

**On Sparse Spanners of Weighted Graphs**By I. Althofer, G. Das, D. Dobkin, D. Joseph, and J. Soares  
1993

Mini Project on Low Distortion Embedding | Prof. Michael Elkin

Ayelet Kalderon | ayeletk@post.bgu.ac.il | 200980357

Aviv Oron | avivor@post.bgu.ac.il | 302865134  
January 2018

Table of Contents

[Introduction 2](#_Toc502653221)

[Meaning of a r-spanner 2](#_Toc502653222)

[Motivation 2](#_Toc502653223)

[Algorithm 3](#_Toc502653224)

[Input & Output 3](#_Toc502653225)

[Steps 3](#_Toc502653226)

[Implementation 4](#_Toc502653227)

[Experiments 5](#_Toc502653228)

[Experiment 1 – number of edges 5](#_Toc502653229)

[Experiment 2 – weight of the graphs 8](#_Toc502653230)

[Experiment 3 – average and maximum stretch 11](#_Toc502653231)

[Conclusions 12](#_Toc502653232)

[Future Work 12](#_Toc502653233)

[Bibliography 13](#_Toc502653234)

# Introduction

As part of the course “**Mini Project on Graph Embeddings**”, we chose the paper   
“**On Sparse Spanners of Weighted Graphs**” which was written by I. Althofer, G. Das, D. Dobkin, D. Joseph, and J. Soares, written in 1993.

In the paper, a greedy algorithm is introduced, which gets as input a weighted undirected graph and gives out a constructed r-spanner for the given graph, in polynomial time by the number of edges.

In our project, we will implement the algorithm described in the paper, test by executing with different parameters values (stretch-factor, number of vertices, density and edges weights) and explain the results.

### Meaning of a r-spanner

Let be a connected n-vertex graph with arbitrary positive edge weights. A subgraph is a r-spanner of if between any pair of vertices , the distance in is at most r-times longer than the distance in . The value of the stretch factor associated with .   
The provided algorithm constructs sparse spanners with a constant stretch factor independent of the size of the graph.

### Motivation

Sparse spanners have great use in various areas, especially in distributed systems, for example in the analysis of arrow distributed queueing protocols.   
Another use is in communication networks, for message routing: In order to maintain succinct routing tables, efficient routing schemes can use only the edges of a sparse spanner.

# Algorithm

### Input & Output

The algorithm’s input is:

- Weighted undirected graph

r – Stretch factor (

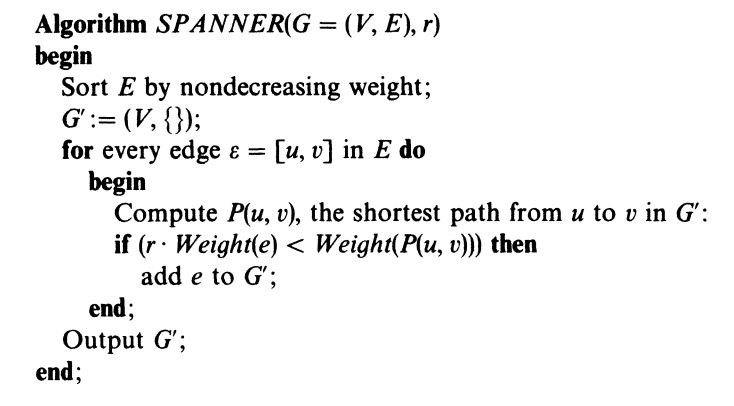
The algorithm’s output is a subgraph which has the same vertices, , and a subgroup of edges, .

### Steps

At first, the algorithm sorts the edges in by non-decreasing weights order.  
In addition, it initializes , the graph which will be reconstructed during the algorithm execution.

The core part of the algorithm has iterations, one for each edge in .  
In each iteration, an edge is being inspected by the following rule –   
If :

\*  *– shortest path from to in* .

  
(The algorithm’s pseudo-code, taken from the paper)

# Implementation

Our final programming language for the algorithm implementation was Java. We wrote our implementation in OOP approach in order to make the code easier for testing and understanding.

We first implemented the basic bricks as separated java classes: “Vertex” and “Edge”.

Following, was the “Dijkstra” class which contains the logic of finding shortest path in graph from source vertex to destination.

The next class we defined was “Graph” which is an object contains lists of vertices and edges, and various methods such as: sorting the graph’s edges, executing Dijkstra on the graph and computing spanner graphs.

Within the *Graph* class we also implemented the PRIM algorithm for finding MST, which will be used in our experiments step of the report.

Our main class, using all of the above, is “Experiments”.

In this class, we’re supplying user-friendly interface which in the user can type the input (number of vertices, edges density), and then choose one of three given tests which will be described in the next part of the report.

# Experiments

In our experiment, we would like to study the new spanner that we generated.   
There are number of parameters that could manipulate our spanner:

1. Number of edges -
2. Number of vertices -
3. R-factor (number between 2 to log(n)) –
4. Weight of the edges (random number between 1 to 100 or uniform) –
5. The probability of having a single edge (between 0.1 to 0.9) -

Our goal is to examine how these parameters affect the spanner’s characteristics and checking the theoretical bounds presented in the article in compared to our experiment’s results.

### Experiment 1 – number of edges

According to the article, the number of edges of the spanner has an upper-bound: , where G’ is the r-spanner. Our goal in this experiment is to examine how much the number of edges in G’ is close to the upper bound and how the and parameters affect this relationship.

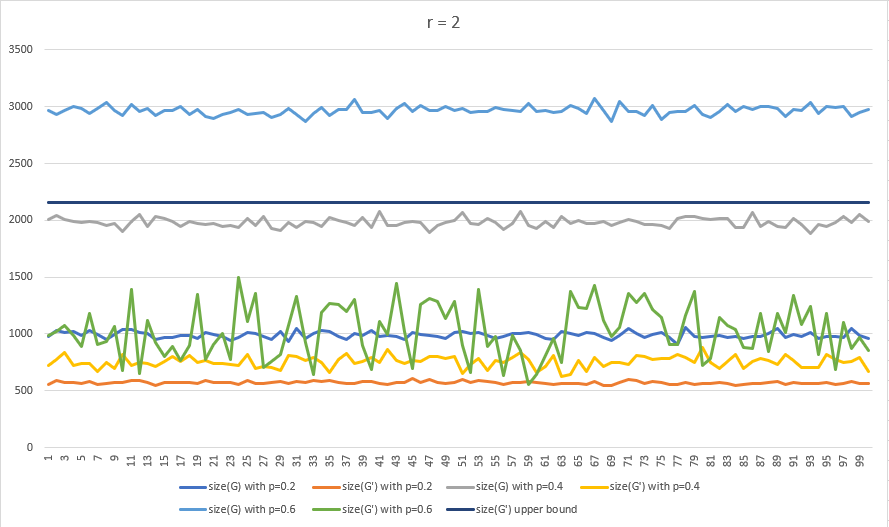
For each r-factor with the values: 2,3,4, we generated 100 G and G’ graphs with 100 verticals and examined 3 values for the p parameter: 0.2, 0.4, 0.6.

The results provided in the next page (figures 1-3) are of the following format: Each figure is for a different r-factor, the 𝑥 axis represents each graph from the 100 graphs, the 𝑦 axis represents the density of the graphs. Note that the weight of each edge is selected randomly between 1 to 100.

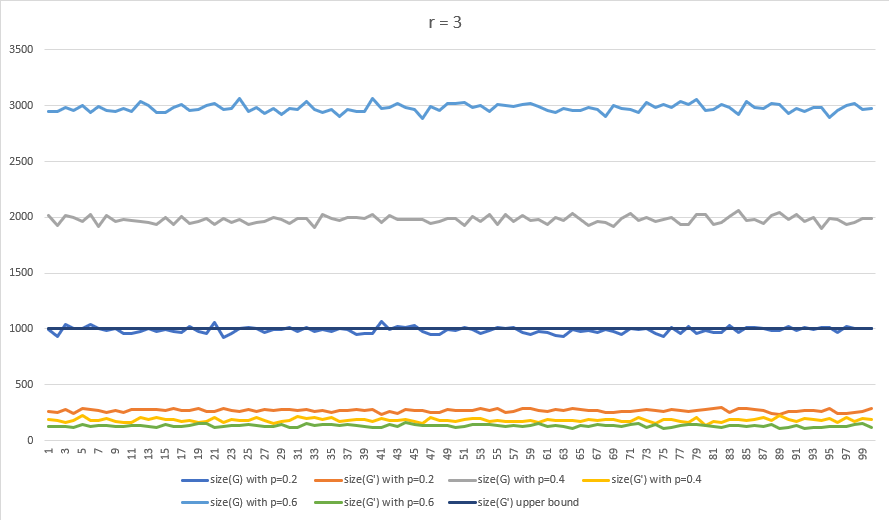
The experimental results are as follows:

1. Spanners generated by the algorithm are indeed have smaller number of edges than the original graphs. We can see clearly that for any value of r or p, We also identified that although 𝑠𝑖𝑧𝑒(𝐺) increases with the value of 𝑝, the 𝑠𝑖𝑧𝑒(𝐺′) stays approximately in the same constant range for any 𝑝 value.
2. The number of edges in the spanners is always greater than n-1 which is the number of edges in the MST of the original graph. This fact is reasonable since spanners with number of edges lower than n-1, are not connected.
3. decreases with the increase of both parameters: p and r. As r and p are getting bigger, the number of the spanner’s edges is getting smaller.
4. We can see that the empirical results of size(G’) are much smaller than the theoretical upper bound. This finding strengthens the theorem that the spanners generated by the algorithm are sparse.
5. Since the upper bound decrease when r increases, we can see that approaches the upper bound with the increase of r.

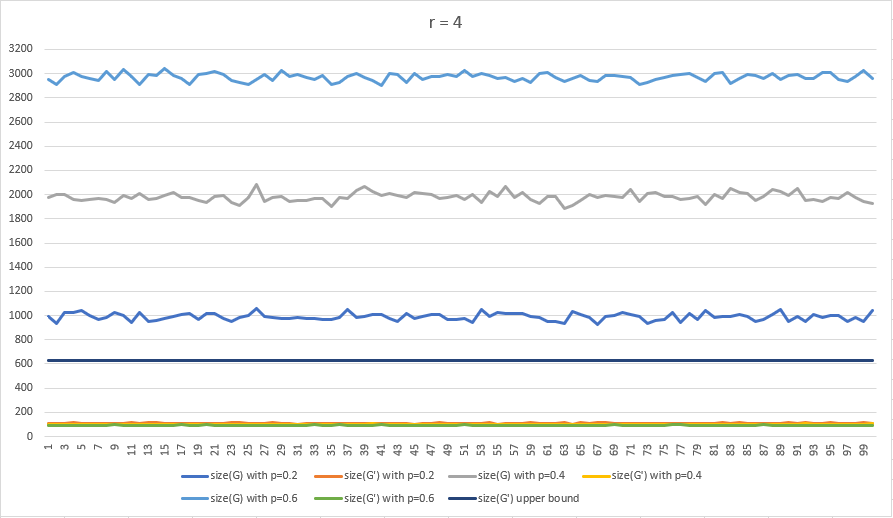
Figure



Figure



Figure



### Experiment 2 – weight of the graphs

In this experiment we will examine the second criteria for being considered as a sparse spanner: weight. According to the article, the upper bound of the spanner’s weight is: , where G’ is the r-spanner and .

Our goal in this experiment is to examine how much the spanner’s weight is close to the upper bound and how the affect the spanner’s weight.

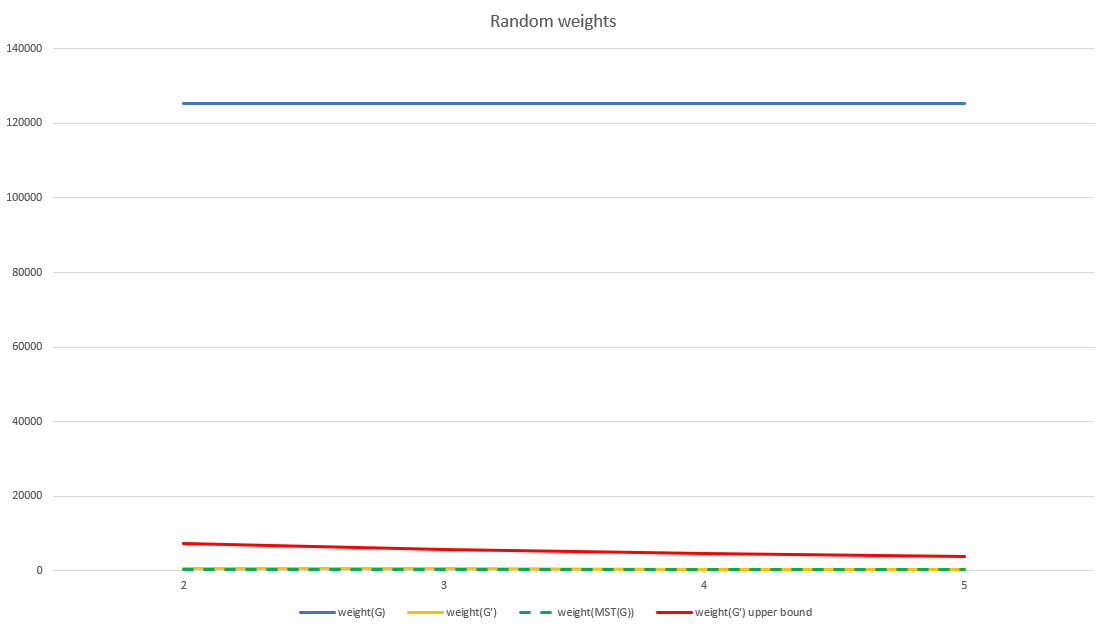
For each r-factor with the values: 2,3,4,5 we generated 100 G, G’, MST(G) graphs with 100 verticals and used so we will have a connected graph for the MST’s creation. Also, we wanted to check whether the weight of each edge has an influence of the total graph’s weight. Therefore, we performed this experiment twice: once with random weights (1-100) and once with uniform weights (1).

The results provided in the next page (figures 4-5) are of the following format: The 𝑥 axis represents the r-factor, the 𝑦 axis represents the average weight of the graphs. Figure 4a includes graphs whom their weight of each edge is selected randomly between 1 to 100 and figure 5 includes graphs whom their weight of each edge is 1 (figure 4b is a closeup to figure 4a).

The experimental results are as follows:

1. Weight(𝐺) is significantly greater than weight(𝑀𝑆𝑇(𝐺)) and weight(𝐺′), as both of them are subgraphs of 𝐺.
2. Weight(𝐺′) decreases when the value of r-factor increases, and approximately converges to weight(𝑀𝑆𝑇(𝐺)).
3. Weight(𝐺′) is always smaller than the upper bound.
4. We can see the same tendency whether each edge’s weight is uniform or random. Although, the tendency in much more clear and sharp in the graphs with the uniform weights.

Figure a



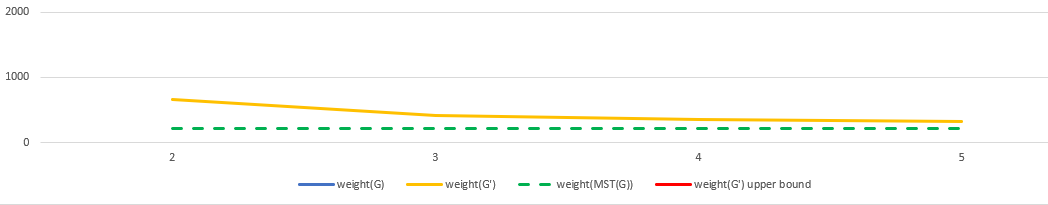
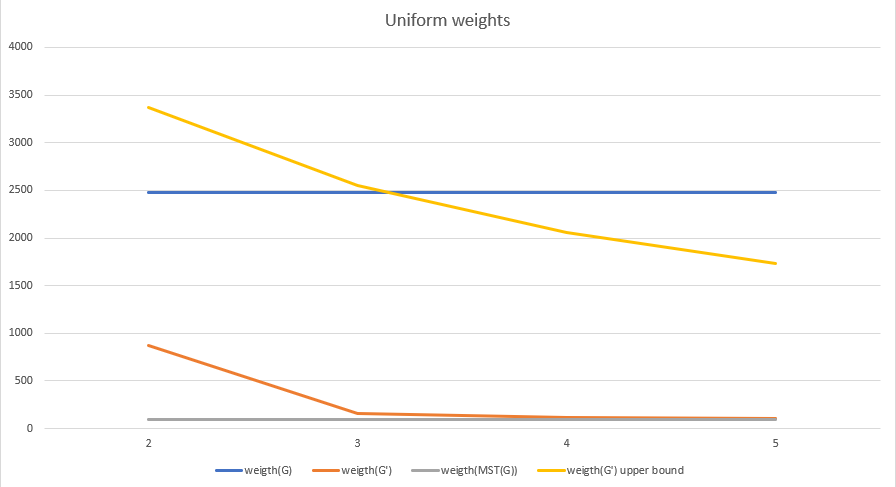
*Figure 4b*

Figure 5



### Experiment 3 – average and maximum stretch

Let stretch number be

Our goal in this experiment is to examine the size of the average stretch and how the affects the maximum and average stretch numbers.

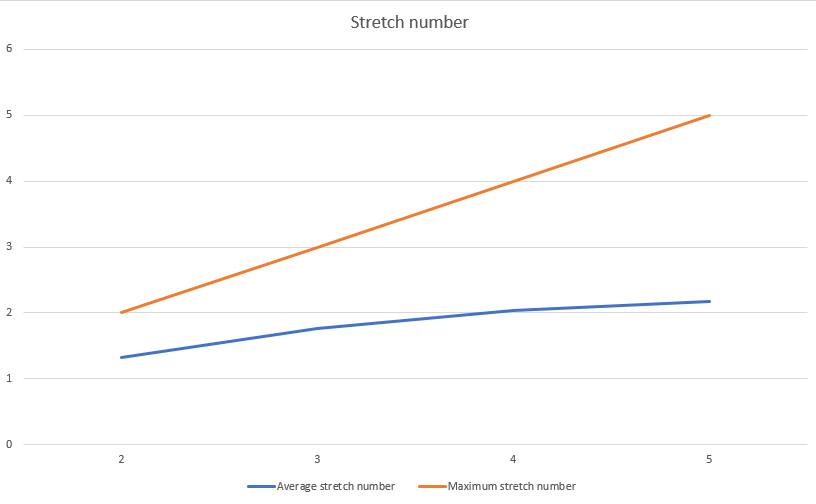
For each r-factor with the values: 2,3,4,5 we generated 100 G and G’ graphs with 100 verticals. Also, we chose uniform weights for all edges and probability of having a single edge, so we could see a greater stretch number.

The result provided in the next page (figure 6) is of the following format: The 𝑥 axis represents the r-factor, the 𝑦 axis represents the stretch number.

The experimental results are as follows:

1. The average short distance does not significantly change between the original graph and the spanner. Although, we can see that the average short distance in the original graph is a bit grater than the average short distance in the spanner.
2. The maximum and average stretch numbers increase with the increase of the .

Figure 6



# Conclusions

After viewing the experiments results, we can clearly tell that the given algorithm constructs a valid r-spanner.   
Our experiments results led us to some conclusions regarding the paper:

* In terms of Graph Size, we noticed that the upper-bound presented in the paper, is higher than the empirical graph size we found and that the number of spanner’s edges decreases with the increases of the .
* In terms of Stretch, we found out that the maximum and average stretch numbers increase with the increase of the .
* In terms of Graph Weight, spanner’s weight decreases with the increases of the , and gets closer to weight of 𝑀𝑆𝑇(𝐺).

However, we will note that because of hardware limitations in our personal PC’s, we were not able to test the algorithm with large enough graph input.

# Future Work

In regarding to our conclusions, the first thing we will do in a future work is to examine the algorithm on much larger inputs, in order to get better results which will might change the conclusions. This can be achieved easily with the proper environment, using distributed computation,

Another thing future work could handle is finding tighter graph size bounds, in respect to the bounds given in the paper.

In general, trying to define a non-greedy algorithm, might construct more efficient spanners with better results.

# Bibliography

* “On Sparse Spanners of Weighted Graphs  
  I. Althofer, G. Das, D. Dobkin, D. Joseph, and J. Soares  
  1993
* SOFSEM 2013: Theory and Practice of computer science.   
  Peter van Emde Boas, Frans C.A. Groen, Giuseppe F. Italitano, Jerzy Nawrocki, Harald Sack (Eds.)