A Comparison between Graph-Compression and Landmark Shortest-Path Approaches

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

Faisal Khamayseh[*]
Department of Computer Science and Engineering,
Palestine Polytechnic University
Hebron, Palestine
faisal@ppu.edu

* Corresponding Author

***Abstract*—Shortest path improvement is one of the most important and recent issues in combinatorial optimization. Emergency routs, road and public transportation networks, routing schemes for computer networks, social networks, and other applications are all motivating the study of improving shortest path finding. One of the big obstacles in such real-word applications is the size of the graphs. Given a weighted graph G(V,E,w), an algorithm to improve classical shortest-path for a given source <s> is provided. The contribution of this paper is to study the improvement of finding shortest-path by compressing the graph while preserving the graph properties and comparing it with the approach of using landmark optimization. In this paper we implement the new approach of graph compression and the landmark approach. We discuss the performance, storage, error rate in our approach compared to landmark. The paper concludes that the new graph compression technique performs with no errors compared to fast landmark approach which is error prone when selecting source or destination vertices out of the landmark nodes.**

Keywords*—*Graph Compression, Shortest-path, landmarks, Reverse Representation, Candidate Subgraphs.

I. INTRODUCTION

The shortest path problem is one of the most encountered graph based problems as it is crucial in the real-world applications. The huge number of graph nodes and links makes the existing path-finding algorithm incredible in terms of time complexity. Shortest path is needed in real-world applications such as communication, transportation and routing networks. In small non-dynamic graphs, pre-processing is very beneficial. However, pre-processing procedures require much space. It is not worthy to move the complexity from real time to pre-processing unless saving is tangible.

Since finding the shortest path over network topology is quite expensive, it is worthy to work towards improving existing classical algorithms such as the well-known Dijkstra's algorithm for finding shortest path [1]. Many considerable research attempts aim to improve existing shortest path algorithms and represent the graph in different structures and representations [2]-[11]. Some recent and current research benefit more from the pre-processed path preparations [6], [9],[11] and [13].

This paper describes some recent existing techniques for improving shortest path finding [9],[11] and [12]. Subgraph compression as a recent technique is very important which focuses on reducing the graph in order to lower shortest path finding effort. In this research we focus on the comparison between the improved shortest-path algorithm using path compression and the technique of using landmark.

Given a graph G=(V,E,w) with vertex set V, edge set E, and given weight function w(e) as nonnegative function for each edge e $\in$ E, a compressed graph CG(CV, CE, w) is generated from graph G using compression procedure. CV is the set of vertices in CG, CE is a set of edges formed using compression function, and w is the positive weight function. The compression function transforms G into new graph while preserving the graph properties. That is, the weight-sum of the shortest path from source <s> to target <t> in the compressed graph is the same as it is in original graph G.

Complexity of shortest path goes around O(|V|log|V|). 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 [14].

Recently, researchers focus on improving shortest path procedure and hence improving complexity. Some attempts commit to improve shortest path algorithm based on search strategy by introducing a constraint function with weighted values and ignoring the large number of irrelevant nodes during shortest path finding [10], [15]-[19]. Some researchers have focused more on overcoming the network structure rather than the algorithm itself. Some improvements introduced to find the shortest path through graph partitioning. They took an advantage of the graph features to improve the search such as network properties, and sentence fusion using dependency graph structure to acquire compression [15] and [18]. The main feature in the recent improvements is the possibility of partitioning the graph into a set of components or clusters and in better representations with some constraints. They focused on simplifying the detailed graph by clustering adjacent nodes and benefiting from components on the transit edges.

Landmark approach selects set of nodes that cover portions of the graph. Finding the optimal set of nodes that covers the entire graph is NP-hard [19]. Landmark approach aims at decreasing the time needed to find the frequent shortest path finding [20] and [21] . This is not very beneficial in dynamic graphs especially when the real-world applications are critical such as aviation routs, emergency communications, and medical data communication networks.

II. Graph representations

Different graph representations have been used to handle the given graph. In graph compression technique, the sets of V nodes and E edges with corresponding positive weights are required to be represented in two matrices with maximum $|V|^2$ elements to represent the graph in normal and reverse representations [1]. For example, Fig. 1 depicts a random generated graph G (V,E,w) to be represented in the matrix structure. Table 2 shows the reverse matrix representation of the graph G. The main advantage of this representation is in finding all possible ancestor nodes starting from the destination node <ti>.

The reverse pre-process structuring procedure limits the candidate subgraph to work in. It is very useful in real-world path-finding applications to exclude irrelevant nodes or subgraphs. For example, if our source vertex in the graph G in Fig. 1 is v21 and the destination vertex is v33, there is no need to put effort out of the candidate subgraph G'(v21, v26, v28, v29, v30, v32, v33) which is the subhraph leading to destination.

It is required to represent the graph in linear representation for fast references of nodes as shown in table 3. For efficient implementation and to save storage, the matrix can also be represented as a linear array with |E| entries. This representation matrix stores all vertices showing all paths. The matrix is beneficial in finding all reachable vertices in the graph for given source and destination nodes. It takes only linear time to check path existence. This matrix, helps in finding simple sub baths of vertices with in-degree and out-degree of 1 where we apply subpaths' compression.

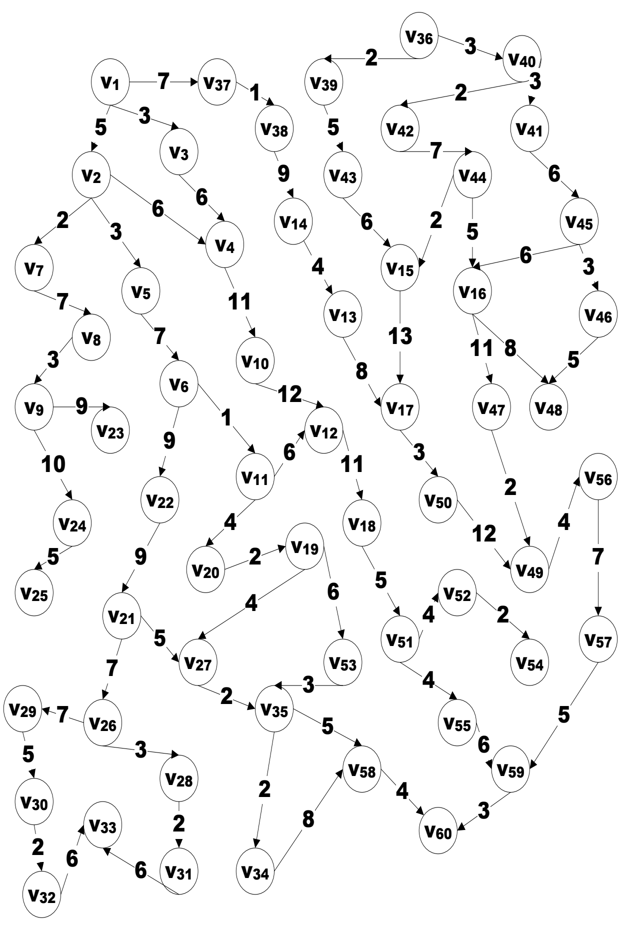

Fig. 1. Graph G = (V,E,w)

TABLE II. Graph Matrix Representation for Graph G

--	0	1	2	3	4	5	6	7	8	9	10
0	**v1**	**v2**	**v4**	**v10**	**v12**	**v18**	**v51**	**v52**	**v54**		
1								**v55**	**v59**	**v60**	
2			**v5**	**v6**	**v11**	**0,4**					
:	:	:	:	:	:	:	:	:	:	:	:
13	**v36**	**v39**	**v43**	**v15**	**12,5**						
:	:	:	:	:	:	:	:	:	:	:	:
17			**v42**	**v44**	**13,3**						
18					**15,4**						

Another advantage, is that all the graph roots (zero in-degree vertices) appear in the first column. The paths are represented as a depth first search traversal order while common parts of the paths are stored only once using array indexing to avoid duplications of subpaths.

When the reference of the vertex is in the form of <ri, cj>, this node <ri, cj> is previously visited from different node. That is, the node <ri, cj> can be visited from more than one path. For example, if paths p1 is represented as vertices <v1,v2,vi,…,vn-1,vn> and p2 is <v1,v2,..,vi,..,vm>, P1 and P2 are presented paths G, then path p2 is stored in the next row of path p1 starting from the column (i+1) representing the rest of p2 as <vi+1, vi+2, …> with empty i+1 entries.

TABLE II. LINEAR ARRAY REPRESENTATION

Node	***Reference***		***Node***	***Reference***		***Node***	***Reference***
0,0	v1		5,8	3,8		12,6	v50
0,1	v2		6.4	v22		12,7	v49
0,2	v4		6.5	v21		12,8	v56
0,3	v10		6.6	v26		12,9	v57
0,4	v12		6.7	v28		12,10	1,8
0,5	v18		6.8	v31		13,0	v36
0,6	v51		6.9	v33		13,1	v39
0,7	v52		7,7	v29		13,2	v43
0,8	v54		7,8	v30		13,3	v15
1,7	v55		7,9	v32		13,4	12,5
1,8	v59		7,10	6,9		14,1	v40
1,9	v60		8,6	3,7		14,2	v41
2,2	v5		9,2	v7		14,3	v45
:	:		:	:		:	:
:	:		:	:		:	:
5,7	v53		12,5	v17			

III. COMPRESSION METHOD

For the given graph G(V,E), graph compression is a task where the goal is to produce a new graph G' from

G while preserving the paths and weights properties such as source, destination, distance between nodes, subpaths' weights, etc. On other words, properties of the original graph is preserved in the compressed graph and vice versa. The procedure finds all linear subpaths in which every node has no more than one out-degree and only one in-degree, adds up the subpath weights and transforms it to only one edge with its total weight. Fig. 2 shows the new compressed graph G' (V,E',w') after applying graph compression procedures which reduces the number of nodes and edges. For example, the path v1 to v17 in G is <v1, v37, v38, v14, v13, v17> costs 29, while in G' the same path is compressed into <v1,v17> preserving the same total weight. This procedure helps in jumping between nodes starting from the head vi to vj if all vertices between vi to vj have no more than one in degree and no more than one out degree. Linear subpaths can be easily determined from rows and columns of table 1. This compression procedure performs well on sparse graphs compressing all consequent nodes on linear subpaths.

Fig. 2 shows a pictorial illustration of having candidate subgraph with reverse representation together with compression generated from the original given graph G shown in Fig. 1.

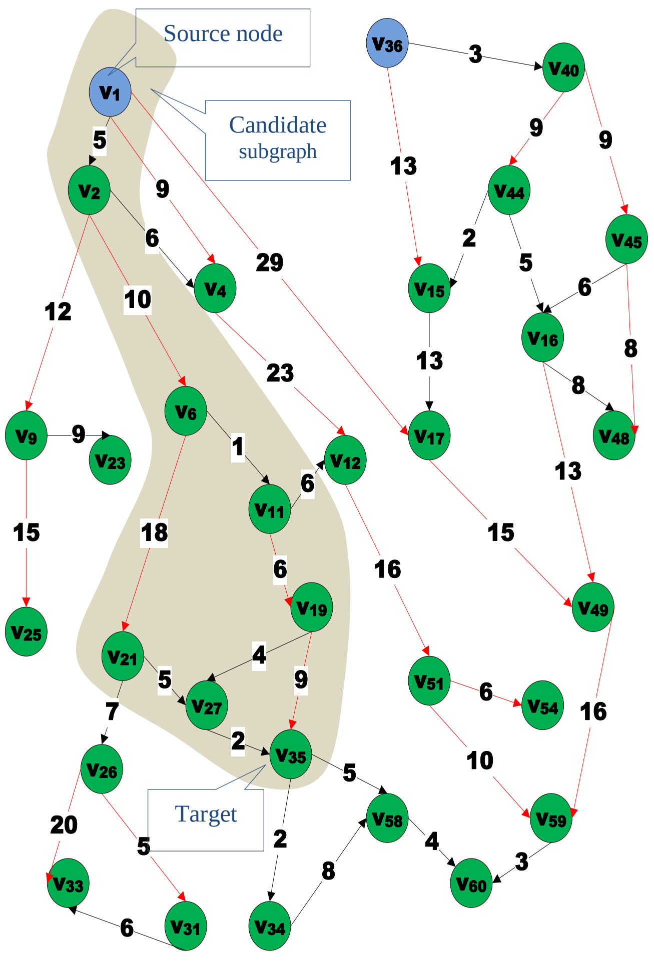

Fig. 2. Compressed Graph

IV. Shortest Path Using Graph Compression Algorithm

This new algorithm established set of matrices to represent the graph in normal structure and reverse form. Reverse matrix array is used to represent the graph G where ancestors and decedents of each node vi are reversed to represent all vertices where vertex vi is reached from. The optimized procedure to compress the graph and determine the candidate subgraph is then take place where huge irrelevant parts of the graph may be excluded. This step limits the graph area where the shortest path presents.

The algorithm establishes the graph matrix representation where all nodes and weights are represented as shown in table 1. The inner structure includes the current vertex, minimum distance, and predecessor node. This aids in determining the direct path with its corresponding 0/1 flag, the goal node, and accumulated weights. Then the algorithm saves the direct paths for nodes vi to vj where out-degree and in-degree of each node on this direct path equals to 1as of the following steps:

- Store the node parent, where out-degree[parent] equals to 1.
- Find the Goal-node, which is the first node along the direct path with the out-degree[Node] is other than 1.
- Save the direct path between the parent and the Goal-node and the accumulated weight in the new representation compressed matrix.

After this compression step, the algorithm constructs the reverse matrix to represent the graph rooted with all graph destinations forming the candidate subgraph. Finally, it traverses the graph G starting by the given destination to mark all candidate nodes in the main matrix representation. This is possible using reverse matrix which enables 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 value in the main matrix which represents the main graph.

After marking the candidate subgraph in the main matrix, and starting from the given source s, the algorithm checks the existence of a direct path. The algorithm directly selects the direct path in the case it exists, otherwise it adds all neighbour edges by visiting all nodes listed in the next column in a breadth fashion of the current node.

This new algorithm "Shortest Path Using Graph Compression" finds the shortest path based on reducing the number of edges and nodes that limits the search to take less time than searching the entire graph. The algorithm also determines the candidate part of the compressed graph which excludes irrelevant graph parts. This efficient procedure saves much work comparing to the functionality of known conventional algorithms especially when applied on real-world problems.

Algorithm: *Shortest Path Using Graph Compression (Graph, Mark, Reverse_Matrix)*

```
{
initialize Mark Matrix to unmark (false mark)
  select destination vertex t
  start at destination vertex t in Reverse Matrix
Mark all ancestor nodes to mark (true mark)
Select a source node s
   dist= FindShortest(Graph Matrix, Mark Matrix, s)
 }
```

Function: FindShortest(GraphMatrix, Mark Matrix, s)

```
{
for each vertex v in Graph Matrix
    {initialize distance of v to infinity;
    Initialize predecessor of v to undefined;
end for

for every vertex starting from paths heads to paths ends
in Graph Matrix // compression
        add up weights of all consecutive vertices vi's with
        in-degree and out-degree no more than 1.
        store the total weight as an a single edge for vi+1
end for
 dist[s] := 0 ;
MarkQ = set of Marked nodes in Graph Matrix ordered
in dfs visit
for selected source s
        Select smallest nodes with distance using marked
        nodes in MarkQ only
        Update dist

return dist;
 }
```

V. Performance

The fundamental objective of shortest path using graph compression algorithm is to come up with better performance comparing with other classical shortest path algorithms and also comparing with other improvements such as using landmark. The algorithm finds the shortest path P(s,t) between two given vertices <s> and <t> in a directed weighted graph G(V,E,w). It obviously determines the compressed graph G'(V',E',w) with the candidate subgraph G" determining the part that includes the possible paths between source and destination. This improvement reduces the number of edges |E| to the new number of edges |E'| and the number of vertices |V| to |V'| which lowers the cost significantly.

Applying compression approach on graph G(V,E,w), reduces the graph G to G'(E',V,') with significant saving about 35% of the total number of edges in G, for example.

It is required to have O((|V|+|E|)log |V|) for each shortest path applying the classical algorithms making the cost extremely high which is impossible in many real-world applications such as in communication networks. Comparing with late shortest path improvement attempts, the recent algorithm using graph compression [9] introduces more tangible improvements. Moreover, this algorithm results in better and improved performance on dense graphs with better and tangible performance on sparse graphs.

With reference to experimental phases, graph compression approach provides evidence that the proposed heuristic outperforms the conventional algorithms. The performance of the algorithm shows a considerable saving in the cost of random generated graphs with different sizes. The experiment shows a considerable saving in performance when applied on dense graphs and more on sparse ones in most of the trials. Figures of average performance of applying the new improving approach on set of random graphs with different density degrees are shown in Fig.3 and Fig.4. This procedure shows the algorithm with compression approach outperforms the improved procedures.

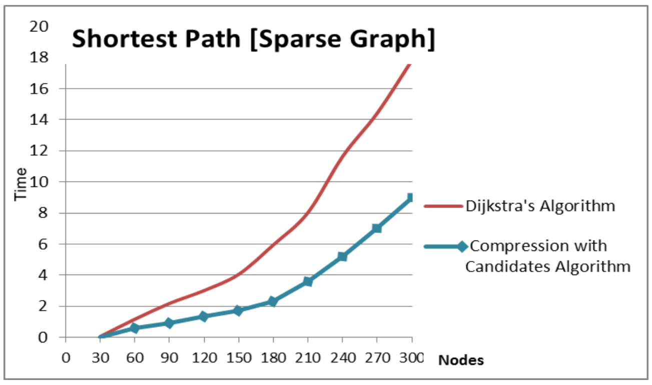

Fig. 3. Performance of Shortest Path using Compression Algorithm on Sparse Graph

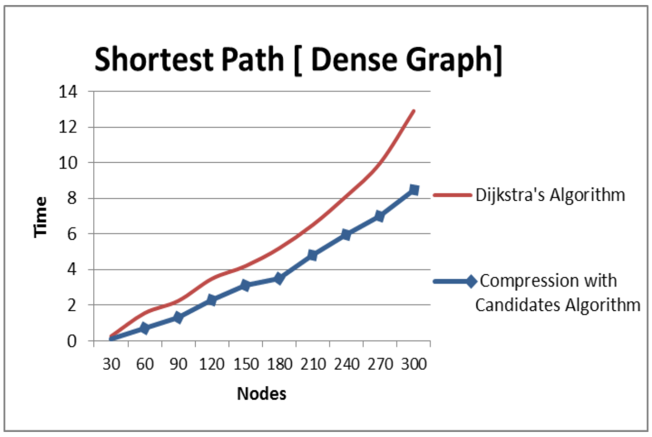

Fig. 4. Performance of Shortest Path using Compression Algorithm on Dense Graph

VI. Compression Approach vs Classical Landmark

Finding shortest path using landmark approach is fast if source node is one of the landmark vertices. However in real random selection of source nodes, landmark approach calculates the shortest path between the source and the closest landmark point. This turns out to have the normal shortest path complexity where classical shortest path finding is required. The risk, is that shortest path between source and destination may not go through any landmark vertices. On the other hand, pre-processed shortest paths via landmark requires large storage on each landmark vertex. Fig.5 shows the same graph G with landmark selection. The performance figures of applying landmark approach on set of random graphs with different density degrees are shown in Fig.6 and Fig.7.

In landmark approach, larger k leads to less errors. The reason is that large k covers the graph in more small-subgraphs covering more source nodes. In this case compression approach and even classical approach is much better.

Compression approach requires less space and keeps preserving the properties of the original graph. Therefore, compression approach is error free. The experiments show considerable saving in performance in dense graphs and more in sparse ones in most of the trials. Average performance figures of applying the new improving approach on set of random graphs with different density degrees are shown in Fig.3 and Fig.4.

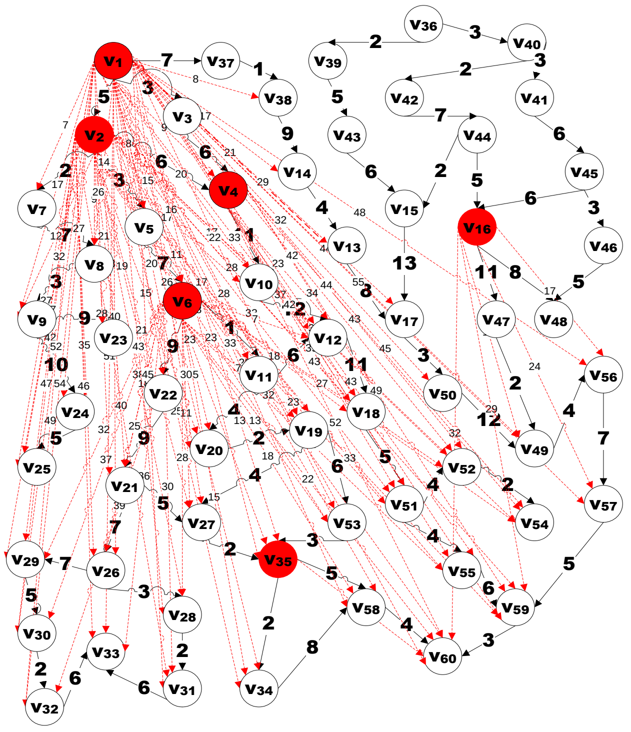

Fig. 5. Graph G with 6 Landmark Points

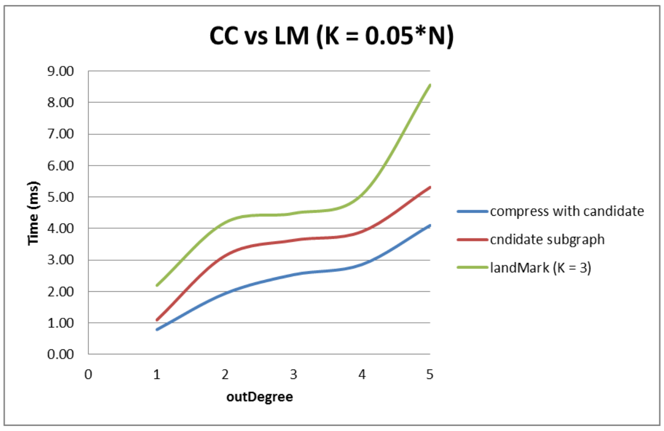

Fig. 6. Landmark performance using .05% of G

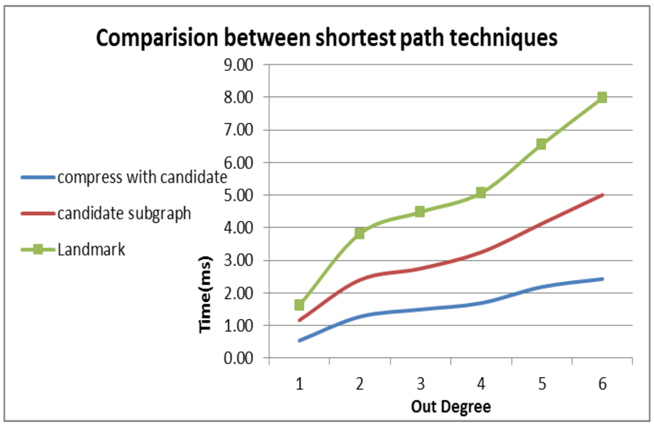

Fig. 7. Landmark performance using 15% of G

Table 3 and Fig.8 summarize the comparison between the two algorithms: graph compression with candidate subgraph approach and the landmark approach in terms of performance, space and correction.

TABLE III. COMPARISON OF GRAPH COMPRESSION WITH CANDIDATES VS LANDMARK APPROACHES

LandMark	Candidate Subgraph	Compress with Candidate	Out Degree
0.71	0.52	0.32	1
1.87	1.67	1.63	2
2.14	2.19	2.01	3
2.94	2.44	2.12	4
3.33	3.14	2.67	5
3.76	3.42	3.00	6

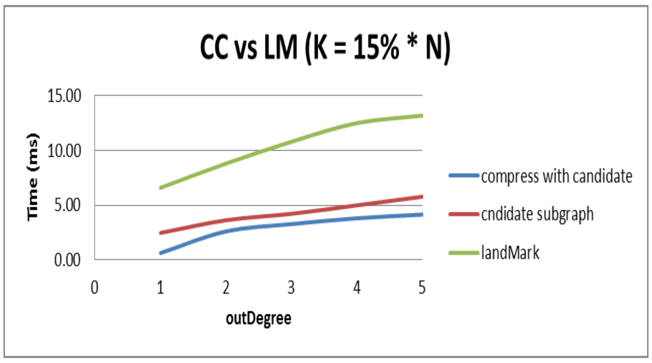

Fig. 8. Comparison between shortest path techniques

VII. RESULTS

Table 3 compares between the heuristic of shortest path using graph compression and the approach of using landmark. The applied new graph compression procedure for improving shortest path finding in weighted graphs outperforms the classical procedures and outperforms the approach of using landmark.

Given the graph G(V,E,w), the new compression technique stores the graph in different matrices with different structure properties while preserving the properties of the original graph. The new approach improves finding shortest path using the candidate subgraph along with compression technique. The statistical figures result in better performance when applying the graph compression together with the exclusion of the irrelevant nodes using candidate subgraphs approach. These experimental statistical figures show the average performance of applying the new approach on set of random graphs with different density degrees. That is the applied new procedure for improving shortest path finding in weighted graphs outperforms the classical procedures and outperforms the approach of using landmark.

VIII. CONCLUSION

Given weighted graph G(V,E,w), w(e) is a positive weight function, an efficient and improved algorithm for finding shortest paths between a given source <s> and destination <t> using candidate subgraph along with graph compression is implemented and discussed. The candidate and compression procedures are applied on random generated weighted directed graphs that vary in sizes and range from sparse to dense. The approach of using landmark to find shortest path is applied on the same graphs and compared in terms of performance. In the practical phase, the algorithm shortest path using compression technique outperforms the performance of landmark approach and other improved algorithms with the given properties and constraints. Graph compression algorithm shows obvious improved performance on generated graphs. As a heuristic algorithm, the complexity will always be bounded by the complexity of known normal algorithms, ie., it will not exceed O((|V|+|E|)log |V|) for each source <s> and each destination <t> while it shows tangible saving especially when applied on real-world graph problems.

The average performance of applying the graph compression approach on set of random graphs with different density degrees is discussed. The applied new procedure for improving shortest path finding in weighted graphs outperforms the classical procedures and outperforms the approach of using landmark technique.

ACKNOWLEDGEMENTS

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 Nasereddin and Ms. Salma Dirbashi for their help in implementing the algorithms.

REFERENCES

[1] E. W. Dijkstra, "A note on Two Problems in Connexion with Graphs", Numerische Mathematik 1: 269 271. doi:10.1007/BF01386390.

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

[3] J. B. Orlin, K. Kamesh 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, 8, 2 (June 2010), 189-198

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

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

[6] 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.

[7] F. Simek and I. Simecek,"Improvement of Shortest Path Algorithms through Graph Partitioning". International Conference Presentation of Mathematics. Liberec, Czech Republic, 2011.

[8] 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, Paris: Atlantis Press, March 2013: 226-229.

[9] F. Khamayseh and N. Arman."Improvement of Shortest-Path Algorithms Using Subgraphs' Heuristics", Journal of Theoretical and Applied Information Technology, 2015. Vol. 76, No.1.

[10] L. Yunpeng, J. Yichuan, Z. Yong. "A new Single-source shortest path algorithm for nonnegative weight graph", Cornell university library, retrieved May 9th, 2015 from http://arxiv.org/abs/1412.1870v6.

[11] N. Arman and F. Khamayseh, “A Path-Compression Approach for Improving Shortest-Path Algorithms”, International Journal of Electrical and Computer Engineering (IJECE), Vol.5, No.4, September 2015.

[12] F. Khamayseh and N. Arman,"An Efficient Multiple Source Single Destination (MSSD) Heuristic Algorithm Using Nodes Exclusions", International Journal of Soft Computing, Vol. 10, No. 3, 2015.

[13] N. Arman, (2005). 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.

[14] 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. ISBN 0-262-03293-7.

[15] K. Filippova. & M. Strube. "Sentence fusion via dependency graph compression". Proceeding of EMNLP-08, 2008, pp. 177–185.

[16] J. Zhang, J. Li, X. Fan, Z. Deng, “Research on Real-Time Optimal Path Algorithm of Urban Transport”, TELKOMNIKA Indonesian Journal of Electrical Engineering, Vol.12, No.5, May 2014, pp. 3515 ~ 3520.

[17] W. Yahya1, A. Basuki2, J. Jiang. "The Extended Dijkstra’s-based Load Balancing for OpenFlow Network", International Journal of Electrical and Computer Engineering (IJECE), Vol. 5, No. 2, April 2015, pp. 289~296.

[18] F. Katja, "Multi-Sentence Compression: Finding Shortest Paths in Word Graphs", Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010), pages 322–330, Beijing, August 2010.

[19] Frank W. Takes and Walter A. Kosters, "Adaptive Landmark Selection Strategies for Fast Shortest Path Computation in Large Real-World Graphs", The Netherlands, Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014 IEEE/WIC/ACM International Joint Conferences on (Volume:1).

[20] F. Volodymyr, T. Konstantin and D. Marlon Dumas, Memory-Efficient Fast Shortest Path Estimation in Large Social Networks, Proceedings of the Eighth International AAAI Conference on Weblogs and Social Media, Association for the Advancement of Artificial Intelligence, 2014.

[21] Q. Miao, C. Hong, C. Lijun, X. Y. Jeffrey, "Approximate Shortest Distance Computing: A Query-Dependent Local Landmark Scheme", Proceedings of the 2012 IEEE 28th International Conference on Data Engineering, p.462-473, April 01-05, 2012.