ANLY550 Homework 4

Yilun Zhu (yz565) & Siyao Peng (sp1184)

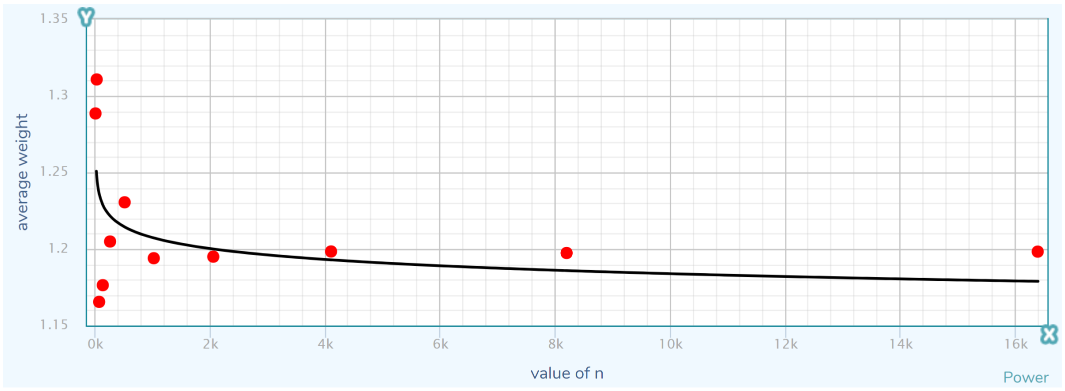
1 Quantitative results

Average MST weights of 5 runs

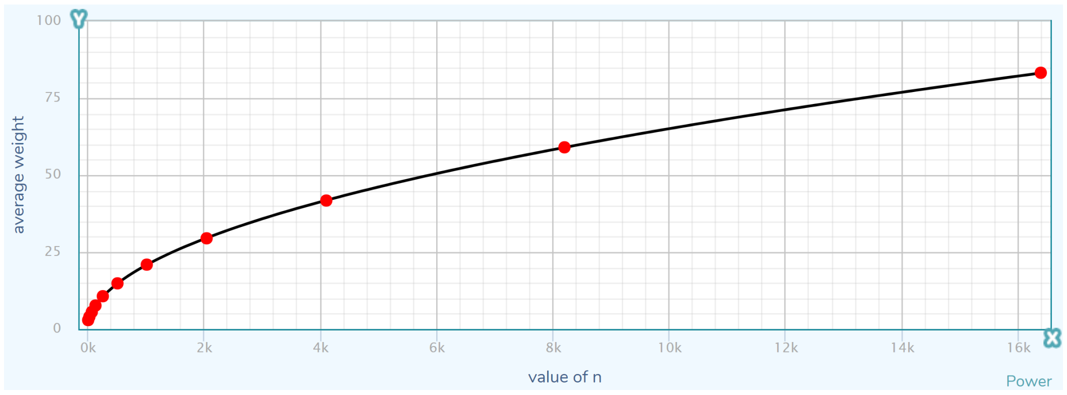
dim\n	16	32	64	128	256	512
0	1.2885	1.3107	1.1655	1.1764	1.2049	1.2305
2	2.9309	4.0064	5.5719	7.6680	10.6818	14.8580
3	4.7609	7.3772	11.2830	17.5525	27.7104	43.3995
4	6.2635	10.3207	17.3958	28.9584	47.5834	78.2025

dim\n	1024	2048	4096	8192	16384	
0	1.1940	1.1950	1.1984	1.1974	1.1983	
2	20.9377	29.4880	41.7523	59.0372	83.2415	
3	67.8099	106.8215	168.9315	267.0506	421.7718	
4	130.3489	216.0683	361.0419	603.5337	1008.19	

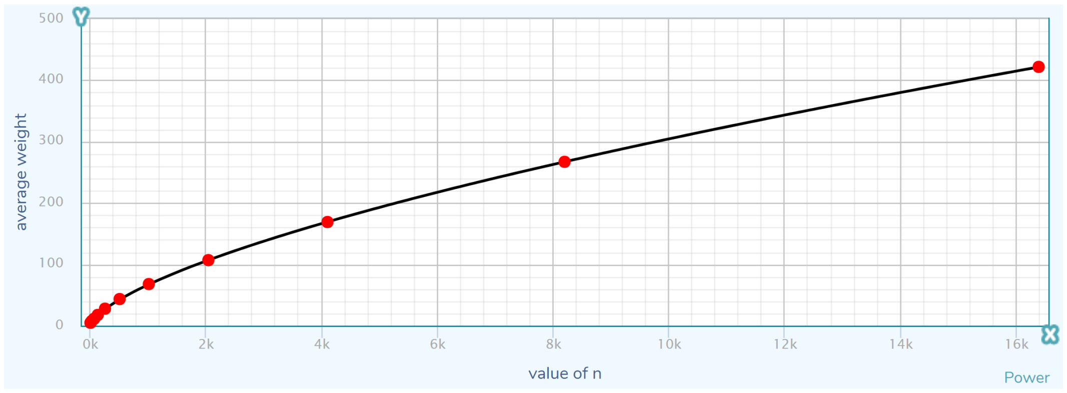
Function $f_d(n)$ To obtain the plot for each dimension, we used the following website (`https://mycurvefit.com/`) to draw the points on a plot. We observed that except when dim=0 the best approximation is the constant 1.2, all other cases are best fitted with power curves. Thus we have the following plots and corresponding approximate $f_d(n)$ where d is the dimension 0, 2, 3, 4.



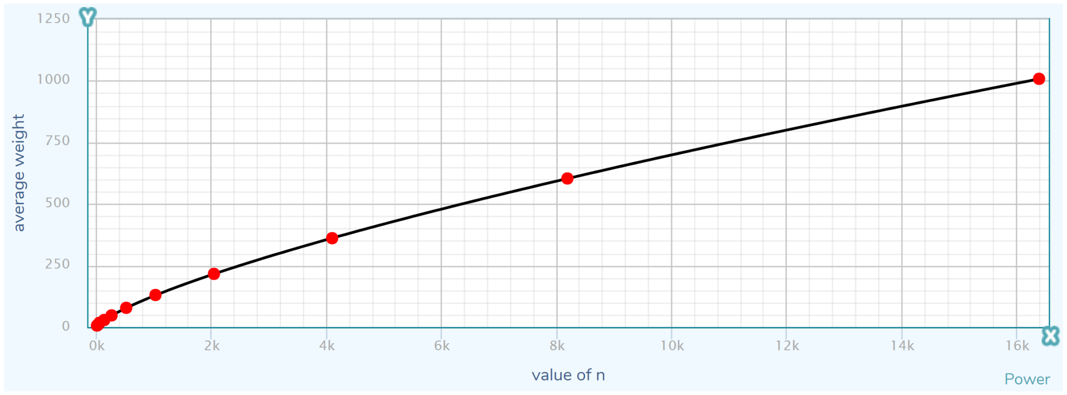
$$f_0(n) \approx 1.2$$



$$f_2(n) = 0.68n^{1/2}$$



$$f_3(n) = 0.71n^{2/3}$$



$$f_4(n) = 0.77n^{3/4}$$

2 Experiments

Cutting edges (Optimization for large *numpoints*) Since the algorithm is run on a complete graph, most edges with large weights are not considered to be part of a minimal spanning tree. We found that with the growing of n, the edge with the maximal weight in a MST increases. The table and figure below present the relation between n and the maximal weighted edge among the *numtrials=5* produced MSTs for each dimension.

dim\n	16	32	64	128	256	512	1024	2048
0	0.4548	0.1515	0.0760	0.0843	0.0230	0.0144	0.0070	0.0051
2	0.3919	0.3264	0.1912	0.1787	0.1097	0.0900	0.0595	0.0432
3	0.6864	0.6864	0.4143	0.3418	0.2986	0.2013	0.1472	0.1266
4	0.6820	0.5731	0.5431	0.4502	0.3643	0.2811	0.2534	0.2323

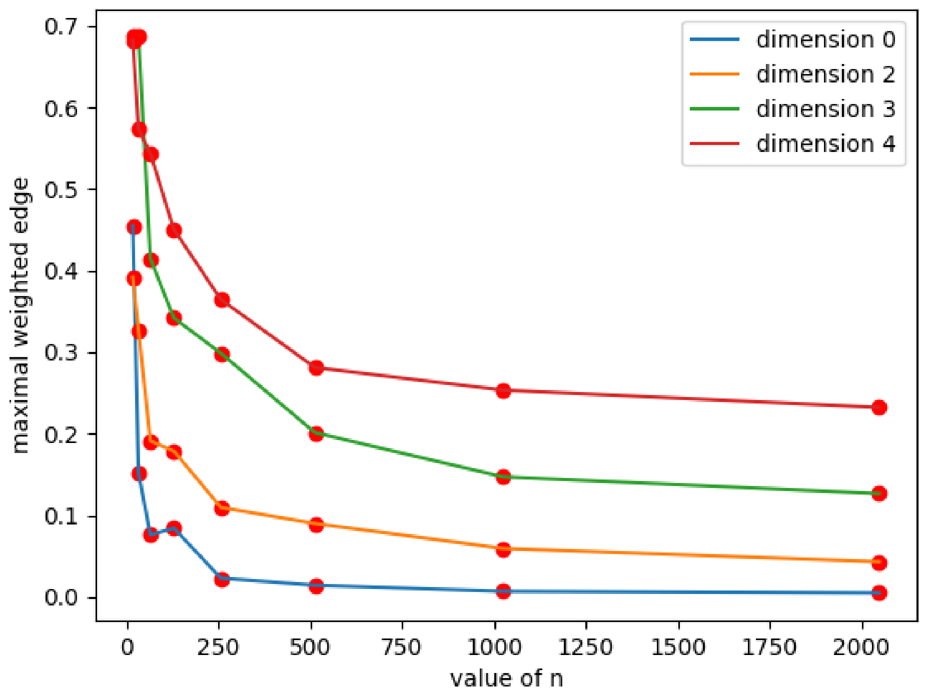


Figure 1: The relation between n and maximal weighted edges on different dimensions

As a result, we set the following threshold functions $g_d(n)$ for the four different dimensions when n is big enough (i.e. $n > 512$): $g_0(n) = 0.01$, $g_2(n) = 0.1$, $g_3(n) = 0.2$, $g_4(n) = 0.3$. We understand that they are really rough constant threhold functions for each dimension (and not really a function of n), however, it does help us to run *numpoints=16384* within less than an hour.

2.1 Which algorithm did you use, and why?

For a graph with V vertices E edges, Kruskal's algorithm runs in O(E log V) time and Prim's algorithm can run in O(E + V log V) amortized time, if you use a Fibonacci Heap. Kruskal's is efficient on sparse graphs while Prim's is better for dense graphs.

We chose the better Prim algorithm because of its faster ideal runtime on a complete (extremely dense) graph. However, due to different reasons, we used `heapq` from the Python Standard Library to substitute the Fibonacci Heap and the algorithm runs slower than the ideal running time.

2.2 Are the growth rates (the $f(n)$) surprising? Can you come up with an explanation for them?

For dimension 0, we are surprised that it approximately equals to a constant function since the number of nodes and edges grows exponentially as a function of n. We observe that when the number of edges in the MST grows exponentially, the extremely small-weight edges also grow exponentially, which indicates that the total weight of the MST may not change substantially.

We are not surprised by the growth rates of $f(n)$ for dimension 2, 3 and 4. Since the number of edges grows exponentially as a function of n, the number of edges in a MST should also grow exponentially. We also found that in $f(n) = an^b$, both a and b are smaller than 1. When the graph has more dimensions, the value of a and b are approaching to 1, though we need more values of n to prove.

$$f(n) = (\frac{dim - 1}{dim})n^{\frac{dim-1}{dim}}$$

The formula indicates that if the dimension is large enough, the average edge is approaching to infinity which is approximately 1. However, we're still not clear about the relation between number of dimensions and number of nodes.

However, we have some preliminary reasonings. Since dim=0 random numbers are true RANDOM numbers so the histogram distribution of the randomly generated weights is constant. For larger dimensions, the Euclidean distance function creates a bell curve and among them dim=2 has the most left-skewed curve whereas the curve for dim=4 is almost symmetric. This also explains why the total weights of a MST is larger for dim=4 than dim=2 (also why the threshold is larger).

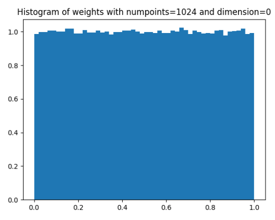


Figure 2: dim=0

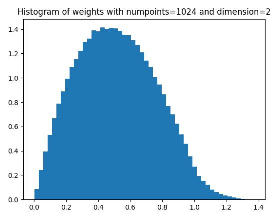


Figure 3: dim=2

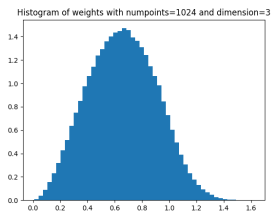


Figure 4: dim=3

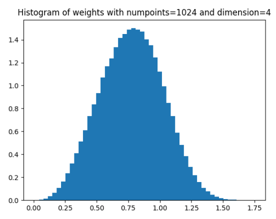


Figure 5: dim=4

Figure 6: Histogram of random weights with 1024 points and 4 different dimensions

2.3 How long does it take your algorithm to run? Does this make sense? Do you notice things like the cache size of your computer having an effect?

Time in seconds The following table shows the runtime (for a total of *numtrials=5* trials) in seconds for different dimensions, numpoints. *dim=0* is almost at most half the time compared to other dimensions (for large n) simply because computing Euclidean distance takes time for *dim=2,3,4*.

The runtime for *dim=2,3,4* are on a larger scale. Runtime for *dim=3,4* do not differ that much. However, we are not sure (and also unexpected) why runtime for *dim=2* is to some extent longer than larger dimensions. One potential reason might be that we do not have a tight threshold for *dim=2* so they are more redundant weights that have been computed.

dim\n	16	32	64	128	256	512	1024	2048	4096	8192	16384
0	4e-4	1e-3	4e-3	0.01	0.05	0.22	0.66	3.27	13.79	60.15	281.69
2	9e-4	2e-3	6e-3	0.03	0.12	0.69	1.36	7.11	33.96	189.51	**1098.83**
3	5e-4	2e-3	6e-3	0.02	0.09	0.59	1.34	6.36	28.11	138.51	689.01
4	9e-4	2e-3	6e-3	0.02	0.10	0.52	1.38	6.31	27.20	125.06	577.65

Caching After setting up the thresholds, we did notice that the required computing memory decreased to some extent. We also found that when $n = 16384$ and $dim = 2$, the running time is extremely slower than other cases when $n = 16384$. However, when $dim = 2$, the cache size should not have significant differences compared with other cases when $dim = 3, 4$. Therefore, we do not think cache sizes but other factors have an effect on our algorithm.