two given source and destination nodes are assumed to exist in the graph G. An efficient algorithm called Path Existence Query that aids in finding the existence of the path in a directed graph from <s> to <t> is presented in ref. [1]. The algorithm proceeds by finding the candidate nodes starting from the destination node <t> visiting all predecessors towards the source node <s>. This is the main advantage of using the reverse representation in Fig. 4. Marking nodes is possible by updating Mark Matrix as shown in Fig. 7. It starts by initializing marked vertices to unmark tag equals to zero. Then the algorithm finds the shortest path among the marked nodes as of Weighted Graph Matrix representation. The function keeps in each entry of the main values; Vertex, Dist, and Pred Node. These values are updated as the procedure proceeds. Specifically, it starts from the source node <s> checking the marked nodes and calculating the path distance horizontally then diagonally. The function stores in Dist (when first visit the node) the accumulated weight up to the Vertex, by adding the vertex weight from its predecessor Pred Node. The function keeps updating the Dist whenever reads a coordinates of revisited node. Similar to updated phases of Dijkstra's algorithm [4], [5], it compares the last calculated weight with the new weight keeping the minimum value and the corresponding predecessor node. This assures the objective of finding the shortest path and can be done by updating and applying any known shortest path algorithm with slight updates.

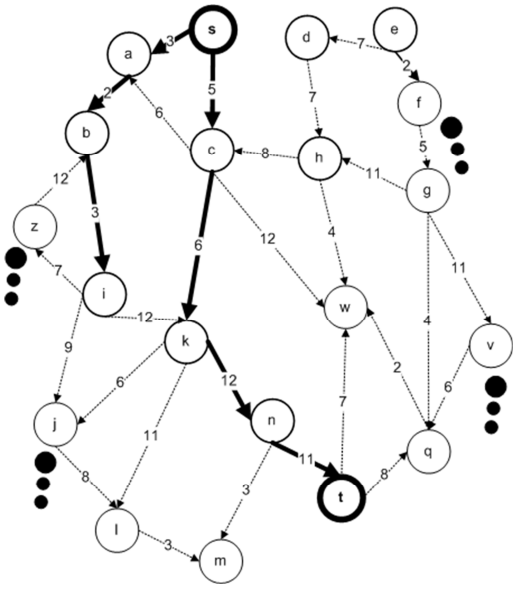

Figure 9: Shortest Tree

Shortest Path Using Subgraph's Heuristics

The proposed algorithm benefits from the real candidate subgraphs and ignores other irrelevant graph parts looking for the shortest Path as illustrated in following algorithm. On the other hand, it finds the shortest path based on the exclusion of all nodes in the graph that do not lead to destination. This efficient procedure saves much work comparing to the functionality of known conventional algorithms, since the later works exhaustively in the whole graph structure. The key procedure in the algorithm it to structure the graph baths in the matrices where candidate paths are every path P(s,t) having source vertex <s> and destination <t> as the P's head and end respectively. This implies that every sub bath P'(s',t') where s'

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

and t' belong to the path P' is also included in the candidate subgraph G'.

```
Algorithm Shortest_Path_Using_Candidates(Graph, Mark, Reverse_Matrix)
{
  initialize Mark[i] to 0
  Node=t
  Mark[Node]=1
  for every vertex next to Node in Reverse_Matrix
    if vertex != coordinates_pointer
       Node=vertex
    else
       Node=ReverseMatrix[coordinates_pointer]
       Mark[Node]=1
   end if
 end for
 dist= FindShortest(GraphMatrix, Mark, s)
}
```

```
Function FindShortest(GraphMatrix,Mark,s)
{
 for each vertex v in GraphMatrix
      Initialize dist[v] = infinity
      pred[v] = undefined
 end for
dist[s] := 0
MarkQ = set of Marked nodes in GraphMatrix
ordered as of depth first search visits.
while MarkQ is not empty and u != t
      u= vertex in MarkQ with smallest distance in
      dist[ ];
      remove u from MarkQ
      if (u == t ||dist[u] == infinity)  then break
      end if
     for each neighbor v of u and v is in MarkQ
       if v is coordinate pointer <a,b>
            v=GraphMatrix[a,b]
       end if
       p = dist[u] + dist_between(u, v)
       if p<dist[v]
            dist[v] = p
            pred[v] = u
            update  v in MarkQ
       end if
     end for
end while
return dist
}
```

5. PERFORMANCE IMPROVEMENTS AND MEASUREMENTS

An algorithm that finds the shortest path P(s,t) between two given vertices <s> and <t> in a directed weighted graph G(V,E,w) is presented.It clearly determines the candidate subgraph G'(V',E',w) corresponding to destination <t> ignoring all subgraphs that do not lead to destination from any source. This obvious improvement excludes all disconnected parts that may form large parts of graphs and this lowers the cost. Each shortest path requires $O((|V|+|E|)\log |V|)$ using traditional algorithms making the cost extremely high.

The proposed algorithm introduces more improvements comparing with some late studies such as the improvements introduced in [7]. Moreover, our algorithm works in better and improved performance on sparse and dense networks.

The experimental phase provides evidence that the proposed heuristic outperforms the conventional algorithms. The performance of the algorithm is compared with that of the conventional procedure and shows a considerable cost saving in random generated graphs with different sizes range from 100 to 500 nodes. Savings in performance occur in dense graphs and more in sparse ones in most of the trials. Table 1 shows the performance saving ratios as a result of the experiment.

Figures 10 and 11 show the average performance of applying the improved Dijkstra's algorithm on set of random graphs with different density degrees. This shows that the proposed algorithm

Table 1: Saving Ratio Of Performance In Sparse And Dense Graphs

Nodes	Sparse (%)	Dense (%)
100	0.8907436	0.7525469
150	0.646433	0.6867926
200	0.5183573	0.6371958
250	0.5245072	0.6202232
300	0.5218704	0.6107841
350	0.5804231	0.4876443
400	0.5880742	0.4801536
450	0.5670171	0.4614253
500	0.6200705	0.4900764

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

outperforms the improved procedures such as improved Dijkstra's algorithm.

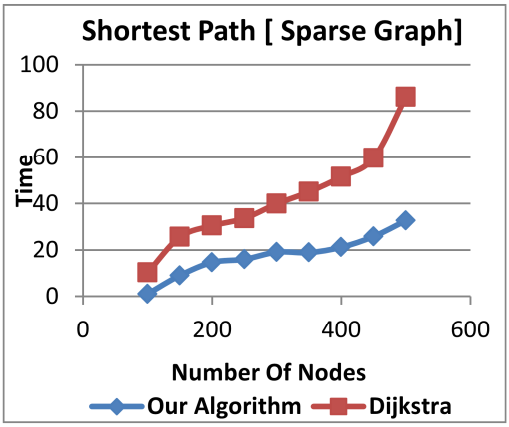

Figure10: Performance Of Proposed Algorithm Vs. Improved Dijkstra's Algorithm On Sparse Graph

6. CONCLUSIONS

An efficient and improved heuristic algorithm for finding shortest paths between a given source <s> and destination <t> using candidate subgraph in a weighted directed graph G(V,E,w) with weights as a function of |w(e)| is presented. In the practical phase, the algorithm outperforms the

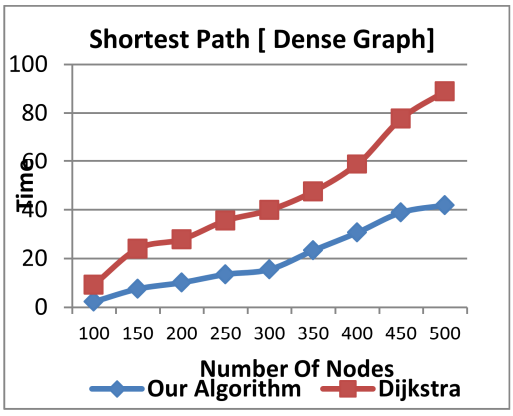

Figure 11: Performance Of Proposed Algorithm Vs. Improved Dijkstra's Algorithm On Dense Graph

performance of improved Dijkstra's algorithm. It shows obvious improved performance in set of random general applied graphs. As a new heuristic algorithm, the complexity will always be bounded by the complexity of known algorithms, i.e., it will not exceed O((|V|+|E|)log |V|) for each source $<s_i>$ and each destination $<t_i>$ in graph G.

The candidate nodes are identified as all nodes exist in paths lead to destination node <t>. The proposed heuristic determines the candidate subgraphs by marking the ancestors of the destination node using reverse matrix representation to enable backward traversals. Comparing with current conventional algorithms, this heuristic improves the shortest path algorithm significantly.

ACKNOWLEDGMENTS

This research is funded by The Scientific Research Council, Ministry of Education and Higher Education, State of Palestine under a project number of 01/12/2013, and Palestine Polytechnic University. The authors would like to thank the research assistants Ms. Walaa Naser Idin and Ms. Salma Dirbashi for their help in implementing the algorithms.

REFERENCES:

[1] N. Arman. An Efficient Algorithm for Checking Path Existence between Graph Vertices. Proceedings of the 6th International Arab Conference on Information Technology (ACIT'2005), pp. 471-476, December 6-8, 2005, Al-Isra Private University, Amman, Jordan, 2005.

[2] H. N. Djidjev, G. E. Pantziou, and C. D. Zaroliagis. Improved Algorithms for Dynamic Shortest Paths. Algorithmica, 2000 28: 367–389.

[3] N. Arman. Parallel Algorithms for the Generalized Same Generation Query in Deductive Databases, Journal of Digital Information Management: 4(3), 2006, 192-196, ISSN 0972-72.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein.. Dijkstra's Algorithm. Introduction to Algorithms (Second ed.). Section 24.3: pp. 595–601. MIT Press and McGraw-Hill. 2001. ISBN 0-262-03293-7.

[5] E. W. Dijkstra. A note on Two Problems in Connexion with Graphs. Numerische Mathematik 1, 1959. 269–271.

[6] J. B. Orlin, K. Kamesh Madduri, K. Subramani, and M. Williamson. 2010. A faster algorithm for the single source shortest path problem with few distinct positive lengths. J.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

of Discrete Algorithms 8(2), June 2010, 189-198.

[7] L. Xiao, L. Chen, and J. Xiao. A new algorithm for shortest path problem in large-scale graph. Appl. Math, 6(3), 2012, 657-663.

[8] F. Zhang, A. Qiu, and Q. Li. Improve on Dijkstra Shortest Path Algorithm for Huge Data. Chinese academy of surveying and mapping: China, 2005.

[9] F. Khamayseh and N. Arman. An Efficient Heuristic Shortest Path Algorithm Using Candidate Subgraphs. International Conference on Intelligent Systems and Applications. Hammamet, Tunisia. 22-24 March 2014.

[10] F. Khamayseh and N. Arman. An Efficient Multiple Sources Single-Destination (MSSD) Heuristic Algorithm Using Nodes Exclusions. International Journal of Soft Computing. IJSC 10(3), 2015.

[11] F. Simekand I. Simecek. Improvement of Shortest Path Algorithms through Graph Partitioning. International Conference Presentation of Mathematics. Liberec, Czech Republic, 2011.

[12] Y. Huang, Q. Yi, and M. Shi. An Improved Dijkstra Shortest Path Algorithm. Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering ICCSEE-2013. Hangzhou, China, March 2013. Atlantis Press, 2013: 226-229.