CS 124 Programming Assignment 1 Writeup

Our goal in this experiment was to determine how the expected weight of the minimum spanning tree of four different types of complete, undirected graphs grows as a function of n, where $n = 128; 256; 512; 1024; 2048; 4096; 8192; 16384; 32768; 65536; 131072; 261444$ represents the number of vertices in the graph and each type of graph corresponds to each of the following algorithms:

- Edge weight is chosen uniformly at random on $[0, 1]$
- Vertices are points chosen uniformly at random in the unit square and edge weight is the Euclidean distance between its endpoints
- Vertices are points chosen uniformly at random in the unit cube and edge weight is the Euclidean distance between its endpoints
- Vertices are points chosen uniformly at random in the unit hypercube and edge weight is the Euclidean distance between its endpoints

Before conducting the experiment, we hypothesized that, regardless of the type of graph, expected MST weight would increase with n because the graphs are complete, meaning that all possible $\binom{n}{2}$ edges are present in the graph. Therefore, the existence of more edges would, in theory, increase the number of edges that would be in the MST, and because the weight of the edges are in some way randomly generated and always positive, an MST with more edges would tend to have more weight. Additionally, we expected the average MST weight to grow approximately linearly with n, considering the fact that $\binom{n}{2}$ grows similar to n^2 but only a fraction of edges contribute to the MST's increased weight.

In order to test our hypotheses, we began by implementing, in C++, the procedures that generate each type of random graph, using the mt19937 pseudo-random number generator. Then, with regard to creating the MST for each graph, we initially set out to implement Prim's algorithm in the hopes that it would be easier to implement. However, after implementing it, we found that our algorithm took a relatively long time to run (for instance, the algorithm took 4 minutes to run once for an input of $n = 4096$ vertices), and we figured that the only way to speed up the process would be to implement a heap. We assumed that this would be more difficult than implementing Kruskal's algorithm with a Union-Find, so we ultimately settled on the latter. This first entailed randomly generating all of the vertices/edges and then storing the edges in an array and sorting them in order of increasing weight. Then, to represent the iterative process in Kruskal's algorithm of adding edges to the MST in order of increasing weight, we traversed through the array of edges and used the Union-Find to ensure that the ends of each edge belonged to disjoint vertex sets (which implies that adding that edge would not create a cycle in the MST) and then keeping a running sum of the weights of the least expensive $n - 1$ eligible edges, thus giving the total weight of the MST.

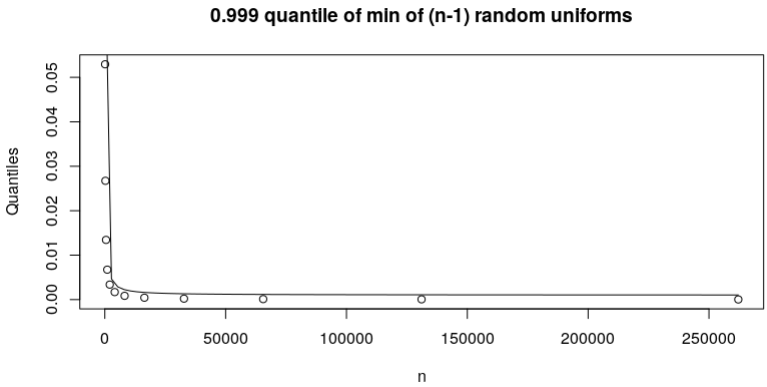
While this worked well for relatively small values of n and was faster than our Prim's implementation (the algorithm now took around one minute to run once for $n = 4096$), we quickly discovered that, for larger n values ($n \geq 16384$), the program would run out of memory. In an attempt to free up space, we changed the types of our edge weights and vertex coordinates from doubles (which take 8 bytes each) to floats (which take 4 bytes), and, instead of keeping all n vertices stored for the duration of the algorithm, we decided to delete a vertex whenever the weights of all of its associated edges were computed. While this helped slightly, we continued to run into space issues for the largest n values. Given that the graphs are complete, and thus all $\binom{n}{2}$ edges are present in each graph, we surmised that the bottleneck with regard to space complexity must be the storage of the edges. Consequently, we set out to find a means by which we could remove certain edges (specifically, those who have a very low probability of being in the MST) from consideration immediately.

We began by printing out the largest edge weights in the MST for as many values of n as we could (i.e. up to $n = 8192$) and for each type of graph 10 times and then keeping track of

the largest one we could find in order to find a reasonable guess for a function $k(n)$, for each graph type, that bounds the weight for an edge in the MST by graphing and inspecting different functions that overestimate the maximum weight. After inspecting the table, we graphed the results, and in doing so, developed a guess for $k(n)$ for the first type of graph (where the edge weights are chosen uniformly at random on $[0, 1]$, which we'll refer to as $d = 0$, where d represents dimension):

$$k(n) = \frac{10}{n} + 0.001,$$

To convince ourselves that this guess was reasonable (i.e. that it would work for all n without mistakenly disregarding MST-eligible edges), and to provide some intuition behind it, we considered the fact that, by the Cut Property, if we were to consider the vertex set $\{a\}$, and another set that contains the remaining $(n-1)$ vertices, the minimum-weight edge connecting those two sets is in the MST. So, of the $n-1$ edges connected to a, the one with the minimum weight must be in the MST. Since the edge weights are uniformly distributed random variables, probability theory tells us that the minimum of the $(n-1)$ edge weights is a Beta$(1, n-1)$ random variable. Taking the 0.999 quantiles of this random variable at the values of n we tested on produced the following graph, which, when graphed against our function, looked reasonably consistent.



If we were to find the exact function that runs through these points we would be given a limit such that there is a 0.999 chance that the weight of the minimum-weight edge will be less than or equal to that value. However, with $(n-1)$ trials (edges), that probability can decrease significantly. Therefore, we healthily over-approximated this value such that we have a close to 1 probability of success. As a result, we were convinced that there was a very low probability that any weights greater than our over-estimated limit would be used in the graph, so, for $d = 0$, if any such weight was produced by the random number generator, we ignored its associated edge and only stored edges whose weights were below our threshold.

Upon inspection of the tables/graphs for the other graph types ($d = 2, 3, 4$), we found $k(n)$ to be:

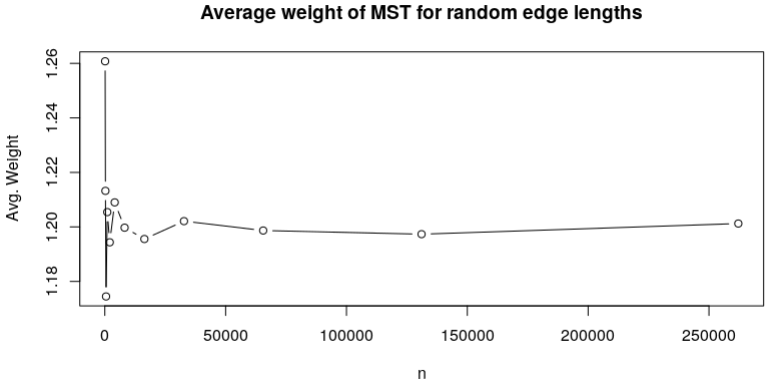
$$k(n) = \frac{d^2 \cdot 10}{n} + d^2 \cdot 0.01 + (d-1) \cdot 0.01$$

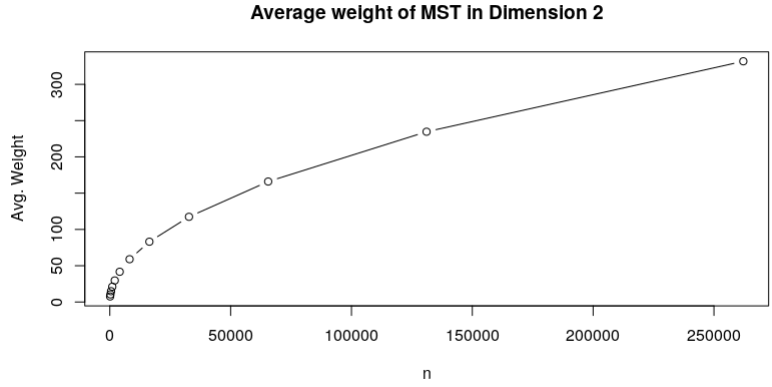
An argument similar to that made above for $d = 0$ graphs could theoretically be carried over to the 2-4 dimensional cases, but in these cases, the edge weights are no longer Unif$(0, 1)$ random variables but functions of Unif$(0, 1)$ random variables, the CDFs and quantiles of which we assumed would be difficult to compute and go beyond the scope of our, admittedly limited, statistics knowledge. Regardless, the function produced a value that was slightly above the all of the largest MST edge weights that we found in every case, so we were convinced. Simliar to above, for $d = 2, 3, 4$, only weights that fell below the threshold set by $k(n)$ were stored, while the rest were ignored.

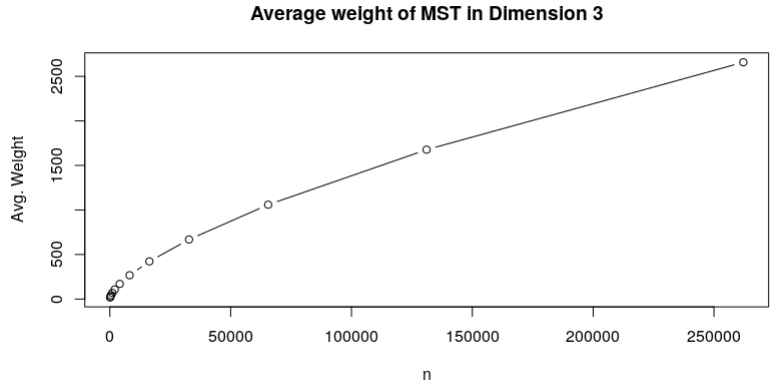
This strategy of disregarding ineligible edges proved to be very effective with regards to both space and time. Now endowed with ample space, we were able to find the total MST weight for every value of n and every graph type (with the longest trial for input $n = 4096$ now taking 0.028 seconds, which we attributed to the algorithm dealing with a significantly smaller input during relatively computationally expensive procedures, such as sorting the array of edges for example). For each n, d combination, we ran the algorithm five times which gave five MST weights, the averages of which are displayed in the table below (where each row represents an n value and each column a d value):

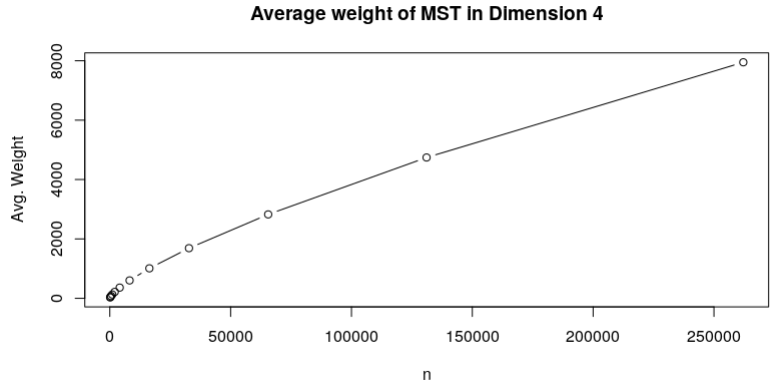
	0	2	3	4
128	1.261	7.648	17.743	28.743
256	1.213	10.543	27.613	46.845
512	1.174	14.902	42.971	78.330
1024	1.205	21.078	68.340	129.269
2048	1.194	29.690	107.087	216.691
4096	1.209	41.674	169.648	360.737
8192	1.200	58.996	267.047	602.683
16384	1.196	83.114	422.624	1010.027
32768	1.202	117.424	668.387	1688.964
65536	1.199	166.031	1059.177	2825.677
131072	1.197	234.671	1676.588	4742.052
262144	1.201	331.748	2657.501	7948.266

Immediately, we noticed that the average MST weight did not grow linearly for each n and d as we hypothesized, but actually grew sublinearly for $d = 2, 3, 4$ and remained essentially constant for $d = 0$. In order to get a clearer idea of the trends for each d value, we also graphed our results, as shown below in order of increasing d:









Upon inspection of the graphs, we observed that the $d = 2$ curve very closely resembled $y = \sqrt{x}$. We also noticed that the curves were becoming more linear as d grew from 2 to 4. These observations suggested a function whose leading term involved n taken to a fractional exponent that approached 1 as d increased. With this in mind, we guessed the following curve forms for the growth rates of each type of graph:

$$\begin{aligned} d = 2&: \ f(n) = n^{1/2} + c \\ d = 3&: \ f(n) = n^{2/3} + c \\ d = 4&: \ f(n) = n^{3/4} + c, \end{aligned}$$

and found that they were all reasonably consistent with our experimental values, giving us a generalized guess for $f(n)$ for each $d \geq 2$:

$$f(n) = n^{(d-1)/d} + c.$$

Of course, for $d = 0$:

$$f(n) \approx 1.20,$$

as all of the values remained essentially constant around that value.

We were fairly surprised to find these growth rates, especially given our hypothesis. However, in retrospect, we realized that a possible explanation behind the lack of growth for the $d = 0$ graphs is the fact that, with a greater number of edges, the chances of an MST-eligible edge being generated whose weight is smaller than those of the edges in the current MST increases, and so the MST weight ends up increasing and decreasing at the same rate, resulting in a constant weight across n. A similar argument does not apply to the $d = 2, 3, 4$ graphs because in this case, the edges themselves are not uniformly generated; they are based on the proximity of the vertices. Therefore, we reasoned that it must be the case that as the number of vertices in the graph increases, the proximity of a given vertex to its closest neighbor also increases (and thus the associated edge weight decreases), and the growth of the proximity must be faster because the space within the unit square, cube, and hypercube is finite. This implies that the rate at which the total MST weight increases with n is decreasing, making the growth sublinear across n.

Ultimately, we found that the growth rates of MST weights for the randomly generated graphs under consideration were much more intriguing than we first thought they would be. Perhaps more importantly, though, in conducting the experiment, we considered trade-offs between simplicity, time, and space (when deciding between implementing Prim's or Kruskal's algorithm), learned about a practical means by which runtime could be reduced and space could be saved (the edge deletion strategy), and developed greater intuition for approximating experimental values with functions in order to create a model (when finding $k(n)$ and $f(n)$).