International Journal of Soft Computing 10 (5): 301-306, 2015
ISSN: 1816-9503

An Efficient Multiple Sources Single-Destination (MSSD) Heuristic Algorithm Using Nodes Exclusions

[1]Faisal Khamayseh and [2]Nabil Arman
[1]Department of Information Technology,
[2]Department of Computer Science and Engineering,
College of Information Technology and Computer Engineering,
Palestine Polytechnic University, Hebron, Palestine

Abstract: The problem of identifying the best paths between given set of nodes and a given single-destination in a graph of vertices is commonly referred to as network multiple sourcessingle-destination problems. In real life researchers always find themselves in a critical situation in which they seek the nearest set of related points such as the urgent need for fire stations. This study describes, the problem and proposesan algorithm for finding the shortest paths between the set of sources $<s_i>$ and a single-destination <t> given that $<s_i>$ and <t>ϵ weighted graph G(V, E, w) with vertex set V and arc set E associated with nonnegative real valued weight. An efficient algorithm is developed based on different graph representations. The proposed heuristic determines a candidate subgraph G' and excludes all nodes that do not lead to destination. The proposed algorithm improves partially the performance of improved traditional shortest path algorithms, i.e., Dijkstra' algorithm. This is shown obviously by applying the algorithm on set of random graphs.

Key words: Shortest path, communication network, graph, multiple sources single-destination, candidate subgraphs, node exclusions

INTRODUCTION

Finding shortest or fastest paths in real network is always demanding. In real life researchers may find ourselves in a critical situation that researchers urgently seek the nearest set of related points such as the fire stations in the area. Many contemporary applications benefit from efficient networking topologies such as active learning space network. This requires putting together the knowledge space and learning content space in active and real time content delivery in a form of directed weighted graph (Khamayseh *et al*., 2009). In fact, the presence of large street networks in towns and residential areas increases the demand on urgently finding the best paths. This means that allocating services such as ambulance and civil defense services in the area around a specific accident point is extremely important as shown in Fig. 1.

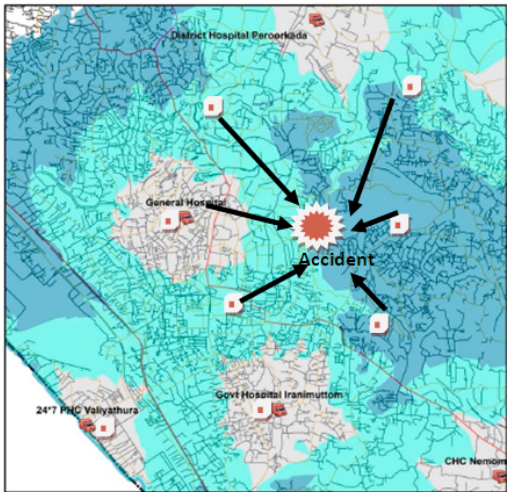

Fig. 1: Application of multiple shortest paths

This study focuses on the Multiple Sources, Single-Destination (MSSD) network problem representing the set of given sources and a given single-destination in graph G. An important method applied in this study represents the graph in several efficient structures and then traversing the graph using these structures to find the set of shortest paths. An efficient heuristic MSSD algorithm is proposed to find the shortest paths towards a single-destination. The main objective of MSSD is that it saves time and cost by excluding none related nodes, comparing to the way of applying traditional shortest path algorithm such as modified Dijkstra's algorithm. Other than finding the multiple sources single-destination

Corresponding Author: Nabil Arman, Department of Computer Science and Engineering, College of Information Technology and Computer Engineering, Palestine Polytechnic University, Hebron, Palestine

shortest paths, the new algorithm finds also the single source single-destination shortest path using the same approach. To find a single-source single-destination or multiple source multiple-destination one may refer to the known algorithms such as Dijkstra's and Floyd algorithms with some variations (Kumar and Kaur, 2011; Cormen *et al*., 2001; Dijkstra, 1959) and some improvements on related existent techniques (Arman, 2005a; Orlin *et al*., 2010; Shibuya, 1999; Xiao *et al*., 2012; Khamayseh and Arman, 2014a, b).

Given a directed graph G(V, E, w), where V is the set of nodes, E is the set of directed edges labeled with weights as a function of |w(e)|. For a given destination $<t>\in V$ and set of sources $\{<s_1>, <s_2>, ..., <s_n>\}$ for each $<s_i>\in V$, the Multiple Source Single-Destination algorithm is to find the shortest paths from each given $<s_i>$ to a single given destination <t> as depicted in Fig. 2.

There are different shortest path algorithms applied on different graphs with various constraints and conditions. Examples include finding the single-source shortest path in a weighted graph, single-source shortest path with the possibility of negative weights, k-shortest paths, shortest path in unweighted graph, single-pair using heuristics all-pairs shortest paths, etc. These graphs with different constraints and assumptions may require applying simple minimum spanning tree procedures to effectively find the shortest path. Other assumptions may require advanced algorithms such as Dijkstra's and Floyd algorithm with some variations. Some improvements on applying the existent algorithms based on tree structures have been presented by Cormen *et al*. (2001). Example of such variations is the running time based on Fibonacci-heap min-priority queue which is calculated to be $O(|V|\log|V|+|E|)$ assuming that $w(e)\geq 0$ (Cormen *et al*., 2001).

In this study, the applied approach represents the directed graph in a matrix structure with entries be the links between nodes starting from the graph roots. Then, the graph is reversely represented using the same structure starting from the graph destinations with entries be the reverse links. The proposed algorithm scans both matrices starting from the reverse one to limit the candidate graph parts to be used in finding the source-destination paths.

GRAPH REPRESENTATION

After studying several alternative data structures and graph representations, researcher represents the structure in several matrices that efficiently represent the given graph as introduced by Kumar and Kaur (2011). This existing technique helps in representations and the path existence in unweighted graph. To represent a graph G = (V, E, w) consisting |V| none repeatable nodes, two matrices with maximum $|V|^2$ entries are needed with matrix entries |E| equal exactly the number of arcs in the general each graph. The first one is to represent the graph in normal form rooted with sources while the other is to represent the same graph in reverse-traverse with destinations are the graph roots (Arman, 2005a-c). These representations are shown in Fig. 3 and 4. Figure 4 shows the graph as linear array with entries representing the nodes and their coordinates in the graph matrix representation.

The graph paths are stored in Graph Matrix Representation as paths starting with graph roots from left to right with sub paths (multi out degree nodes) extending the current node in the consecutive rows. The

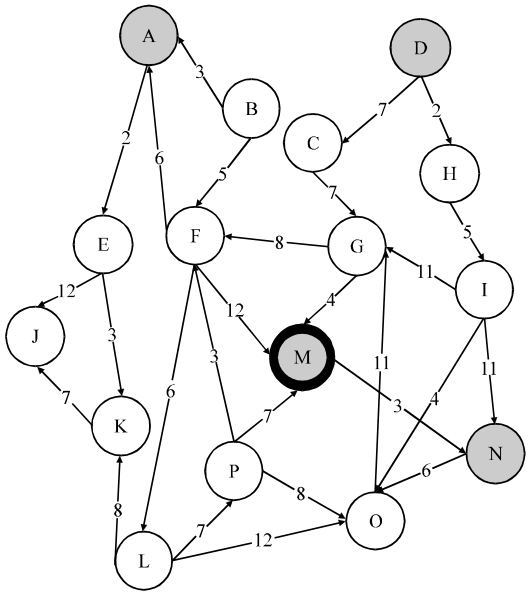

Fig. 2: Random general directed graph G(V, E, w)

-	0	1	2	3	4	5	6	7	8
0	A	E	J						
1			K	0, 2					
2	B	0, 0							
3		F	0, 0						
4			L	1, 2					
5				P	3, 1				
6					M	N	O	G	3, 1
7									6, 4
8					6, 6				
9				6, 6					
10			6, 4						
11	D	C	6, 7						
12		H	I	6, 7					
13				6, 6					
14				6, 5					

Fig. 3: Graph matrix representation

0, 0	0, 1	0, 2	1, 2	1, 3	2, 0	2, 1	3, 1	3, 2	4, 2	4, 3	5, 3	5, 4	6, 4	6, 5	6, 6
A	E	J	K	0, 2	B	0	F	0, 0	L	1, 2	P	3, 1	M	N	O

6, 7	6, 8	7, 8	8, 4	9, 3	10, 2	11, 0	11, 1	11, 2	12, 1	12, 2	12, 3	13, 3	14, 3
G	3, 1	6, 4	6, 6	6, 6	6, 4	D	C	6, 7	H	I	6, 7	6, 6	6, 5

Fig. 4: Linear array representation

-	0	1	2	3	4	5	6	7	8	9	10
0	J	E	A	B							
1				F	0, 3						
2					G	C	D				
3						I	H	2, 6			
4						O	3, 5				
5							N	M	2, 4		
6									1, 3		
7									P	L	1, 3
8								4, 6			
9							7, 8				
10							7, 9				
11					7, 8						
12		K	0, 1								
13			7, 9								

Fig. 5: Reverse matrix representation

advantage of this representation is that it stores all graph paths in the order of depth first search traversal showing clearly all subpaths. In this way, each node name/number is shown only once while all revisited common nodes are shown in reference pointer coordinates (i, j) based on matrix indexing to avoid subpath duplication. The matrix also shows the graph roots and paths' ends enabling counting the path nodes in constant time. As an example of path representations of graph G, if path $p_1 = <v_1, v_2, v_i, ..., v_{n-1}, v_n>$ and $p_2 = <v_1, v_2, ..., v_i, ..., v_m>$ exist in the graph (branching after node $<v_i>$), then p_2 is stored in the next row of p_1 starting from the column (i+1) to represent the rest of p_2 as $<v_{i+1}, v_{i+2}, ...>$ with all entries on the left of $<v_{i+1}>$ are empty. The linear array representation stores the index of all nodes and common nodes as stored in the main matrix representation.

The applied technique requires having the graph G represented in reverse structure. That is, the nodes are stores in the reverse matrix using the same construction technique but with paths start with destinations (as roots) towards the paths' sources as end nodes considering only the weighted reverse edges. The reversed-paths $p_1 = <v_1, v_2, v_3, v_4, v_5>$ and $p_2 = <v_1, v_2, v_7, v_8>$ that are linearly retrieved from reverse matrix mean that the source nodes v_5 and v_8 reach the destination node v_1. The advantage of this representation is that it stores all paths that can reach the given node from the given source nodes.

The sizes of both matrices may differ depending on the depth first search traversing technique. This means that some parts of some paths may be visited earlier and being referenced later. These paths may explicitly appear in consecutive row entries and also may appearin some row and column entries if first visited. It is required that the number of entries in both matrices must be the same, since these entries represent the nodes and edges of the main given graph.

The weighted graph can also be represented using the same representation technique. Each link is represented by associating the node name with its weight required to reach it from its direct predecessor. This representation requires that each entry contains three main parts (node name, weight, predecessor node). The advantage of this structure is to keep modifying the accumulated weight and the corresponding predecessor node name as required by the applied shortest path finder such as Dijkstra's. Such, technique assures the minimum sum of weights from a selected source via the predecessor node and neglecting all nodes out of the candidate subgraph. The candidate set of nodes are determined by marking candidate nodes using reverse traversing. Figure 5 shows the graph rooted by destinations. It helps

finding and marking the candidate nodes lead to destination node starting from destination node visiting all predecessor nodes as backward fashion. For example, node M is visited by G, F and P, node P is visited by L and F, node F is visited by B and G and so on. At the end, researchers find that some nodes are not visited for example, nodes A, E, J and K. The primarily advantage of marking nodes using reverse traversing is to exclude all nodes that do not lead to destination and hence save searching time in these nodes or subgraphs.

MULTIPLE SOURCES SINGLE-DESTINATION SHORTEST PATHS (MSSD)

In real hugenetwork topologies and in real life communication applications, the need for more than one shortest path leading to single-destination is demanding. The primarily procedure is minimizing all subgraphs that do not lead to destination from some selected sources or vice versa. The traditional algorithms do not satisfy this optimization. The new Heuristic algorithm benefits from the representation of the graph in different structures and by updating the traditional shortest path techniques. The general steps are:

- Constructing the main matrix to represent the graph structure where each entry contains the node name, weight and predecessor node. This structure represents all the graph paths with node weights
- Constructing the reverse matrix representing the graph rooted by destinations. This structure depicts all paths lead to destination. The advantage of this structure is the ability of marking all candidate nodes lead to destination and discarding all irrelevant ones. This step requires having the mark matrix containing the candidate nodes. Traversing the graph based on reverse structure results in new limited graph G'(V', E', w) as shown in Fig. 6 and Table 1
- Find shortest paths from the given sources $<s_i>$ towards the destination <t>. For each source $<s_i>$, the algorithm adds all neighbor edges (in breadth fashion) by visiting all nodes listed in the next column of the current node. For each current node, researchers always accumulate the subpath weight by adding the current node weight to accumulated path weight (dist). Revisiting the node using new edge e means researchers encounter node coordinates (i, j) with new weight w(e). In this case, researchers jump to node pointer with coordinates (i, j) in the main graph matrix and compare the new weights in order to keep the minimum distance and the corresponding predecessor nodes

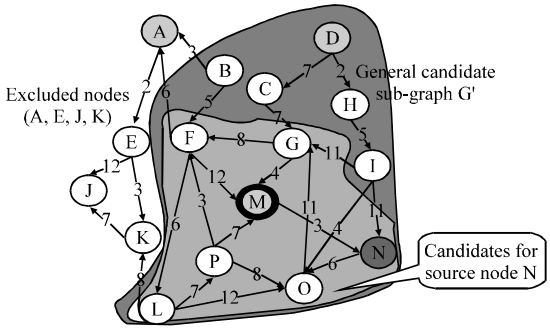

Fig. 6: Candidate nodes graph

Table 1: Candidate nodes matrix

Vertex No.	Vertices	Coordinates
0	B	<2, 0>
1	C	<11, 1>
2	D	<11, 0>
3	F	<3, 1>
4	G	<6, 7>
5	H	<12, 1>
6	I	<12, 2>
7	K	<4, 2>
8	L	<6, 4>
9	M	<6, 5>
10	N	<6, 6>
11	O	<5, 3>

The excludedgraph nodes are:{<A>, <E>, <J>, <K>}

MULTIPLE SOURCES SINGLE-DESTINATION SHORTEST PATH (MSSD) ALGORITHM

The algorithm finds each shortest path as structured in the main weighted graph matrix by excluding all irrelevant nodes and by keep checking the current node in the marked candidate nodes. Candidate nodes are determined from the generated reverse matrix in linear time. The path existence between each source $<s_i>$ to destination <t> can be determined in linear time by applying efficient algorithm called path existence query as presented by Arman (2005a-c).

The algorithm finds the shortest path between each source $<s_i>$ to destination <t> among the marked nodes only. The algorithm keeps updating entries using values (Vertex, Dist and PredNode). The function stores in Distentry the accumulated weight up to the current Vertex, by adding the vertex weight from its predecessor PredNode. The function continues updating the distentry whenever reads a coordinates pointer of the revisited node. Using the updated procedure of Dijkstra's technique (Xiao *et al*., 2012), the algorithm keeps the minimum weight and the corresponding nodes by comparing the last calculated weight with the new weight.

Multiple Sources Single-destination Shortest Path algorithm using node exclusions: The algorithm finds the shortest path based on discarding all nodes in the graph that do not lead to destination. This efficient procedure may save much work comparing to the functionality of known algorithms.

MultipleSources_Single_Destination_Shortest_Path (Graph, Mark, Reverse_Matrix):

```
    {
    initialize Mark to empty
          begin at destination vertex t in Reverse_Matrix
        Node = t
        Add node to Mark
        for every vertex next to Node in Reverse_Matrix
              //vertex is predecessor to node in G
              if vertex ! = coordinates_pointer
                    Node = vertex
              else
                    Node = Reverse Matrix[coordinates_pointer]
        add node to Mark
    endfor
     for every source node s
       dist = FindShortest(GraphMatrix, Mark, s)
    endfor
    }
FindShortest(GraphMatrix, Mark, s)//no need for destination , it is needed
in Mark matrix
{
for each vertex v in GraphMatrix          // Initializations
    dist[v] = infinity;                    // Unknown distance function from
                                              source to v
   pred[v] = undefined;      // Predecessor node in the node in optimal path
end for
dist[s] : = 0;                             // Distance from source to destination
            // All marked nodes in Mark matrix arestill not processed. Put
               them in MarkQ
MarkQ = set of Marked nodes in GraphMatrix ordered as of depth first search
visits
while MarkQ is not empty and u != t://destination is not reached
    u = vertex in MarkQ with smallest distance in dist[];     // starting
    from s
    remove u from MarkQ;
     if (u == t || dist[u] == infinity)     //end while, because destination is
    reached and
            break ;          // remaining vertices are inaccessible from
          source or unmarked
    end if
for eachneighbor v of u and v is in MarkQ     // where v has not yet been
removed from Q.
    if v is coordinate pointer <a,b>
          v = GraphMatrix[a,b]     // cross-fetch previously visited and
          marked vertex
    endif
    p = dist[u]+dist_between(u, v);
if p<dist[v]:                              // choose the best subpath
          dist[v] = p;
          pred[v] = u;
    update v in MarkQ;              // Reorder v in the Queue
    end if
    end for
end while
return dist;
}
```

CONCLUSION

For a given weighted graph G(V, E, w) with weights as a function of |w(e)|, an efficient and improved algorithm for finding shortest paths between a set of sources $<s_i>$ and single-destination <t> in a weighted directed graph G is presented. In the practical phase, the algorithm outperforms the performance of improved Dijkstra's algorithm. As a 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 $<s_i>$.

This study discussed the procedure offinding shortest paths between source $<s_i>$ and a given single-destination <t> using candidate subgraphs. The set of candidate nodes is determined as all nodes exist in paths lead to destination <t>.

ACKNOWLEDGEMENT

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.

REFERENCES

Arman, N., 2005a. Graph representation comparative study. Proceedings of the International Conference on Foundations of Computer Science, June 27-30, 2005, Las Vegas, USA., pp: 1-5.

Arman, N., 2005b. Graph representation: Comparative study and performance evaluation. Inform. Technol. J., 4: 465-468.

Arman, N., 2005c. An efficient algorithm for checking path existence between graph vertices. Proceedings of the 6th International Arab Conference on Information Technology, December 6-8, 2005, Amman, Jordan, pp: 471-476.

Cormen, T.H., C.E. Leiserson, R.L. Rivest and C. Stein, 2001. Introduction to Algorithms. 2nd Edn., MIT Press, Cambridge, MA., USA., ISBN-13: 978-0262032933, pp: 595-601.

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische Mathematik, 1: 269-271.

Khamayseh, F. and N. Arman, 2014a. An efficient heuristic shortest path algorithm using candidate subgraphs. Proceedings of the International Conference on Intelligent Systems and Applications, March 22-24, 2014, Hammamet, Tunisia, pp: 1-5.

Khamayseh, F. and N. Arman, 2014b. Using candidate subgraphs to improve single-source single-destination shortest path algorithms. SYLWAN J., 158: 303-312.

Khamayseh, F., A.S.E. Ahmed and L.M. El-Fangary, 2009. Evaluation of learner achievement within Active Multimodal Presentation (AMP) and static e-learning modes. Egypt. Comput. Sci. J., Vol. 31, No. 3.

Kumar, A. and M. Kaur, 2011. A new algorithm for solving shortest path problem on a network with imprecise edge weight. Applic. Applied Math., 6: 602-619.

Orlin, J.B., K. Madduri, K. Subramani and M. Williamson, 2010. A faster algorithm for the single source shortest path problem with few distinct positive lengths. J. Discrete Algorithms, 8: 189-198.

Shibuya, T., 1999. Computing the n X m Shortest Paths Efficiently. In: Algorithm Engineering and Experimentation, Goodrich, M.T. and C.C. McGeoch (Eds.). Springer Science and Business Media, New York, pp: 210-225.

Xiao, L., L. Chen and J. Xiao, 2012. A new algorithm for shortest path problem in large-scale graph. Applied Math., 6: 657-663.