CS 125 Problem Set 2 Report

Lilly Shen, Tiffany Wu, Andrew Wong-Rolle

September 19, 2014

Disclaimer: Salil approved our 3 person group due to the unfortunate circumstance of Andrew's partner dropping the class.

1 Purpose:

We implemented an algorithm to find the minimum spanning tree given a randomly generated output in the unit n-cube, where $n = 2, 3, 4$ (the 2-cube is a square).

2 Procedure and Choices:

We chose the language C mainly for convenience, but also for the dynamic memory allocation properties.
The randomly generated graph was seeded using the time in srand(time(NULL);, after which it generated numpoints number of points as vertices, and we populate a graph using the euclidean distances as edgeweights.

We used Prim's algorithm seemed cleaner to implement, and given the edge-density, has a better expected runtime ($O(E + VlogV)$ vs $O(ElogV)$) if you use a fibonacci heap. Our implementation isn't as efficient, but we do think that Prim's is generally better for edge-dense graphs.

We then used an adjacency matrix to store the edge weights. Since we're dealing with complete graphs, it wouldn't be more efficient to implement adjacency lists if we took into account every edge. In one of our attempts, we tried to throw away edges and use adjacency lists, but this was not successful and ended up being slower than our "naive" implementation. We used an array Q where the index corresponds to the vertex number and each cell stores the vertex's current distance from S, the current tree being built.

3 Conjectures about $f(n)$:

We honestly did not have many preconceived notions about how the weight of the MST would grow as n increased. We thought it would eventually level off for some cases as the points became so closely distributed that additional points would not contribute significantly to the MST, but that turned out not to be true.

4 Obstacles and Oddities:

One of the challenges was balancing tradeoffs. In a theoretical sense, our attempt where we used adjacency lists and threw out heavy edges probably should have been faster than the naive implementation, but in the end, it was sluggish. To efficiently implement that idea probably would have required more careful choices. In our very first and second attempts (one using a weight matrix and another using adjacency lists), we tried to use a priority queue containing with sort key "distance from S". But after each round of decrementing the remaining distances, we had to build the heap again since everything became out of order. We realized that this made the priority queue no more efficient than a naively implemented array of static vertex distances.

During the coding process, we realized how difficult it was to throw away or separate ourselves from the work we had done so far. We overcame this by messing up many times and having to recognize that what we were doing just wasn't working efficiently.

The random number generator was seeded with the time, which is known to be not actually very random and fairly predictable. If you also happen to run two things within a small time interval (the time between two clock 'ticks'), it seeds the same way. Luckily, this shouldn't be a huge issue for us.

5 Data

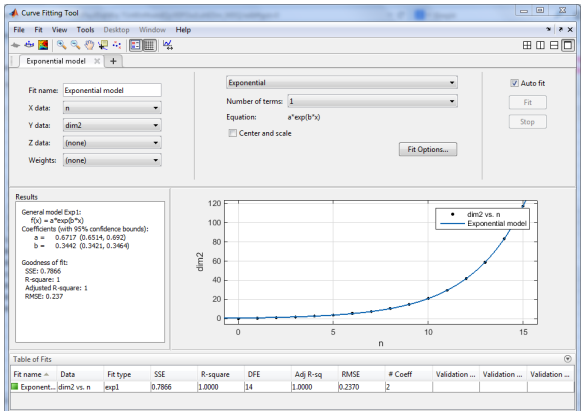
n	0-d	2-d	3-d	4-d
1	0	0	0	0
2	0.497424	0.521045	0.658039	0.776578
4	0.888562	1.123707	1.533853	1.872455
8	1.084877	1.82838	2.750332	3.539551
16	1.154542	2.707063	4.492899	6.134174
32	1.176956	3.863098	7.153409	10.321494
64	1.193776	5.440154	11.252183	17.150188
128	1.195756	7.610541	17.634075	28.464687
256	1.202955	10.66954	27.66396	47.096104
512	1.204001	14.952843	43.313686	78.231117
1024	1.209746	21.048546	68.017227	130.223969
2048	1.202435	29.638803	106.853149	216.420975
4096	1.206262	41.649773	169.069992	360.659241
8192	1.202442	58.878613	267.81543	603.404968
16384	1.200816	83.206734	422.173981	1008.160034
32678	1.204628	117.407021	NaN	NaN

6 Curve Fitting

We found that for dimensions 2,3, and 4 that the average weight of the random minimum spanning tree grew in exponentially compared with the log_2 of the input. We plotted our data in MATLAB and found that the best-fit curves were of the form ae^{bx} where a and b varied directly with the dimension.
Our best guess of an interesting relation between dimension and $f(n)$ occurred for dimension 2, where $f(n) = 0.677 * 2^{\log_2(\sqrt{n}}$.
However, none of the other equations seemed to have an interesting relation in that way.
One of the most interesting relations that we noted was that the MST for randomly generated weights (dimension 0) leveled off at ≈ 1.20.

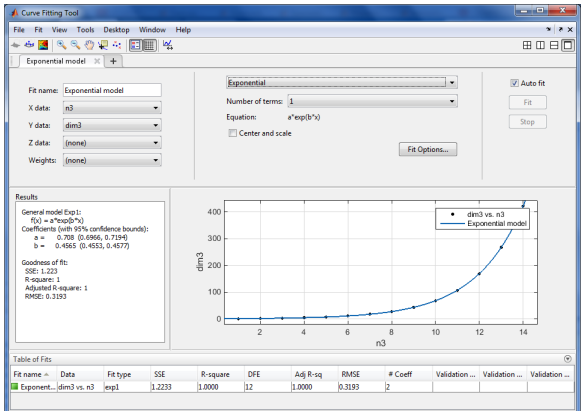
Dimension 2

$$f(n) = 0.677e^{0.3442x}$$



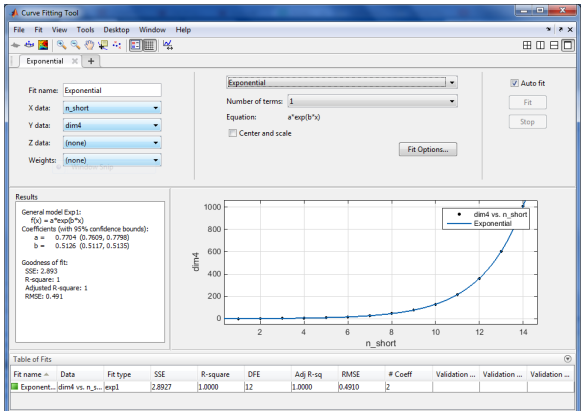
Dimension 3

$$f(n) = 0.708e^{0.4565x}$$



Dimension 4

$$f(n) = 0.77796e^{0.5116x}$$



Dimension 0 (weights randomly generated from 0 to 1)

$$f(n) = 0.677 \cdot 2^{(1/2)\log(x)}$$