CS 124 Programming Assignment 1: Spring 2023

Your name(s) (up to two): Nicholas Lyu, Aghyad Deeb
Collaborators: None
No. of late days used on previous psets: Nicholas: 0, Aghyad: 2
No. of late days used after including this pset: Nicholas: 0, Aghyad: 2

Code setup

- *randmst.cpp*: source code satisfying specification: running *./randmst numpoints numtrials dimension* outputs *average numpoints numtrials dimension*
- *runexperiments.py*: run *python3 runexperiments.py* to generate statistics for $D = 2, 3, 4$, $N = 2, 4, 8, ..., 262144$.
- analysis.ipynb: A jupyter notebook performing the analysis for our guess for $f(n)$.

Quantitative Results

- *A table listing the average tree size for several values of n.*

(N, D)	0	2	3	4
2	0.766071	0.495751	0.648872	0.808514
4	0.976677	0.887915	1.46341	2.02087
8	0.920475	1.57052	2.60319	3.30022
16	1.09876	2.49126	4.36537	5.98524
32	1.06875	3.67377	7.18076	10.1032
64	1.22524	5.33149	10.6858	17.1664
128	1.17781	7.65754	17.9341	28.3262
256	1.20004	10.6459	27.8595	47.1526
512	1.16687	14.8572	43.8642	78.6729
1024	1.20322	21.0597	68.6714	129.923
2048	1.20562	29.7164	107.895	217.255
4096	1.2161	41.7654	169.395	360.606
8192	1.19908	58.8814	267.156	603.05
16384	1.20364	83.2029	422.219	1008.36
32768	1.20027	117.442	667.682	1690.05
65536	1.20263	165.954	1057.73	2829.69
131072	1.20284	234.584	1676.37	4741.55
262144	1.2042	331.482	2658.09	7949.02

All results above are averaged over 5 runs.

Discussion

- *Algorithm design*: even moderately large $N \geq 65536$ deems $O(N^2)$ algorithms unfeasible. There are $O(N^2)$ edges in a complete graph with N vertices, so we must somehow operate on a pruned graph.

Consider the following pruning method: given G, we choose $B > 0$ and produce G' by ignoring all edges of weight $> B$ in G. First note that Kruskal's algorithm selects the MST with minimum longest edge. Let T be the result of running Kruskal's algorithm on G, and let its longest edge have weight B'. If $B > B'$, running Kruskal on G' will produce $T' = T$ since Kruskal considers the edges used to construct MST in increasing order of weights, then the result of any MST algorithm on G' will also be a MST on G. If $B < B'$, Kruskal will fail to find a MST on G', in which case there is no MST on $G' \iff G'$ is not connected.

The above analysis motivates the following proposition: given B, if we can find an MST on the pruned graph G', it will also be an MST on G. This essentially proves the validity of our algorithm as follows: given N, D

1. Generate N vertices randomly samples in $[0,1]^D$. Initialize cutoff $\delta_0 = \frac{1}{2} \cdot \frac{\sqrt{D}}{N^{1/D}}$
2. Increase k from 1 until prim's algorithm succeeds: create an adjacency list E containing all edges with weight less than $k\delta_0$, and run Prim's algorithm on graph defined by E.

- *Choice of threshold*: we first note that no computationally-helpful threshold works on all graphs. Since the maximum distance between two points in $[0,1]^D$ is $\sqrt{D}$, any $\delta_0 \geq \sqrt{D}$ is not computationally helpful (does not result in any effective pruning). However, any threshold $\leq \sqrt{D}$ will fail for the following graph: consider one point v_0 at $(0, ..., 0)$ and all other points congregated infinitely close to $(1, ..., 1)$. Pruning will disconnect v_0 from the rest of the graph.

Our choice of threshold δ_0 is motivated by the following intuition: distributing N points inside $[0,1]^D$ is like distributing $N \cdot 2^D$ points inside $[0,2]^D$ (in which case all lengths will be doubled), then increasing N by factor of k^D increases length properties by k, which means length properties scale by $N^{-1/D}$. The choice of constant $\sqrt{D}/2$ is not as thoughtfully justified – it is just half the diagonal.

We further note that a better thresholding algorithm will be possible if the following question can be answered: given $B > 0$, what is the probability that a random graph generated via N, D pruned by B will be disconnected? In general we find this a hard question to answer, as even the $N = 2$ case involving the distribution of distance between pair of randomly sampled points involves nontrivial integral for large D.

In the case where weights are chosen randomly: from each vertex V we have $n - 1$ edges the weight of which is chosen uniformly at random from the range $[0, 1]$. While we recognize that for a given vertex, the edge with the minimum weight isn't necessarily enough to connect it to the entire graph, it's a good starting point for the threshold. Therefore, we can calculate the probability of all edges connecting V having values greater than a specific threshold x:

$$P(E_1 > x, E_2 > x, \cdots, E_{n-1} > x)$$

Given that each edge weight is chosen uniformly at random, these random variables are independent, Therefore:

$$P(E_1 > x, E_2 > x, \cdots, E_{n-1} > x) = P(E_1 > x)P(E_2 > x) \cdots P(E_{n-1} > x) = (1 - x)^{n-1}$$

We can see that for $n = 1000$ even a very small x will suffice to get a very low probability of not

reaching an edge. Choosing $x = 0.3$, for $n = 1000$ we get $P \approx 1.253x10^{-155}$. For larger values for n, we can decrease x. In practice, we found best results (minimum number of threshold increments while still giving relatively fast results) when choosing $x = \frac{100}{N}$ and doubling it upon failure (which grows exponentially).

- *Component time-analysis and parallelism*
  We note that the MST-finding component of our algorithm is surprisingly fast – the bottleneck turned out to be generating the adjacency matrix, which requires doing N^2 computations. This remains the bottleneck even after we've introduced shortcuts such as ignoring an edge as soon as the kth component $|v_{i,k} - v_{j,k}|$ differs by more than threshold δ.
  We mitigated this issue by splitting edge-distance computation into chunks: e.g. given $N = 1000$, we may split into 10 parallel threads each computing edge-statistics of edges $0 \sim 99, 100 \sim 199, ...$

- *Asymptotic runtime*
  A rigorous runtime analysis will not be in order without knowing the connectedness-threshold distribution discussed above. As such, $O(N^{1/D} \cdot N^2 \log N)$ provides an upper bound on the runtime of our algorithm since the largest effective threshold is $\sqrt{D}$ and Prim's runtime with heap implementation is at worst $M \log M = N^2 \log N$. We finally note that this is an extremely crude bound since this does not account for pruning savings nor the fact that most of the edges do not enter the heap prior to MST being generated. In practice we find that Prim's algorithm runs much faster than pruning (which takes $O(N^2)$).

- *Guess for $f(n)$ and Chi-Squared-Analysis:*
  We've performed analysis on the data point to guess the functions and assess the accuracy of our guesses. First of all we have made the following plots for each of value of D:

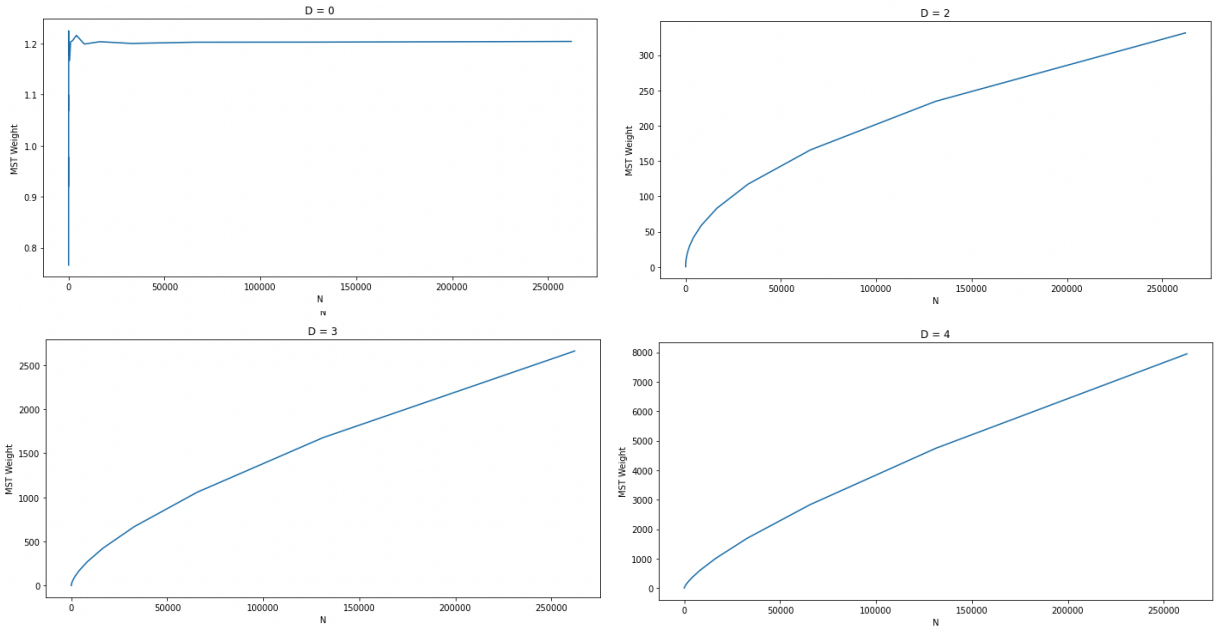

Figure 1: Graphs for $D = 0,\ D = 2,\ D = 3,\ D = 4$, respectively

From the graphs, it seems like the function $f(n)$ becomes more and more linear with respect to n as the dimension D increases. When $D = 2$, it looks like a square root function.

Given this observation, we guessed that:

$$f(n) = \begin{cases} a * n^{(D-1)/D} + b & ; D \neq 0 \\ a + b & ; D = 0 \end{cases}$$

Where a, b are constant we will later determine.
In order to measure the accuracy of our guess, we decided to use Chi-Squared-Analysis to get a robust score for accuracy. We used the standard deviation for our values as the error.

```
D =  0 :
Param 0: 0.317730 +/- 0.123026
Param 1: 0.817730 +/- 0.123026
0.026501673898985702

D =  2 :
Param 0: 0.647678 +/- 0.001261
Param 1: 0.095487 +/- 0.215263
0.11036397693213189

D =  3 :
Param 0: 0.648935 +/- 0.000916
Param 1: 1.691638 +/- 1.139078
3.4458047639449894

D =  4 :
Param 0: 0.686655 +/- 0.001461
Param 1: 5.656836 +/- 5.026814
73.80446693574842
```

Figure 2: Chi-Squared values for the predicted function $f(n)$

As we can see, our chi-squared values are pretty close to 1 for Dimensions 0, 2, and 3. Which shows that the model is fairly accurate for these dimensions. For $D = 4$, we got a high chi-squared value indicating that either the guess for function doesn't accurately describe what's happening or that we've underestimated our error.