SparseNets vs Minimum Spanning Trees

Abhinav Madahar*, James Abello Monedero†
Rutgers University, New Brunswick, NJ
Email: *abhinav.madahar@rutgers.edu, †abello@dimacs.rutgers.edu

I. Introduction

Many data sets are graph data sets, and these are getting larger over time. Already, large graph data sets are too large to plot in entirety. For example, the Facebook social network graph has billions of vertices (people) and up to 5000 edges (friendships) per vertex [1]; plotting this graph in full would be impossible and the results would be unintelligible. Instead, we can use an algorithm to ignore the minor details of the SparseNet to focus on the large-scale structure of the graph. Some algorithms do this for a connected graph by creating a minimum spanning tree (MST). A spanning tree of a connected graph graph is a subgraph which connects all the vertices in that graph; a minimum spanning tree is a spanning tree where is the sum of weights is minimal. Many algorithms exist to calculate the MST of a connected graph, Kruskal's Algorithm being the most popular.

Alternatively, we can use a SparseNet. A SparseNet connects the vertices of a connected graph by finding the shortest paths between far-away vertices, iteratively adding them until all the vertices are either in the SparseNet or adjacent to a vertex which is. A SparseNet, like an MST, connects the vertices in a connected graph while excluding unnecessary information, but the results can look slightly different. In this paper, we will compare how tightly an MST connects the vertices in a connected graph with how well a SparseNet does so.

A. Dataset

For our work, we generate graphs of different random sizes. We then test the two algorithms on the same graph to compare how tightly they connect it.

B. Algorithm

The SparseNet algorithm considers a connected graph. We first create a distance matrix for the graph using the Floyd-Warshall algorithm. Then, we select the two vertices which are farthest apart from each other. If there is a tie, we prefer the pair which is lexicographically lesser; for example, (2, 1) is better than (3, 4). We then find a shortest path between the two vertices using Dijkstra's algorithm. After we find the shortest path, we use the distance matrix to find the vertex which is as far from the first path as possible. Then, we find a path from the vertex to the vertex on the path closest to it. After that, we repeat, finding another vertex far away from the two paths and a third path to connect it. We repeat until every vertex is either on a path or adjacent to a vertex which is. The final set of paths forms the SparseNet.

TABLE I
Distances between vertices in subgraphs

Algorithm	Median distance	Mean distance	Maximum distance
SparseNet	7.9375	7.81203295	16.925
MST	7.8375	7.89613201	18.2125

II. Methods

To compare the two algorithms, we ran them on the same dataset of randomly-generated graphs. We used NetworkX to generate the graphs and run graph operations. We used Matplotlib to plot the results.

III. Results

We can consider how tightly the two algorithms connect the vertices in a multitude of ways.

First, we look at all the pairs of vertices (u, v) in the original graph whose distance is the diameter of the graph. We would like the distance between them in the SparseNet and MST to be the same, the diameter. However, that is not necessarily the case, so we instead simply prefer the distances to be as low as possible. In Figure 1, we plot the distance between u and v in the SparseNet and MST of a graph where u and v have diameter distance in the original graph. As you can see, sometimes the SparseNet was better and sometimes the MST was better. We can summarize the plots in Figure 1 with a number by considering the difference of the definite integrals of the curves. If the SparseNet curve has a lower integral than the MST curve, then the SparseNet was better for that graph, and *vice versa*. Running this for $n = 80$ trials where the graph had 200 vertices and 1000 edges, we saw that the SparseNet had a lower definite integral 82.5 percent of the time, so the SparseNet is superior in this regard. This means that the SparseNet is better at tightly connecting the most distant vertices when a graph is sparse, which is an important measure in some applications. When we made the graphs more dense, we saw that the difference between the SparseNet and MST was negligible.

After running the two algorithms on the same graph with 200 vertices and 1000 edges, we took the median, mean, and maximum values of the distance matrices. We did this for $n = 80$ trials, all with different randomly-generated graphs, and took the mean of those three statistics across all the trials, shown in Table 1. As you can see, the SparseNet tended to have a higher median distance, but the mean was lower, as was the maximum distance.

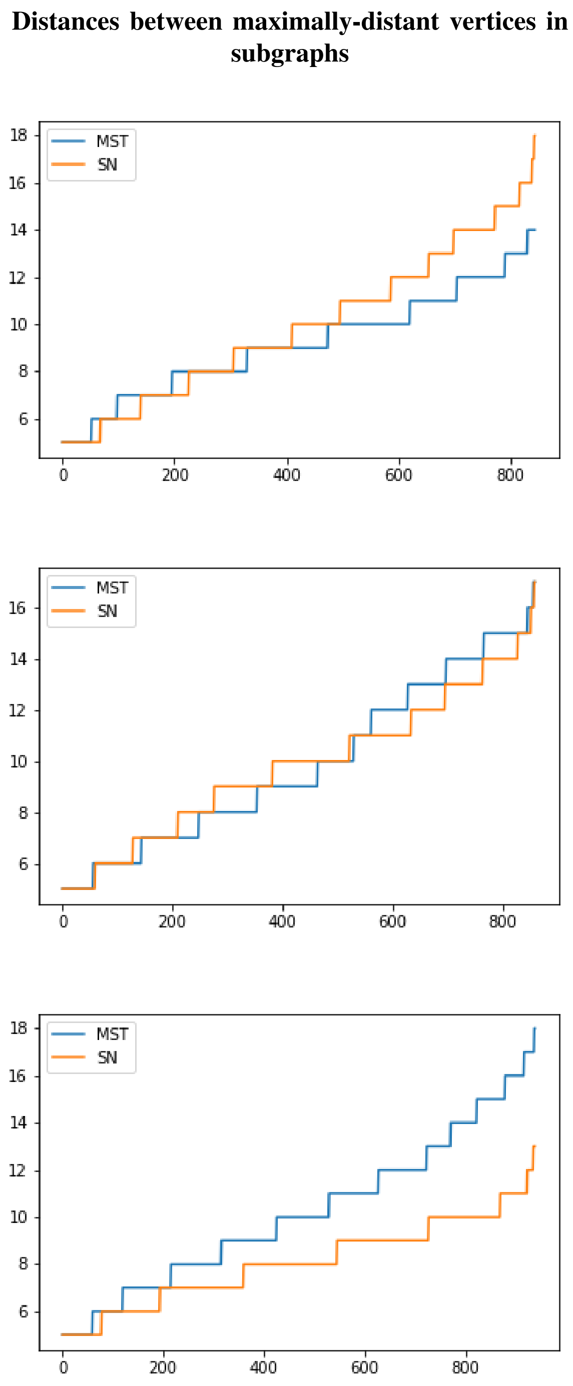

Fig. 1. Here, we plot the distances in the MST and SN between all pairs of vertices u and v whose distance in the original graph is diameter-length. Ideally, the vertices would be connected using short paths, so a lower line in this plot means that the algorithm more tightly connected the vertices. As you can see in these example plots, the SparseNet did not always connect the vertices more tightly than the MST.

IV. Future Work

In the future, we want to consider how the SparseNet compares with other algorithms. We also want to compare it to other data sets from the real-world, particular those which are very large. We also can evaluate how quickly the algorithms run to get a better sense of time complexity.

For very large graphs, running Floyd Warshall and related algorithms to find the distance matrix would use too much memory space. Instead, we can use parallel BFS, which uses less memory, making it better suited for very large graphs.

We also had a large amount of code written in preparation of other, larger projects which could not be finished in the time of this REU. If someone would like to continue this work, then we have posted it publically on GitHub at ttps://github.com/AbhinavMadahar/abello. The source code for our work can be found in the *src* directory, written mostly as Jupyter notebooks. Our code is written for Python 3.

V. Acknowledgements

I appreciate the advice given to me by Prof. Abello and the help given by other members of the lab, including Fatima Al-Saadeh.

VI. References

[1] Statista. 2020. Facebook: Active Users Worldwide — Statista. [online] Available at: ¡https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/¿ [Accessed 10 September 2020].