

How to Cite:

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*, 12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

## **A COMPLETE COMPRESSING SPARSE GRAPHS IN DIJKSTRA'S SHORTEST PATH ALGORITHM**

**Poonam<sup>1</sup>**

**Ph.D. Research Scholar Dept. of Mathematics,  
Maharishi School of Science,  
MUIT University, Lucknow, U.P.**

**Dr. Chitamani Tiwari<sup>2</sup>**

**Professor, Research Guide,  
Dept. of Mathematics,  
Maharishi School of Science,  
MUIT University, Lucknow, U.P.**

### **ABSTRACT**

*Dijkstra algorithm is one of the prominent algorithms to find the shortest path from the source node to a destination node. It uses the greedy approach to find the shortest path. The concept of the Dijkstra algorithm is to find the shortest distance (path) starting from the source point and to ignore the longer distances while doing an update. One of the problems that arises from the continuously growing amount of data is that it slows down and limits the uses of large graphs in real world situations. Because of this, studies are being done to investigate the possibility of compressing data in large graphs. This report presents an investigation on the usefulness of compressing sparse graphs and then applying Dijkstra's shortest path algorithm. A minimal spanning tree algorithm was used to compress a graph and compared with a self-implemented compression algorithm. The minimal distances and how long time it takes for Dijkstra's algorithm to find the shortest path between nodes are investigated. The results show that it is not worth compressing the type of sparse graphs used in this study. It is hard to compress the graph without losing too much of the edges that preserve the shortest paths. The time gained when running Dijkstra's algorithm on the compressed graphs is not enough to compensate for the lack of getting a good shortest path solution.*

**Keywords:** Algorithm, graphs, tree, spars, shortest, compressed etc.1.0 Introduction

The first graph problem which lay the foundation for graph theory was introduced by Leonard Euler. The problem is called "Seven bridges of Königsberg". The problem was about finding a walk over the seven bridges and crossing each bridge exactly once. This graph problem was an undirected graph with seven edges and four nodes. Where the nodes

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Poonam and Dr. Chitamani Tiwari**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

**How to Cite:**

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*, 12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

represented the islands of the city and the edges represented the bridges connecting the islands [7].

Since then the usage of graphs have been increasing and is now used almost everywhere when there is a problem to be modeled. The difference today compared to the early graphs that were used is that they are a lot bigger now. Instead of around ten nodes there are now millions of nodes in the graphs. Because of this, research has been made to compress the size of these graphs in the cases where they become too large [4].

Graphs has for a long time been widely used in computer science. It is used for modelling, storage and more. A lot of problems can be modelled as a graph and therefore they are of great importance when a complex problem have to be solved or when trying to find certain information in connected data. However, today's large sets of data is making it hard for engineers to make use of such large graphs. One example is the web graph covering the entire internet with all the hyperlinks between the different links, making it a large graph with billions of nodes [4]. It is because of problems like this that a search for compressing the large data graphs into smaller ones exists. The advantage of compressing a graph is that the size can be smaller and that it can be a way for speeding up certain algorithms. A lot of research has been made with compressing large graphs, such as the web graph or social network graphs. The motivation for the search after good compression algorithms has mostly been to be able to store graphs in the main memory of servers and computers. This is because the time it takes to read data from the hard drive is about five times slower than reading directly from the main memory [6].

Dijkstra's shortest path algorithm is a well known graph algorithm in computer science. It is used for finding shortest paths in a graph between two given nodes and can for example be used for finding the shortest way from one city to another city in a road network graph. This report will investigate if a compression on a sparse graph will speed up runtime of Dijkstra's shortest path algorithm without losing too much of the minimal distances between the nodes. Problem Statement

The goal of this report is to look at a way to compress a graph that will increase the speed of the Dijkstra's shortest path algorithm while still trying to keep the minimal distances between nodes. In this report a minimal spanning tree algorithm will be used to compress a graph, thus greatly reducing the number of edges of the graph. Besides this we will also construct our own compression algorithm to compare results of running the shortest path algorithm on the original graph, minimal tree spanned graph and our own compressed graph.

The question that we will try to answer is if it is worth compressing a weighted directed sparse graph corresponding to a road network, for running Dijkstra's algorithm while still keeping all the nodes with either a minimal spanning tree or our own dynamic Dijkstra compression? This will be evaluated with the results of the following:

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Poonam and Dr. Chitamani Tiwari**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

#### How to Cite:

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*, 12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

- the average path length between two randomly chosen nodes when running the shortest path algorithm on the different graphs
- the time it takes to run the shortest path algorithm on the different graphs

## 2.1 MOTIVATION

According to this algorithm here we use compressed graphs work which is an important question in today's technology because the information that is in use everyday continues to grow. The field of big data introduces problems with large sets of information that is hard to handle. If there is no way to store all that information in a smarter and more efficient way it will become problematic to use the information. Compressing graphs can reduce the size of such large datasets and is a way for increasing the speed for certain algorithms, like finding a path between two nodes in a dataset. It is also a great way to shrink the size of the graph so that it can be stored in the main memory and therefore avoid reading from the disk, or to visualize a graph to make its properties more visible.

Previous work has mostly investigated the ability to compress large dense graphs. These often contain lots of nodes and a high amount of edges. There has been several studies showing that large graphs can be efficiently compressed. This report will however focus on smaller graphs and, especially so, corresponding to a road network or a graph used for deliveries with nodes corresponding to addresses and drop off points. Clearly, an interesting study would be to investigate if there is any time to gain when compressing a sparse graph and then trying to find the shortest path between two nodes. This while still making sure that the found path differ as little as possible in length compared to the original graphs shortest path.

To compress the graph a minimal spanning tree (MST) algorithm will be used to experimentally see if this well known algorithm can yield good results when applying Dijkstra's algorithm on it. It will give a graph that is compressed by the number of edges as much as possible without losing the connectivity of the graph. Since it is a MST and not only a regular spanning tree it should give relatively good results when the weights of the edges are of concern. This type of compression will be compared to a second compression where a higher number of edges will be kept, which should give better shortest path lengths in the graph at the cost of longer runtime for Dijkstra's algorithm compared to MST.

The second compression algorithm that will be used is a modification of Dijkstra's algorithm, with a goal of keeping the cheapest path between the nodes while still removing as many edges as possible. This compression will not reduce the number of edges as much as the MST but it will hopefully give better minimal distances between nodes.

The results will give an insight of how much you can reduce the number of edges in a graph to speed up the Dijkstra's algorithm while still trying to keep the distances between nodes as close as possible to the original.

How to Cite:

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*, 12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

### 3.0 Background

In the background some definitions, compression schemes and algorithms are presented needed for the reader to understand the report.

#### 3.1 Definition of graph

A graph consists of nodes and edges. There are several types of graphs. They can be directed or undirected. An example of an undirected graph is for example a social network graph where you have edges between people who know each other. A directed graph is for example a public transport graph where every bus and subway has a fixed direction where it goes.

The notation of a graph is the following:

$G = (V, E)$  where  $G$  is the graph,  $V$  is the number of nodes in the graph and  $E$  is the number of edges in the graph.

The above is however not enough information to describe the graph. You also need to know which nodes are connected by the different edges. Below is described how an edge can be represented.

$E = (vSource, vTarget, weight)$ , where  $E$  is the edge,  $vSource$  is the node where the edge starts from,  $vTarget$  is the node the edge goes to and  $weight$  is the given weight the edge has. The weight can for example represent how long the road is between two crossroads or drop off points for delivery companies.

A multi-graph which is looked at in this report has the property that there can be several edges with the same direction between the two same nodes but with different weights [2, 3].

##### 3.1.1 Sparse graph

A sparse graph is a graph where there are relatively few number of edges compared to the number of nodes in the graph. The limit of when a graph becomes sparse is vague but in our graph experimentations it is clear that it is a sparse graph since the number of edges is about four times bigger than number of nodes [8]. Even though the definition of a sparse graph is vague formula 1 acts as a guideline.

$|E| = O(|V|^k)$ , formula 1

Where  $E$  is the number of edges,  $V$  is the number of nodes and  $1 < k < 2$  [8].

### 3.2 Compression schemes

A number of compression ideas have been proposed in previous different articles. Below are some good and easily understandable compressions presented to give the reader an idea of how graphs can be compressed.

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Poonam and Dr. Chitamani Tiwari**

Submitted: 27 Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

How to Cite:

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*,12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

### 3.2.1 Virtual Node

The idea of using virtual nodes is to add a node to the graph which connects bipartite cliques where a group of nodes are all connected to another group of nodes. The virtual node then reduces the number of edges at the cost of only adding one node to the graph. See figure 1 how 9 nodes and 20 edges is replaced with 10 nodes and 9 edges. Since it looks for large maximum bipartite cliques to reduce the number of edges it works best on dense graphs. This type of compression is used to speed up certain algorithms, like a shortest path algorithm, and reduce the size of information in the graph [1, 6].

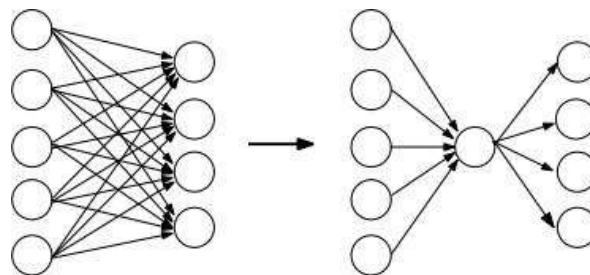


Figure 1. Virtual node [1]

### 3.2.2 SUPER-EDGES

Another compression scheme which is intended specifically for weighted graphs is the one proposed in “Compression of weighted graphs” [5]. It is based on merging nodes and edges which have similarities in terms of same neighbors and edges with close amount of weight. This is best illustrated in figure 2. The three nodes (named: 2,3,4) in the graph given in the picture have the same neighbors, 1 and 5. Node 2 and 3 are then merged into one node and their corresponding edges will be given a new weight which is the average of the two edges that were merged. It thereby compresses the graph by reducing it with 2 edges and 1 node.

The merge can continue as long as you have not reached the compression ratio you are after. This means that in the figure 2 we could as well merge in node 4 with the 2,3 node resulting in additional 2 edges and 1 node being removed. The compression ratio for this scheme is measured in  $|E'|/|E|$  where  $|E'|$  is the number of edges in compressed graph and  $|E|$  number in the uncompressed graph [5].

How to Cite:

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*,12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

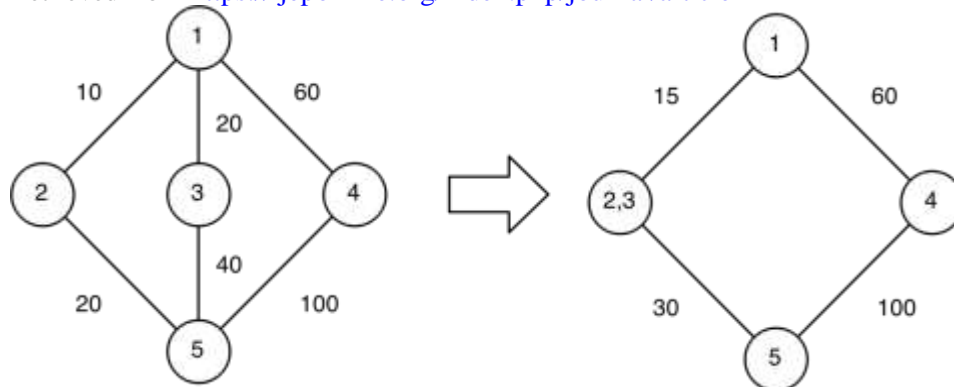


Figure 2. Super-edge compression

### 3.2.3 DYNAMIC DIJKSTRA COMPRESSION

For comparison to the graph that has been compressed with the minimal spanning tree (MST) algorithm we choose to implement our own compression. This algorithm is based on running Dijkstra's algorithm on the graph several times until a wanted compression is reached. In both MST and our compression algorithm all the nodes are kept. This makes it easy for us to do the testing for the desired results since we can specify the same two nodes to run Dijkstra's algorithm between for all the different graphs i.e the original graph, minimal spanned tree graph, and our dynamic Dijkstra compressed graph.

Our compression algorithm goes on until wanted compression is reached which is specified as a compression ratio between zero and one as mentioned in formula (2). The compression is measured as number of edges in the compressed graph divided by the number of edges in the original graph. This ratio between the graph  $G'$  and graph  $G$  and tells how much smaller graph  $G'$  is compared to  $G$ .

$$cr(G', G)$$


$$\frac{|E_{G'}|}{|E_G|}$$

formula 2 where  $cr$  = compression ratio,  $G'$  is the compressed graph,  $G$  is the graph we are comparing to and  $E$  is the number of edges.

What the dynamic Dijkstra compression algorithm does is that it chooses a number of nodes that are considered important. In our experiment we chose the nodes with the highest degree as important. This is because those nodes would represent a drop off point or a crossroad with many available road connections. That makes it a good node to use since there are many edges for available paths to other nodes in the graph and can be considered a vital node.

Before the algorithm starts we read in the original graph and we also create a new array that will hold all the nodes and edges of the new compressed graph.

We then call a function to fill a node priority queue with the nodes that are considered

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Poonam and Dr. Chitamani Tiwari**

Submitted: 27 Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

How to Cite:

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*,12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article> important.

The algorithm then extracts these important nodes from an list and begins running Dijkstra's algorithm from the important node to all the other nodes in the graph. Then the edges which gives the shortest path from each node in the graph to the chosen node are added to the new graph. In the case where the edge already exists in the new compressed graph i will not be added. This algorithm keeps running until the desired compression ratio is reached. An example, with only five nodes and seven edges, of what the first iteration could look like can be seen in figure 3.

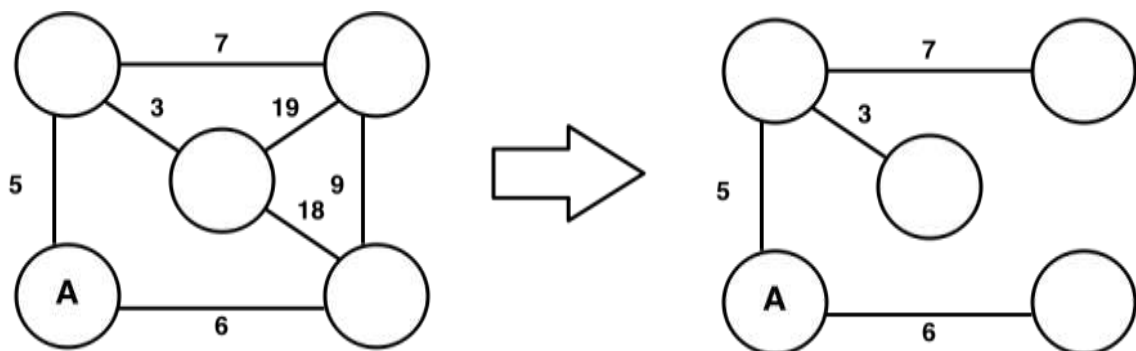


Figure 3. First iteration of the dynamic Dijkstra algorithm This compression algorithm can be developed and improved further with regards to which nodes to fill the priority queue with and deciding even more which edges to keep according to which nodes are considered more important.

#### Algorithm 1. Dynamic Dijkstra compression

**Input:** An uncompressed graph  $G$ , compression ratio  $cr$  where

**Output:** A compressed graph  $G'$  **1: for each** Vertex  $v$  in  $G$  **do**

**2: if**  $v$  is considered important **do 3: add**  $v$  to vertexPriorityQueue **4: end if**

**5: end for**

**6: while**  $cr$  is not reached **do**

**7: pop** vertex  $v$  from vertexPriorityQueue

**8: Dijkstra**( $v, G, G'$ )

**9: for each** path calculated by Dijkstra to  $v$  **do 10: if** edges in path do not exist in  $G'$  **do 11: add** edges to  $G'$

**12: update**  $cr$

**13: if** wanted  $cr$  is reached **do 14: break while loop**

**15: end if**

**16: end if**

**17: end for**

**18: end while**

**19: return**  $G'$

### 3.3 Minimal Spanning Tree

The MST works on a undirected, weighted and connected tree. A regular spanning tree will

© 2018 by The Author(s). ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Poonam and Dr. Chitamani Tiwari**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

**How to Cite:**

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*,12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

always have a path to all nodes and remove all other edges. When the MST is used then the path will also have the lowest possible total weight.

The uses for a MST of a graph is widely used. One example is if a neighbourhood should get new water pipes put in the ground. Then the company can make a graph over the neighbourhood with all the houses as nodes and all the possible ways to lay the pipes as edges between the houses. The company will of course try to minimize the cost of putting in the new pipes. Every edge in the graph gets a value, the cost of putting in the pipes along that path. Then the company will use a MST algorithm on that graph and the cheapest possible way to put in the pipes will be found [2].

One of the most popular and best shortest path algorithms out there is the Dijkstra algorithm. Dijkstra is a greedy algorithm that has the time complexity  $O(|E| +$

directed graph with positive weights on the edges between the nodes.

The algorithm finds the shortest path between the start node and the end node. When the algorithm starts it puts a really high number, possibly infinity, on all the nodes except the starting node. As seen in figure 4(a). The algorithm starts to check if any of the neighbours weights of the nodes in fig 4(a), which is infinity at the beginning, is lower than the starting nodes weight plus the weight of the edge that binds them. If the total weight is lower than the neighbours nodes weight it replaces it with the new lower weight, as seen in fig 4(b). Now the algorithm will go through all the neighbours of the new node to see if any values can be replaced, and so it continues. Once it has visited all the nodes it is finished. It can then answer with the cheapest way and which path it took to get there, see the finished version in fig 4(f) [2].

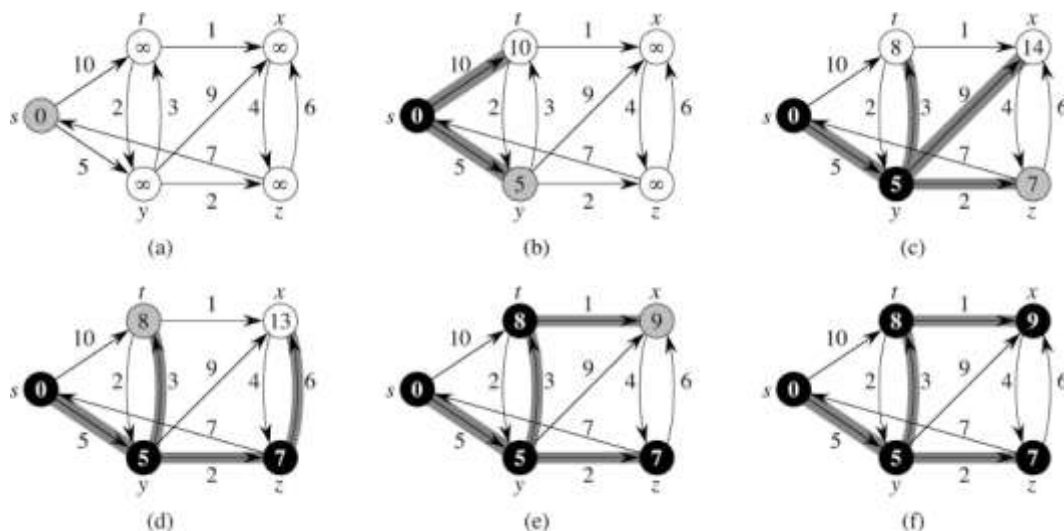


Figure 4. Dijkstra algorithm [9]



How to Cite:

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*,12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

#### **4.0 Method**

The work began by gathering information on the topic of graph compression. We started out by reading articles on current state of graph compression and how they are used today. After this background study we started looking for popular compression algorithms for compressing weighted graphs. Since it was hard for us to get a hold of compression algorithms presented in the different papers that we read we eventually chose to use minimal spanning tree (MST) as a graph compression. MST is not really considered a graph compression since it is a naive approach to compress a graph. MST can be seen as a type of extreme compression where only one edge for every two connected nodes are kept. This means MST cuts off precisely as many edges that it still preserves the connectivity of the graph. We chose this algorithm because it preserves all the nodes and will be easily compared to our own compression algorithm where we will keep more edges than the MST.

To compare the MST to something else we also chose to implement our own compression algorithm which is described in the background under section 3.2. This algorithm as well as everything else that we implemented is done in JAVA.

#### **4.1 The graph generation**

The graph which is used in this report is supposed to simulate a road network graph of some sort. For example a delivery companies drop off points graph with edges between the connected drop off points and weights corresponding to the road lengths between them. This type of graph is also called a weighted multigraph meaning that there can be parallel edges. Parallel edges occur in a road graph when there are different roads going from and to the same places.

A fairly obvious but still important note about the graph is that every node has at least one neighbour node and it does not exist an island of nodes that are separate from the rest of the graph. This means that regardless of which nodes a passenger starts in he or she can reach every other node in the graph. The road graph can be considered a sparse graph since there are a few number of edges going out from the nodes compared to the number of nodes in the graph. This differs from previous works where the graphs are usually dense and also with more nodes.

The graphs which the experiments are made on is generated by code written and implemented by ourselves. The graph generation starts with the creation of the required number of nodes. It then connects node zero to node n, where n is the last node in the graph, in both directions. Node zero connects to node one and then node one connects to node two and so on. When this step is done we know that the whole graph is connected, thus we have no islands of nodes and every node can reach any other node in the graph.

We then randomly add one to six edges from a node to another randomly chosen nodes in the graph. The reason for this is that we do not consider a crossroad to have more than seven

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Poonam and Dr. Chitamani Tiwari**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

**How to Cite:**

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*,12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

outgoing roads from it. The node can not have edges to itself for obvious reasons. All the edges are given a weight between 1-20. The procedure of making the graph is then finished.

#### **4.2 Other considered test results**

We could test additional properties besides the ones stated in the problem statement. Some examples are that the time for running the compression algorithms or the size of the graph when saved to file could be measured. None of these were chosen in the end for comparison. This is because these results will differ greatly depending on of how the algorithms were implemented and how we choose to store the graph.

Instead the size is measured in the number of nodes and edges since this is easily comparable with other graphs and compressions. This is also how the authors of [5] choose to measure their compression ratio.

#### **4.3 The experiments**

The experiments are made on graphs with 10 000, 100 000 and 1 000 000 nodes. The three graphs were generated by our graph generator. In the paper "Compression of weighted graphs" [5] they used graphs with 10 000 - 200 000 nodes and therefore we thought it would be a good idé to use those numbers as guidelines and one even bigger than that, hence the graph with 1 000 000 nodes.

We then began compressing these three graphs with MST and dynamic Dijkstra's compression. The MST naturally compresses the number of edges to  $|V| - 1$  number of edges, where V is the number of nodes. Since we have back and forward edges between all the nodes the end sum of the edges is  $2 * (|V| - 1)$ . The dynamic Dijkstra compressed the graph with 0.7 and 0.9 compression ratio, see formula 2 and algorithm 1 for this. These are referred to as DynDij 0.7 and DynDij 0.9 in the result section.

We tried to present the results as generally as possible so it can be easily compared and that the hardware and algorithm implementation is of less importance. We randomly choose 20 nodes for each graph size to run our Dijkstra's algorithm from and to. The nodes from which we start and end are the same for all the graphs with the same number of nodes. The shortest path algorithm was performed ten times for each graph to minimize the chance that a lucky path was found in one of the graphs.

### **5.0 Result**

Next the results will be presented. The results that are given are the results we are after in the problem statement to answer our question whether a minimal spanning tree (MST) or our own compression can be used for sparse directed weighted graphs. With the

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

*Corresponding author:* **Poonam and Dr. Chitamani Tiwari**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

**How to Cite:**

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*,12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

consideration that this will be applied to a road network graph.

Firstly we present numbers for how many edges the graphs in the compressed and uncompressed original graphs has. These numbers are presented in figure 5 below. As can be noticed the number of edges in the MST is as mentioned before ( $|V|$ )

Dynamic Dijkstra's compression of course have 70 and 90 percent of the number of edges in the original graph. The original uncompressed graph has about four times more edges than the number of nodes because of how the graph was generated, see 4.1.

Nodes	MST	DynDij 0.7	DynDij 0.9	Original
10 000	19 998	28 524	36 674	40 748
100 000	199 998	287 380	369 488	410 542
1 000 000	1 999 998	2 871 206	3 691 550	4 101 720

Figure 5. Table representing the number of edges that the different compressed and uncompressed graph has.

The results for the average distance between two nodes is represented visually in figure 6 so the reader can get an overview of how well the different compression algorithms worked compared to the original graph. In table 6 the specific numbers for the results are given. The distance is the average calculated distance when running Dijkstra's algorithm ten times.

In figure 7 the the average distance between two nodes are represented. The percentage of how much the graphs differs from the original can be found in figure 7.

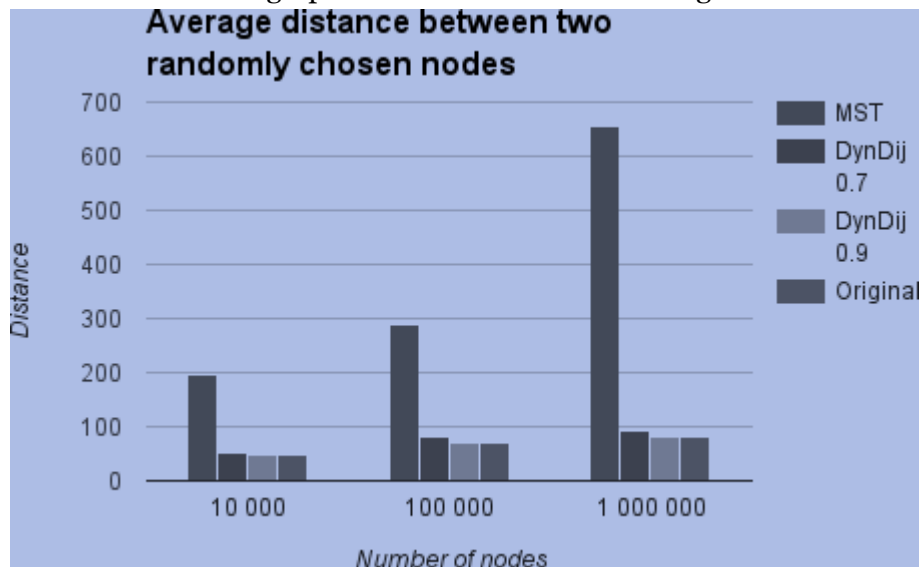


Figure 6. Showing the average path length when running Dijkstra's algorithm ten times.

**How to Cite:**

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*,12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

Nodes	MST	DynDij 0.7	DynDij 0.9
10 000	407%	109%	100%
100 000	404%	114%	101%
1 000 000	798%	115%	101%

Figure 7. Showing the path length percentage compared to the original graphs path lengths. The original graph values for this is of course 100%.

The last results to be presented are the ones mentioned from the problem statement. Which was the attribute of how much time the Dijkstra algorithm needed to run on the different compressed graphs compared to running Dijkstra on the original graph. Again the result for this is first presented visually in figure 8 and then in a table with more specific numbers in figure 9 and 10.

In figure 8 the runtime for how long the Dijkstra algorithm took to finish compared to running it on the original graph is given. The lower the staple is the less time it took for Dijkstra's algorithm to finish compared to running Dijkstra's algorithm on the original graph. In figure 9 we can see more specific results of how fast the Dijkstra's algorithm performed on the different graphs. In figure 10 the time for running the Dijkstra algorithm is presented. The numbers are given in how many milliseconds it took for Dijkstra's algorithm to finish. The results are, as mentioned earlier, an average of running the algorithm ten times.

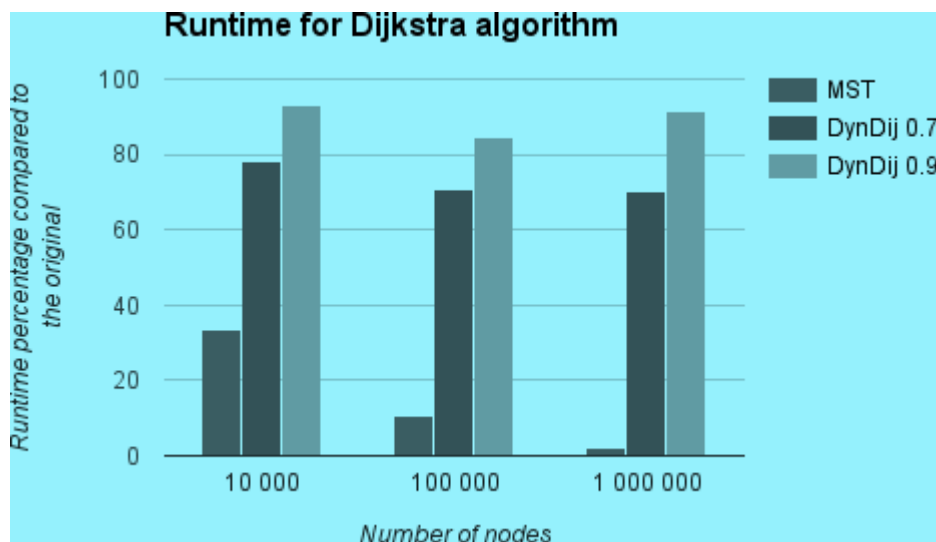


Figure 8. Shows the average time it took to run the Dijkstra algorithm ten times compared to the time it took to run the same algorithm with the same nodes on the original graph.

How to Cite:

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*,12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

Nodes	MST	DynDij 0.7	DynDij 0.9
10 000	33,5%	78,3%	93,2%
100 000	10,8%	70,5%	84,7%
1 000 000	2,2%	70,2%	91,3%

Figure 9. The percentage of the running times compared to original. The original graph has of course a running time of 100%.

Nodes	MST	DynDij 0.7	DynDij 0.9	Original
10 000	5.4	12.6	15.0	16.1
100 000	116.6	764.3	918.3	1 084.3
1 000 000	2 284.8	73 483.3	95 583.4	104 702.0

Figure 10. Showing the average time it took, in milliseconds, to finish running Dijkstra's algorithm between two randomly chosen nodes.

## 6.0 Discussion

A first notice when looking at the results is that the MST graph greatly differs from the DynDij and original graphs in both running time and minimal distance between two nodes. This is not a surprise and confirms our thoughts that MST would greatly speed up Dijkstra since it reduces the number of edges as much as possible while still keeping the nodes and connectivity in the graph. MST is however a very naive approach to compress the graph and therefore we should not be surprised when we look at the minimal distance between nodes in it. It does not preserve the minimal distances as well as the DynDij, but it also compresses the graph a lot more.

The Dynamic Dijkstra's compression performs quite well when it comes to finding a path with minimal distance close to the original graphs distances. The difference for average minimal distance between two nodes in the graph is almost none (0-1%) compared to original for the compression with 0.9 (DynDij 0.9) and about 10-15 % for the DynDij 0.7 compression.

The running time for 10 000 nodes is so low, about 15 ms for running Dijkstra's algorithm on original graph, that there is no need for compressing a sparse graph like those experimented on. The running time could also be improved with some optimization of the Dijkstra algorithm. The advantage of getting a slightly faster Dijkstra's algorithm does not win over getting an accurate shortest path in the original graph. The same can also be said for 100 000 nodes where the average running time is about 1000 ms or 1 second on the original graph. If we look at the percentages we could argue about which one is better but since the running times for Dijkstra's algorithm are so low even though the algorithms and data structures for the graph could be improved there is no reason to compress with our

#### How to Cite:

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*, 12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

compression on such sparse small sized graphs like the ones with 10 000 and 100 000 nodes.

It is when it comes to the graph with 1 000 000 nodes and about 4 000 000 edges that we can see bigger differences in runtime between the graphs. The DynDij 0.9 is around 9 % faster than the original but it only has about one percent difference in average path length. The DynDij 0.7 has 15 % longer path length and cuts the time with 30 %. While the 30 % cut in time is good the 15 % longer distance makes it bad for usage in real life. The MST graph cuts the Dijkstra's algorithm running time with 98 % but the average length is the 798 % longer than the original length which makes it useless for real life applications. With these results we think that the only one that could be used in real life would be the DynDij 0.9 because of the low difference in average path length compared to the uncompressed graph.

If the MST can be used for a road map application must be discussed. It will be a give and take between how much longer the minimal distance is between two nodes and how fast the Dijkstra's algorithm finds this path. If it is of less importance how long a person has to travel to reach their destination like if the trips total length is low no matter what path they take than MST is definitely a good choice on 1 000 000 nodes graph since it calculates the path in only

2.2 % time compared to the original graph.

### 6.1 Method critique

The graph that we originally wanted to use in this report was the SL graph. As mentioned we could not make it work, even after several phone calls and emails. Having a real world graph to test on would have been better than the simulated one that we used. A mistake we did with the graph was that we thought we would make it work with SL and therefore we did not look into any other graphs before it was too late. However now afterwards when the results are finished the SL graph might have been too small since it only consists of 10 000 buss/train stations (nodes). Once we realized that we could not make it work we were anxious to get started with the testing and figured that we would generate the graph instead.

The MST algorithm that we used is a naive way to compress a graph but it is as good lowest possible compression when seen to the number of edges. But the MST and our DynDij has a bit of different goals. The MST finds the shortest path when trying to reach all the nodes without visiting the same node twice and the DynDij find the shortest path between only two given nodes. This makes their goal a bit different and not optimal for comparison. Though both of the algorithms reduces the number of edges and still keeps all the nodes which makes it easy for us to compare how well they perform between each other.

When the testing of the graphs begun we figured that ten test for each graph would suffice. We did get clear results but they would have been more precise if we would have tested it with 100 or more tests with the Dijkstra algorithm.

#### How to Cite:

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*, 12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

### **6.2 FUTURE SCOPE AND FURTHER ENHANCEMENT OF THE DISSERTATION**

There is a lot of scope of the evaluation that we have accomplished in our thesis; our analysis should be used for the motive of monitoring the unauthorized improvement of the colonies, safety of the trees, the agricultural planning, due to the fact in our learn about location a lot of agricultural land is available. Water resource planning, infrastructures planning and so many other areas the place analysis could be used, in our united states hyper spectral and very high resolution satellites will be handy in coming years so they can furnish us very useful data, now it is up to us how we technique it and extract the beneficial information from this data.

### **7.0 CONCLUSION**

What we can see that if time is crucial and the minimal distances between nodes wants to be kept as much as possible then the Dynamic dijkstra's 0.9 compression algorithm can be used for sparse graph. It is however not worth the effort of compressing the graph for this purpose if only a few searches in the compressed graph will be performed. If a lot of searches will be performed in the graph the amount of time that can be saved for Dijkstra's algorithm will become relevant. In the end it will depend on the application and intended usage of the graph that will decide whether a compression is worth performing on sparse graphs or not.

### **8.0 REFERENCES**

[1] C. Karande, K. Chellapilla and R. Andersen, Speeding up Algorithms on Compressed web graphs, <http://dl.acm.org/citation.cfm?id=1498836>, page 272-181, last updated 2009, doi:10.1145/1498759.1498836, accessed 7 february 2015

[2] Kleinberg, Jon and Tardos, Eva. *Algorithm Design*. Second edition. Harlow, Pearson Education Limited, 2014. ISBN 978-1292023946, accessed 7 march 2015

[3] Biggs, Norman.L. *Discrete mathematics*. Second edition. New York, Oxford university press inc, 2002. ISBN: 9780198507178, accessed 7 march 2015

[4] F. Zhou, Graph Compression, [https://www.cs.helsinki.fi/u/htoivone/teaching/seminarS10/reports/zhou-graph-compression-v2](https://www.cs.helsinki.fi/u/htoivone/teaching/seminarS10/reports/zhou-graph-compression-v2.pdf).pdf, last updated 2010 , accessed 8 february 2015

**How to Cite:**

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*,12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>

[5] H. Toivonen, A. Hartikainen, F. Zhou, and A. Hinkka, Compression of weighted graphs, [https://www.cs.helsinki.fi/u/htoivone/pubs/toivonen\\_kdd2011.pdf](https://www.cs.helsinki.fi/u/htoivone/pubs/toivonen_kdd2011.pdf) , last updated 2011, accessed 28 february 2015

[6] G. Buehrer and K. Chellapilla, A Scalable Pattern Mining Approach to Web Graph Compression with Communities  
<http://wsm2009.org/wsm2008.org/WSDM2008-papers/p95.pdf>, last updated 2008, accessed 28 february 2015

[7] Wikimedia Foundation inc, Graph theory  
[http://en.wikipedia.org/wiki/Graph\\_theory](http://en.wikipedia.org/wiki/Graph_theory), last updated february 2015, accessed 2 march 2015

[8] Vreda Pieterse and Paul E. Black, Sparse graph,  
<http://xlinux.nist.gov/dads//HTML/sparsegraph.html>, last updated 14 August 2008, accessed 2015-04-17

[9] Arun Chauhan, School of Informatics and Computing - Indiana University, Graph algorithms,  
<https://www.cs.indiana.edu/~achauhan/Teaching/B403/LectureNotes/10-graphalgo.html>, last updated 2011-01-12, accessed 1 april 2015



**How to Cite:**

**Poonam and Dr. Chitamani Tiwari (Dec 2018). A complete compressing sparse graphs in dijkstra's shortest path algorithm**

*International Journal of Economic Perspectives*,12(1), 137-153.

Retrieved from <https://ijeponline.org/index.php/journal/article>