An Efficient Heuristic Shortest Path Algorithm Using Candidate Subgraphs

Faisal Khamayseh
Department of Information Technology
College of Information Technology and Computer Engineering
Palestine Polytechnic University
Hebron, Palestine
faisal@ppu.edu

Nabil Arman*
Department of Computer Science and Engineering
College of Information Technology and Computer Engineering
Palestine Polytechnic University
Hebron, Palestine
narman@ppu.edu

*Corresponding Author

***Abstract*— *Given a graph G=(V,E,w), where Vand E are finite set of vertices and edges respectively, in a directed weighted graph G with weights denoted by w(e)>0 for each edge e∈ E. Given two vertices <s> and <t>∈V, P(s,t) is the shortest path between <s> and <t> containing the least sum of edge-weights on the path from <s> to <t>. The properties of the graph representation, using matrix structures to represent the graph in normal flow and reverse representations, are considered. Based on these structures, the 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 structure representations of the graph.*

***Index Terms*— Shortest path, pruning, graph algorithms.**

I. Introduction

Shortest pathfinding problems are the most encountered problems in graph algorithms and communication network applications. Since finding shortest paths over network topology is 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-shortest path is Dijkstra's algorithm [3]. There are many attempts to improve the functionality of shortest path algorithms using different assumptions and graph representations [1,4,5,6]. This paper 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 parts of the graph.

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 $<s>$ to $<t>$.

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 presented in the literature [3]. Example of such variations is the running time based on Fibonacci-heap min-priority queue which is $\boldsymbol{O}(|V|\log|V|+|E|)$ assuming that w(e) is a nonnegative weight [3].

II. Data Structure

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

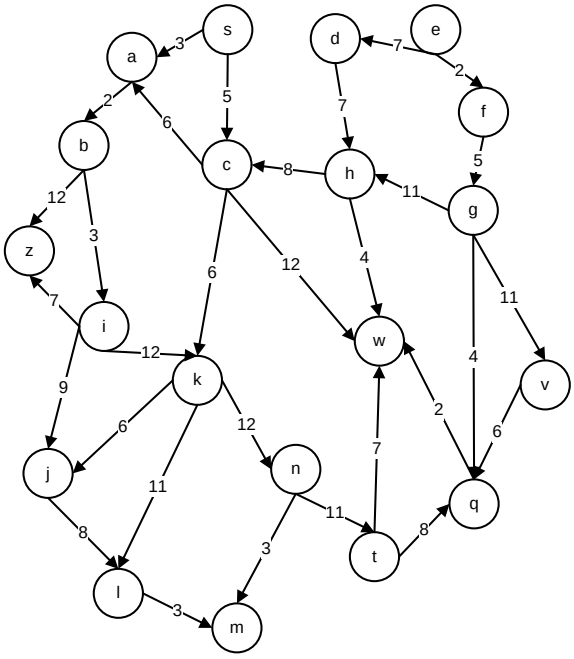

Figure 1: Directed graph

-	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: Graph Matrix Representation

0,0	0,1	0,2	0,3	1,3	1,4	2,4	2,5	2,6	3,4	3,5	4,5	5,5	5,6	6,6	6,7	7,7	7,8
s	a	b	z	i	0,3	j	l	m	k	2,4	2,5	n	2,6	t	w	q	6,7

8,1	8,2	9,2	10,2	11,0	11,1	11,2	12,2	13,0	13,1	14,1	14,2	14,3	15,3	16,3	16,4
c	0,1	3,4	6,7	d	h	8,1	6,7	e	11,0	f	g	11,1	7,7	v	7,7

Figure 3: Linear Array Representation

The main advantage of the graph matrix representation is that it stores all paths from each node to all reachable nodes in the graph. This representation introduces a set of benefits. It takes a linear time to check the path existence and also the number of path links. 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 subpath duplications. For example, if paths p_1=<v_1,v_2,v_i, …,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 p_1 starting from the column ($i+1$) representing the rest of the p_2 as <v_{i+1}, v_{i+2}, …> 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 representation is to avoid duplicate representations of common subpaths.

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 matrixis constructed. The 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 sourcesv_5 and v_8 reach the destination vertex v_1.

Note that the sizes of both matrices may differ. The reason is that some parts of some paths may be visited earlier and being referenced later, while these nodes may explicitly appear inconsecutive row entries or consecutive row and column entries if first visited. For example; in the main matrix representation, as shown in Figure 2 and Figure 3; 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 the number of entries in both matrices must be the same, since these entries represent the nodes and edges of the main graph.

III. Weights representation

The straightforward representation of the graph is to modify the matrix in Figure 2 by adding the weights, accumulative weight and the previously utilized predecessor vertex in the same structure. This representation, the structure (*Vertex, Dist, Pred Node*) is used as a matrix entry as shown in Figure 5. The advantage of this matrix is to find 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 Figure 5 taking into account the candidate vertices produced by marking the set of vertices *V'* by going through the reverse matrix starting from the destination vertex <*t*> as shown in marked (shaded) entries in Figure 4. In this case the path weight (dist) is to be updated to the minimum sum of weights based on the new weight via the new vertex. This ends up with the minimum sum of weights from source vertex to current vertex via the previously selected one named *Pred Node*. 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. As an example, the procedure goes through only the candidate marked entries as illustrated in Figure 5 finding the shortest path.

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

Figure4: ReverseMatrix representation

IV. Finding Shortest Path Using Candidate Subgraphs

According to the increase in network nodes and the huge graph structures, 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 performance.

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. 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*, tha 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 and compare the new weights and hence we keep the minimum path distance with updated predecessor nodes.

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 Figure 2. The two given source and destination nodes are assumed to exist in 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 [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 Figure 4. Marking nodes is possible by updating Mark Matrix as shown in Figure 6. It starts by initializing marked vertices to unmark

-	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.

Each matrix entry corresponds to: [`Vertex, Distance, Predecessor`]

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 [3], 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.

Algorithm Shortest PathUsing Candidates

The algorithm in Figure 7 finds the shortest path based on the exclusion of 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.

Marked Vertices	s	a	b	i	k	n	t	c	d	h	e	f	g	…

Figure 6: Mark Matrix

```
Algorithm Shortest_Path_Using_Candidates(Graph, Mark, Reverse_Matrix)
{
      initialize Mark[i] to 0
      begin at destination vertex t in Reverse_Matrix
      Node=t
      Mark[Node]=1
      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]
      Mark[Node]=1
        endfor
      dist= FindShortest(GraphMatrix, Mark, s)// returns destination of shortest path for source s
}
```

Figure 7: Algorithm Shortest Path Using Candidates

V. CONCLUSIONS

In this paper, an efficient and improved algorithm for finding shortest path between a given source and destination in a weighted directed graph is presented. The algorithm benefits from representing the graph in a matrix with revisited nodes being represented as coordinate pointers. The main contribution is presented in terms of a heuristic that excludes all irrelevant nodes from path calculations even if they are reachable from source. In addition, the heuristic determines the candidate subgraphs by marking ancestors of a destination node using reverse matrix representation. The time complexity of the proposed algorithm is expected to be bounded with complexity of known algorithms, ie., $O((|V|+|E|)log\ |V|)$. Researchers will invistigate in-depth the performance improvements and measurements of the new algorithm in cumming extended research.

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, Al-Isra Private University, Amman, Jordan, December 6-8, 2005.

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

[3] T. Cormen,. C. Leiserson, R. Rivest, C. SteinX, Section 24.3: Dijkstra's algorithm". Introduction to Algorithms (Second ed.). MIT Press and McGraw-Hill. pp. 595–601. ISBN 0-262-03293-7, 2001.

[4] J. Orlin, K. Madduri, K. Subramani, and M. Williamson, "A faster algorithm for the single source shortest path problem with few distinct positive lengths," J. of Discrete Algorithms, vol. 8, no. 2, pp. 189-198, 2010.

[5] L. Xiao, L. Chen, and J. Xiao, "A new algorithm for shortest path problem in large-scale graph,". Appl. Math, vol. 6, no. 3, pp. 657-663, 2012.

[6] F. Zhang, A. Qiu, and Q. Li, "Improve on Dijkstra;s Shortest Path Algorithm for Huge Data," Chinese academy of surveying and mapping, 2005.