CS 1240 Programming Assignment 1

Kathy Jia & Hanjing Wang

February 18, 2026

1 Collaborators and Late Days

Late days are the same for both partners.
Collaborators: none
No. of late days used on previous psets: 1
No. of late days used after including this pset: 1

2 Algorithm

2.1 Overview

We compute minimum spanning trees using Kruskal's algorithm combined with unionfind from class. For each dimension, we generate a weighted graph according to the handout and optimize to reduce the number of edges considered (from the hint). The algorithm then sorts the retained edges by weight and iteratively adds the lightest edge that connects two previously disconnected components until a spanning tree is formed.

2.2 Correctness

We will show that the algorithm outputs a correct minimum spanning tree whenever it terminates successfully.

Our algorithm is based on Kruskal's algorithm, which we know from class can correctly compute a minimum spanning tree for any connected, weighted, undirected graph by repeatedly adding the minimum-weight edge that connects two previously disconnected components. Since our implementation faithfully follows this procedure using unionfind to maintain connected components, it correctly computes an MST on the graph it is given.

Graph Generation

For each dimension, we generate a weighted graph consistent with the problem specification. In dimension 0, we simulate a complete graph with independent edge weights drawn uniformly from $[0, 1]$. In dimension 1, each vertex is connected to its $\log_2 n$ hypercube neighbors with independently sampled weights. In dimensions 2, 3, and 4, vertices are sampled uniformly at random from the unit hypercube, and edge weights correspond to Euclidean distances. In all cases, the resulting graph is undirected and weighted, satisfying the prerequisites for Kruskal's algorithm.

Optimization using $k(n)$

To improve efficiency, we discard edges whose weights exceed a dimension-dependent threshold $k(n)$ (implemented in the code as the variable `limit`). We choose $k(n)$ by taking the asymptotic form $f(n)$ of the MST weight (ignoring constant factors), dividing by n, and replacing the constant numerator with $\log n$ to ensure that we retain sufficiently many low weight edges incident to each vertex. For dimension 1, this procedure yields a constant bound, so no pruning is required in that case.

Notice that an MST cannot exist for a disconnected graph. Therefore, we will show that, for a given complete and undirected graph, the MST of the pruned graph is the same as the MST of the original graph.

- We first observe that the pruning step preserves connectivity with high probability. Since the original graph is complete, every cut contains many edges. Under random edge weights, the minimum-weight edge crossing any fixed cut is therefore very likely to have small weight. In particular, with high probability there exists at least one edge of weight at most $k(n)$ crossing each cut. As a result, removing only edges with weight greater than $k(n)$ does not disconnect the graph with high probability. Intuitively, our choice of $k(n)$ is a logarithmic factor larger than the typical edge weight used by the MST, ensuring that all edges needed for connectivity and for the minimum spanning tree are retained with high probability.

- We now show that the MST of the pruned graph is the same as the MST of the original graph for when pruning preserves connectedness. Consider running Kruskal's algorithm on both the original graph and the pruned graph. Since the pruned graph is a subgraph of the original graph, both executions encounter all edges of weight at most $k(n)$ in the same order. The only difference is that the execution on the original graph additionally considers edges of weight greater than $k(n)$ later.

  When Kruskal selects an edge e in the pruned graph, it connects two previously disconnected components and has minimum weight among all edges crossing the corresponding cut in the pruned graph. Assume for contradiction that there is an edge crossing this cut in the original graph with smaller weight. This edge would also have weight at most $k(n)$ and thus would appear in the pruned graph, contradicting the choice of e. Therefore, e is also the minimum-weight edge crossing that cut in the original graph.

  By the cut property of minimum spanning trees, such an edge is safe and must belong to some MST of the original graph. Since the pruned graph is connected, Kruskal selects exactly $n-1$ such safe edges, producing a spanning tree. This tree is therefore a minimum spanning tree of the original graph as well.

  Thus, whenever the pruning step preserves connectivity, running Kruskal's algorithm on the pruned graph produces the same MST as running it on the full graph.

Therefore, we see that pruning preserves the MST of the original complete, undirected graph.

2.3 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 graph construction takes $O(n^2)$ time. We retain an edge if its weight is at most $k(n) = \frac{2\log n}{n}$, which happens with probability $k(n)$ since weights are uniform on $[0,1]$. Therefore the expected number of retained edges is $\mathbb{E}[|E|] = k(n)\binom{n}{2} = \Theta\left(\frac{2\log n}{n} \cdot n^2\right) = \Theta(n \log n)$.

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 $\mathbb{E}[|E|] = O(n \log n)$.

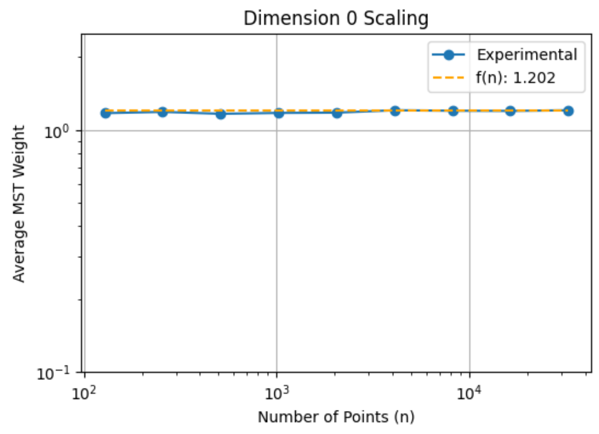
For dimensions $2, 3, 4$, we generate n points uniformly in $[0, 1]^d$ and only keep edges whose Euclidean length is at most $k(n)$ (the variable `limit`). Although the worst case is still $O(n^2)$, we can bound the expected number of edges. For two random points in $[0, 1]^d$, the probability their distance is at most $k(n)$ is $\Theta(k(n)^d)$ (proportional to the volume of a radius-$k(n)$ ball). Therefore, $\mathbb{E}[|E|] = \Theta\big(\binom{n}{2} \cdot k(n)^d\big) = \Theta(n^2 k(n)^d)$. With our choice $k(n) = \Theta\big((\log n/n)^{1/d}\big)$, we have $k(n)^d = \Theta(\log n/n)$, so $\mathbb{E}[|E|] = \Theta(n \log n)$. To construct the graph efficiently, we first create the n points, which is $O(n)$ time. We then sort points by their x-coordinate (which can be done is $O(n \log n)$ time) and only test pairs whose x-distance is at most $k(n)$. For each point, the expected number of such neighbors is $\Theta(nk(n))$, giving an expected total number of distance checks $\Theta(n^2 k(n))$. Thus, the expected edge construction time is $\Theta\left(n^2 \left(\frac{\log n}{n}\right)^{1/d}\right) = \Theta\left(n^{2-\frac{1}{d}} (\log n)^{1/d}\right)$. So, we expect graph generation in this case to take a total of $O(n) + O(n \log n) + \Theta\left(n^{2-\frac{1}{d}} (\log n)^{1/d}\right) = \Theta\left(n^{2-\frac{1}{d}} (\log n)^{1/d}\right)$ time. In the worst case, the runtime remains $O(n^2)$ when no edges are left out.

Kruskal

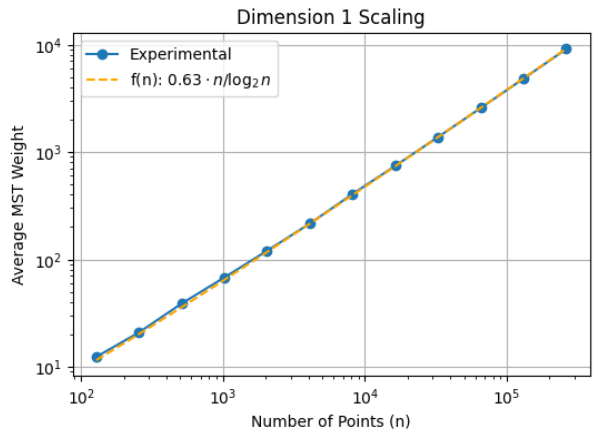
From lecture, our disjoint-set union–find structure gives an amortized time of $O(\log^* n)$ per operation. Also from lecture, we know that Kruskal's algorithm with sorting runs in $O(|E| \log |E| + |E| \log^* n) = O(|E| \log |E|)$ time, where the first term comes from sorting edges and the second from union find operations, and the sorting step dominates. Using the bounds above (typically $|E| = \Theta(n \log n)$ in expectation), the expected Kruskal time is $O((n \log n) \log(n \log n)) = O\big(n(\log n)^2\big)$ and the worse case time is $O(n^2 \log n^2)$.

3 Asymptotic Analysis of MST Size

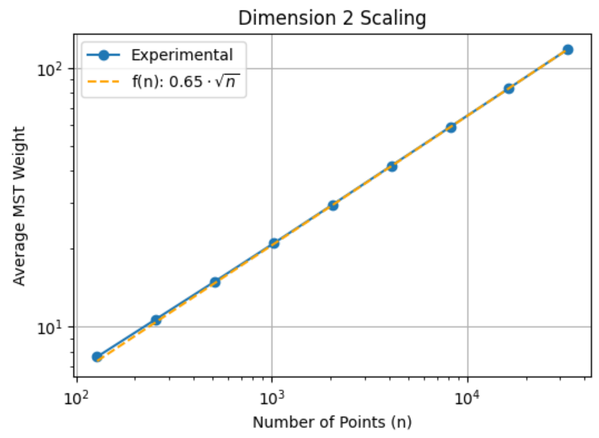
3.1 Experimental Results



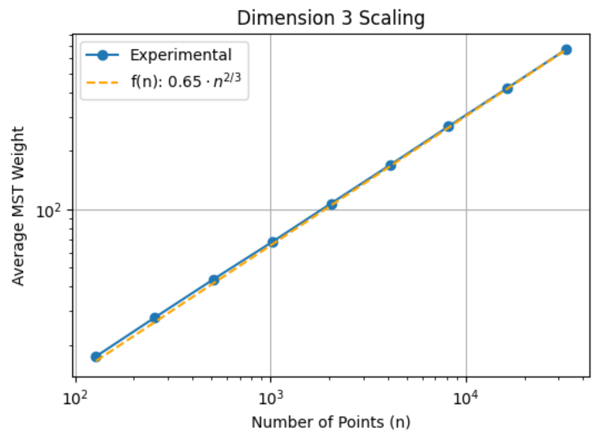
(a) Dimension 0 (Random Weights)



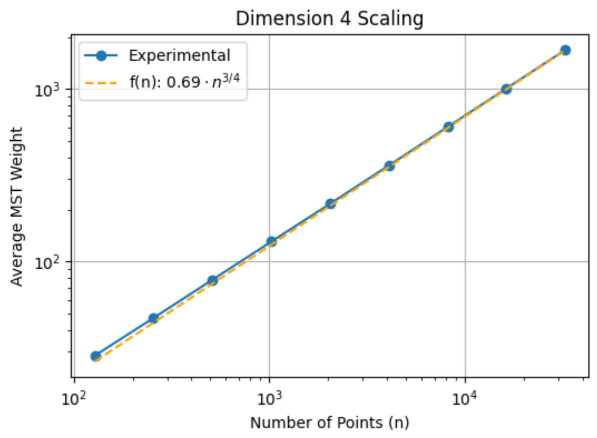
(b) Dimension 1 (Hypercube)



(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 $f(n)$.

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)

3.2 Asymptotic Forms

Across all dimensions, we observe strong empirical agreement between the experimental MST weights and the predicted asymptotic forms, including the leading constants, as shown in Figure 1. We see.

- 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 such that, with high probability, the $c \cdot n$ smallest edges can connect the graph, since edge weights are i.i.d. uniform random variables. 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
$$(n-1) \cdot \frac{2cn}{n(n-1)+2} = \frac{2cn \cdot (n-1)}{n(n-1)+2}$$
  We note that as n grows large, this is a fixed proportion $2c$. Using our experiments, we estimate this $2c$ to be 1.2 (see next subsection for details).

- Dimension 1: $\boldsymbol{f(n) = \frac{0.63n}{\log n}}$

  Justification: Unlike the previous dimension, the hypercube, each vertex only has $\log_2 n$ neighbors. Using order statistics, the expected minimum among $\log n$ independent edge weights is $\frac{1}{\log n+1}$. 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 $c = 0.63$ (see next subsection for details).

- Dimension 2: $\boldsymbol{f(n) = 0.65\sqrt{n}}$

Justification: We know that the n vertices are chosen randomly inside a unit square, which has area 1. By uniform random sampling, the expected area is $1/n$; 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

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

Using our experiments, we estimate that the constant to be 0.65 (see next subsection for details).

- 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 is expected to 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

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

  Using our experiments, we estimate that the constant to be 0.65 (see next subsection for details).

- 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 is expected to 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

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

  Using our experiments, we estimate that the constant to be 0.69 (see next subsection for details).

3.3 Determination of Leading Constants

To estimate the leading constants in each asymptotic form, we use the experimental MST weights (see table) and normalize them by the predicted growth rate. For each dimension, we divide the observed average MST weight by the theoretical scaling term (e.g. $n/\log n$, $\sqrt{n}$, $n^{2/3}$, or $n^{3/4}$). We observe that this ratio stabilizes to an approximately constant value as n increases.

We estimate the constant by averaging this ratio over the largest values of n for which experiments were feasible. The resulting constants are then used in the theoretical curves shown in Figure 1, which closely match the experimental data across all tested input sizes.

4 Discussion

Our experimental results closely match the predicted asymptotic growth rates of the minimum spanning tree weight across all dimensions considered. In particular, the observed scaling behaviors and leading constants align well with theoretical intuition based on geometric arguments and order statistics. This agreement provides empirical support for the asymptotic forms derived for each dimension.

The edge-pruning optimization using the threshold $k(n)$ significantly reduced the number of edges considered without affecting correctness in practice. While overly aggressive pruning could in principle disconnect the graph, our chosen thresholds preserved connectivity across all tested input sizes. Overall, this demonstrates that carefully chosen probabilistic optimizations can yield substantial performance improvements while maintaining correctness for large random graph instances.