CS 1240 Programming Assignment 1

Kathy Jia & Hanjing Wang

February 18, 2026

1 Quantitative Results

1.1 Table of Results

n	Dim 0	Dim 1	Dim 2	Dim 3	Dim 4
128	1.1986	20.0752	7.8473	18.0895	28.8345
256	1.1209	34.1532	10.7753	27.6187	47.1073
512	1.2365	66.1920	15.1069	43.1719	79.2117
1024	1.1942	116.4278	20.9311	68.0096	130.1026
2048	1.1986	215.5197	29.5082	107.4619	216.0709
4096	1.2114	387.7399	41.8154	168.9980	360.4264
8192	1.2157	724.0368	58.8329	267.2348	602.0764
16384	1.1945	1338.8172	83.2863	422.2005	1008.0576
32768	1.2079	2518.5486	117.4832	669.1275	1687.5327
65536	-	4734.8078	-	-	-
131072	-	8935.7861	-	-	-
262144	-	16868.9239	-	-	-

Table 1: Average MST Weight by Dimension and Number of Points (n)

1.2 Asymptotic functions

We give.

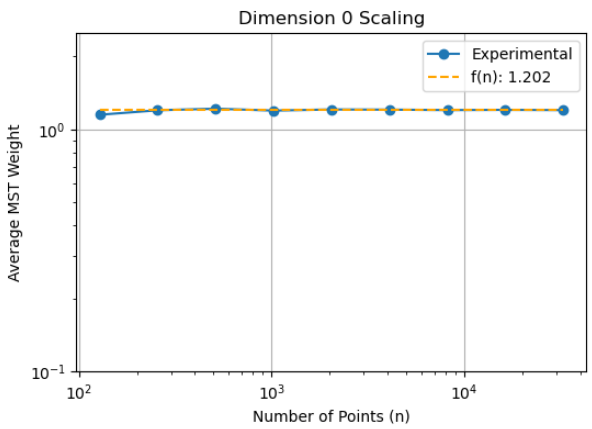
- Dimension 0: $\boldsymbol{f(n) = 1.2}$

  Justification: we know that the graphs in dimension 0 are complete, so there are $\frac{n(n-1)}{2} = O(n^2)$ edges and each of them are chosen uniformly from the interval $[0, 1]$. We note that there exists a constant $c \cdot n$ such that if we choose a set of $c \cdot n$ edges, we're almost certain to be able to build an MST for the graph. Then, supposing that we choose the smallest $c \cdot n$ edges and using order statistics, the weight of the largest edge in this set would be

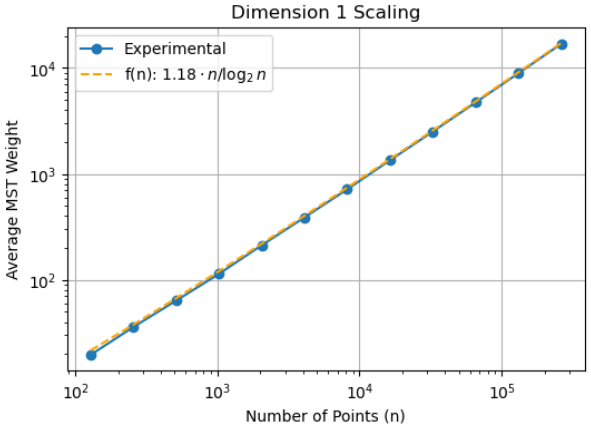
  $$\frac{cn}{n(n-1)/2+1} = \frac{2cn}{n(n-1)+2} = \frac{2cn}{n(n-1)+2}$$

  Thus, the weight of the MST is

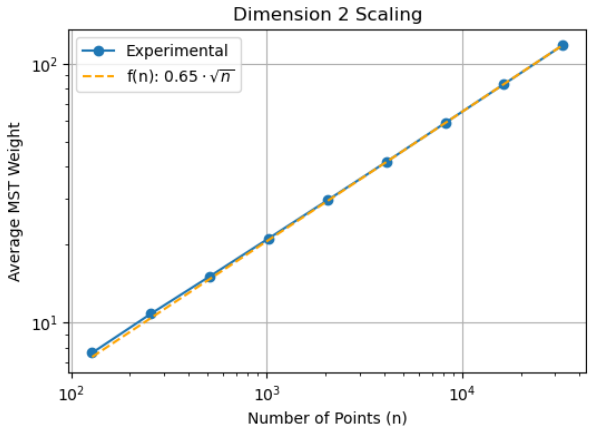
  $$(n-1) \cdot \frac{2cn}{n(n-1)+2} = \frac{2cn \cdot (n-1)}{n(n-1)+2}$$



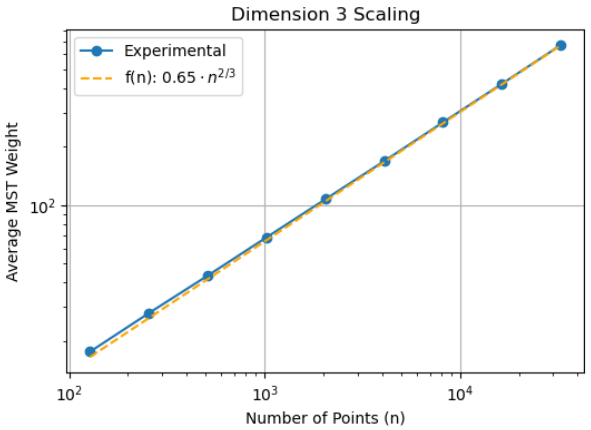
(a) Dimension 0 (Random Weights)



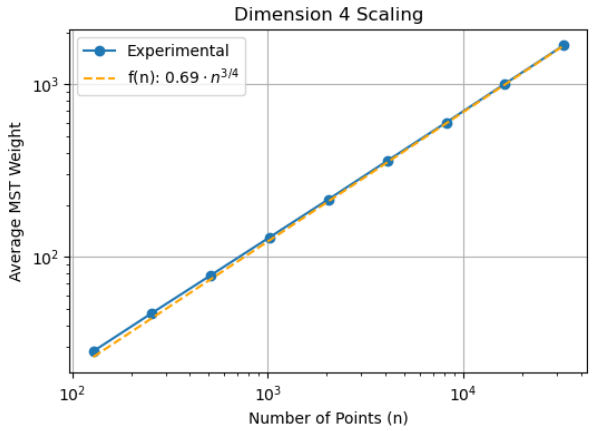
(b) Dimension 1 (Hypercube)



(c) Dimension 2 (Euclidean)



(d) Dimension 3 (Euclidean)



(e) Dimension 4 (Euclidean)

Figure 1: MST Average Weight Scaling by Dimension. The blue lines represent experimental data, and the orange dashed lines represent the theoretical growth functions $f(n)$.

We note that as n grows large, this is a fixed proportion $2c$. Using our experiments, we estimate this $2c$ to be 1.2.

- Dimension 1: $\boldsymbol{f(n) = \frac{1.18n}{\log n}}$

  Justification: Unlike the previous dimension, the hypercube, each vertex only has $\log_2 n$ neighbors. Using order statistics, this then means that the cheapest edge for each vertex is around $\frac{1}{\log n+1}$. Summing this over $n-1$ edges gives a total weight of the MST proportional to $\frac{n-1}{\log n+1} \approx c \cdot \frac{n}{\log n}$. Using our experiments, we estimate that $c = 1.18$.

- Dimension 2: $\boldsymbol{f(n) = 0.5\sqrt{n}}$

  Justification: We know that the n vertices are chosen randomly inside a unit square, which has area 1. This means on average, they each inhibit space of area $1/n$; we can picture this as a box, without loss of generality. Then, the distance between it and its nearest neighbor is simply the side length of this box, or $\sqrt{1/n}$. Therefore, the total weight of the MST is a sum of $n-1$ edges, which is approximately

$$n \cdot \sqrt{1/n} = n \cdot n^{-1/2} = n^{1/2}.$$

- Dimension 3: $\boldsymbol{f(n) = 0.65n^{2/3}}$

  Justification: We know that the n vertices are chosen randomly inside a unit cube of volume 1. Similar to dimension 2, this means each vertex inhibit a cubic space of $1/n$ on average. Then, the distance between it and its nearest neighbor is simply the side length of this cube, or $\sqrt[3]{1/n}$. Therefore, the total weight of the MST is a sum of $n-1$ edges, which is approximately

$$n \cdot \sqrt[3]{1/n} = n \cdot n^{-1/3} = n^{2/3}.$$

- Dimension 4: $\boldsymbol{f(n) = 0.69n^{3/4}}$

  Justification: We know that the n vertices are chosen randomly inside a unit hypercube of hypervolume 1. Similar to dimensions 2 and 3, this means each vertex inhibit a hypercubic space of $1/n$ on average. Then, the distance between it and its nearest neighbor is $\sqrt[4]{1/n}$. Therefore, the total weight of the MST is a sum of $n-1$ edges, which is approximately

$$n \cdot \sqrt[4]{1/n} = n \cdot n^{-1/4} = n^{3/4}.$$

2 MST Algorithm Runtime

2.1 Optimization using $k(n)$

We simply divide our $f(n)$ functions (without the experimentally determined constant) by n and, after simplifying, change the numerator from 1 to $\log n$ to ensure that we're not removing too many edges from each individual vertex. For dimension 1, that ends up being 1, so we don't end up imposing a limit.

Crucially, this will still lead to the correct MST when we run Kruskal's algorithm. This is because our algorithm sorts the edges by weight, so an edge with weight $w > k(n)$ would only be used if it were necessary to connect two components that couldn't be connected by cheaper edges. Our threshold almost guarantees that the included cheap edges already connect the entire graph,

so more expensive edges would be discarded by Kruskal's anyway. Therefore, preemptively discarding them changes nothing about the final result, but drastically improves runtime of Kruskal's algorithm.

2.2 Asymptotic Runtime

Graph Construction

For dimension 0, we visit every single pair regardless of edge weight and decide whether or not to discard it by comparing it to $k(n)$. This means that the worst-case time is equal to the expected time of $O(n^2)$. The threshold $k(n) = \frac{2\log n}{n}$ ensures that the expected number of edges is $\frac{2\log n}{n} \cdot O(n^2)$, or $M = O(n \log n)$.

For dimension 1, we visit every vertex and for each, we iterate $\log n$ neighbors. This means that the worst-case time is equal to the expected time of $O(n \log n)$. Since this graph does not discard any edges, the total number of edges is $M = O(n \log n)$.

For dimensions 2, 3, and 4, we sort the points and, for each point, iterate through its neighbors until the x-coordinate distances is greater than $k(n)$. The worst case here is $O(n^2)$, where we iterate through all of each vertex's neighbors.

Our disjoint-set union-find data structure which implements both path compression and union by rank during `unite`. This ensures that the time complexity is $O(N)$ for each find. We know that Kruskal's algorithm runs in time $O(|E| \log |V|)$

3 Discussion