Programming Assignment 1

Anusha Murali **CS124** **February 20, 2024**

1 Introduction

In this programming assignment, we determine the expected weight, $f(n)$, of the minimum spanning tree (MST) as a function of the number of vertices, n, in certain completely connected undirected graphs. Specifically, we determine the expected weight $f(n)$ of the MST for the following four types of graphs:

1. **Dimension 0:** Complete graphs on n vertices, where the weight of each edge is a real number chosen uniformly at random on $[0, 1]$.
2. **Dimension 2:** Complete graphs on n vertices, where the vertices are points chosen uniformly at random inside the unit square. (That is, the points are (x, y), with x and y each a real number chosen uniformly at random from $[0, 1]$.) The weight of an edge is just the Euclidean distance between its endpoints.
3. **Dimension 3:** Complete graphs on n vertices, where the vertices are points chosen uniformly at random inside the unit cube (3 dimensions) As with the unit square case above, the weight of an edge is just the Euclidean distance between its endpoints within the unit cube.
4. **Dimension 4:** Complete graphs on n vertices, where the vertices are points chosen uniformly at random inside the hypercube (4 dimensions). As with the unit square and unit cube cases above, the weight of an edge is just the Euclidean distance between its endpoints within the hypercube.

For each of the above four cases, we first generate a random fully connected graph of n vertices, where the edge weight is either a real number chosen uniformly at random on [0,1] (in the first case) or the Euclidean distance between two points within the geometric structure. Then we find an MST of the above graph using Prim's MST algorithm, and sum up its edge weights to find the weight of the MST. We average the results over multiple trials to find the expected weight of the MST for a given value of n. By varying the number of vertices (from $n = 2^7$ to 2^{18}), we are able to determine the asymptotic behavior of the weight of the MST. Finally, we provide a closed form expression for the expected weight of the MST, $f(n)$, for each of the four cases, based on our experimental results.

2 Algorithm

In order to find the MST of a fully connected, weighted, and undirected graph, I decided to implement the Prim's algorithm. My main reason for selecting Prim's algorithm over Kruskal's algorithm is that Prim's algorithm runs faster in dense graph than Kruskal's algorithm, and our problem involves using fully connected graph with $\binom{n}{2}$ edges. While selecting the Prim's algorithm, I made two implementation decisions: (1) Using an adjacency list to

maintain the vertices as all the vertices can be traversed in $O(E+V)$ time using breadth first search; (2) The **extractMin** operation of the Prim's algorithm, which is used to find the least-cost edge, can be implemented in $O(\log V)$ time using a Min Heap. This implementation gives an overall time complexity of $O((E+V)\log V)$ or $O(E\log V)$ (since $V = O(E)$ for a fully connected graph). In conjunction with edge pruning optimization (i.e: discarding edges with weight above a certain threshold $k(n)$), the storage requires only $O(V+E)$ using an adjacency matrix implementation.

My implementation of the Prim's algorithm is based on the pseudocode provided in the lecture notes #6, page #4. At any given point of time during the execution, we have two disjoint sets S and $V-S$. We initially add a random vertex s to S, so $S = \{s\}$. From this point on, the algorithm grows a single tree, adding an edge e to the tree, and thus a new vertex to S, until the complete MST is built. I implemented a binary Min Heap to maintain all the vertices in $V-S$, which are adjacent to the vertices in S. The priority of a vertex v, according to which the Min Heap is ordered, is the weight of its lightest edge to a vertex in S. The vertices that are removed from the Min Heap (i.e: from the set $V-S$) generate the set S for the cut property. Following are some of the important functions that implement the Min Heap framework in my C code:

1. **initMinHeap(**N**)** Creates a new Min Heap by allocating the memory for N vertices.
2. **leftChild(**i**)** Returns the left child of node at index i.
3. **rightChild(**i**)** Returns the right child of node at index i.
4. **minHeapify(**H**,** i**)** Reorganizes heap H (recursively) to preserve the Min Heap property at index i.
5. **extractMin(**H**)** Extracts the element with the smallest weight while preserving the Min Heap property.
6. **decreaseKey(**H, v, w**)** Decreases edge weight of vertex v.

The input graph, G, for the Prim's algorithm is implemented using an adjacency list. There is one adjacency list for each vertex, so there are $|V|$ number of adjacency lists in G. The adjacency lists are grown dynamically as new edges are added. Due to edge pruning optimization (suggested in the handout), not all possible $\binom{n}{2}$ edges need to be added to the graph. Following is the C structure of my graph:

```
struct Graph
{
  struct adjacencyList *adjListArray;   // Adjacency list
  int V;                                // Number of vertices in the graph
}
```

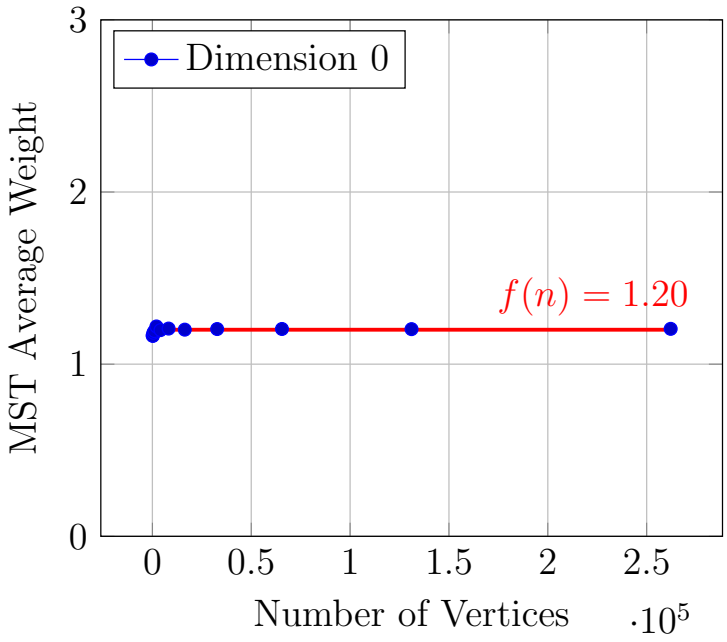
3 Experimental Results

3.1 Dimension 0, $f(n) = 1.20$

My experimental results for dimension = 0 are provided in the table below. The average weight of MST is the average of 10 trials.

No	No of Vertices	Average Weight of MST	Runtime (sec)
1	128	1.1662471294	0.000716
2	256	1.1642097235	0.003022
3	512	1.1819406748	0.009175
4	1024	1.1920676231	0.017937
5	2048	1.2189629078	0.036874
6	4096	1.1976749897	0.091573
7	8192	1.2066898346	0.318486
8	16384	1.2004313469	1.166200
9	32768	1.2040016651	4.236260
10	65536	1.2043765783	15.871895
11	131072	1.2034161091	60.747547
12	262144	1.2051198483	237.473801

The plot of the above experimental results is shown in blue below. Based on the experimental results, my guess function for dimension = 0 is $f(n) \approx 1.20$. The plot for $f(n) = 1.20$ is shown in red.

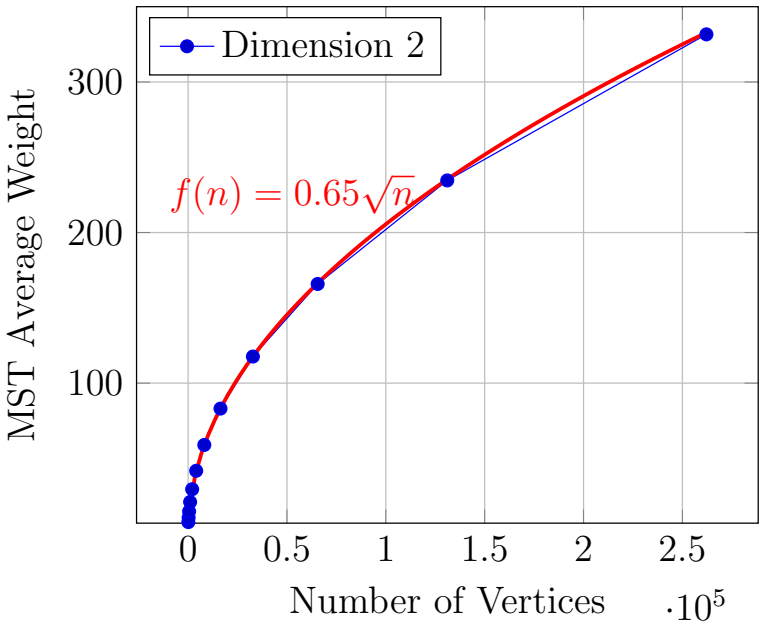


3.2 Dimension 2 $f(n) = 0.65\sqrt{n}$

My experimental results for dimension = 2 are provided in the table below. The average weight of MST is the average of 10 trials.

No	No of Vertices	Average Weight of MST	Runtime (sec)
1	128	7.7088584900	0.000334
2	256	10.6747636795	0.002773
3	512	14.7976322174	0.006889
4	1024	21.0650691986	0.040100
5	2048	29.5337257385	0.274430
6	4096	41.7864608765	0.628942
7	8192	58.9421501160	0.864193
8	16384	83.0817718506	1.253019
9	32768	117.6479721069	2.721428
10	65536	165.8600006104	9.027367
11	131072	234.6125488281	34.319263
12	262144	331.6229553223	133.729492

The plot of the above experimental results is shown in blue below. Based on the experimental results, my guess function for dimension = 2 is $f(n) = 0.65\sqrt{n}$. The plot for $f(n) = 0.65\sqrt{n}$ is shown in red.

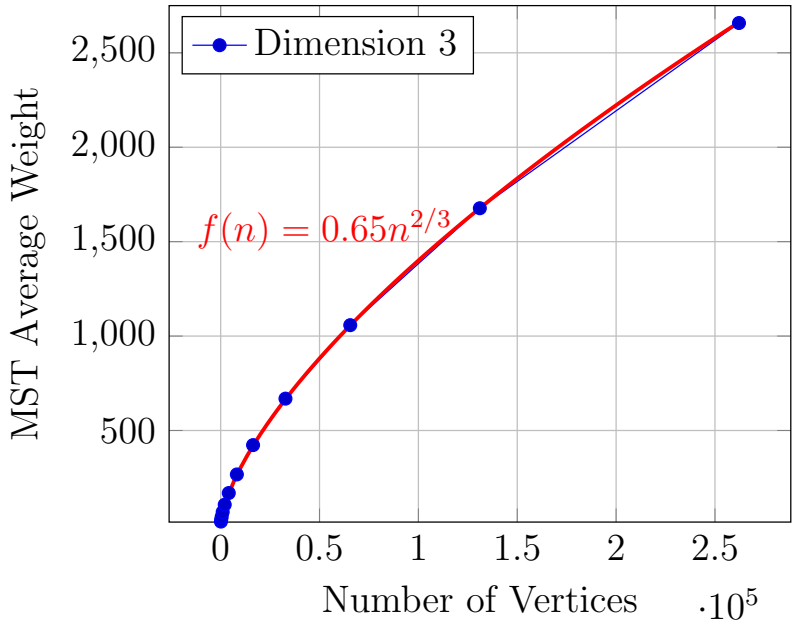


3.3 Dimension 3 $f(n) = 0.65n^{2/3}$

My experimental results for dimension = 3 are provided in the table below. The average weight of MST is the average of 10 trials.

No	No of Vertices	Average Weight of MST	Runtime (sec)
1	128	17.6050910950	0.002250
2	256	27.5121307373	0.001193
3	512	43.1290817261	0.007557
4	1024	68.3373947144	0.034372
5	2048	107.3108291626	0.257261
6	4096	168.8387756348	1.297971
7	8192	267.2348022461	4.526847
8	16384	422.6395874023	5.641924
9	32768	668.6824951172	6.136513
10	65536	1057.7683105469	12.909258
11	131072	1676.9371337891	46.883057
12	262144	2657.8237304688	183.570175

The plot of the above experimental results is shown in blue below. Based on the experimental results, my guess function for dimension = 3 is $f(n) = 0.65n^{2/3}$. The actual plot for $f(n) = 0.65n^{2/3}$ is shown in red.

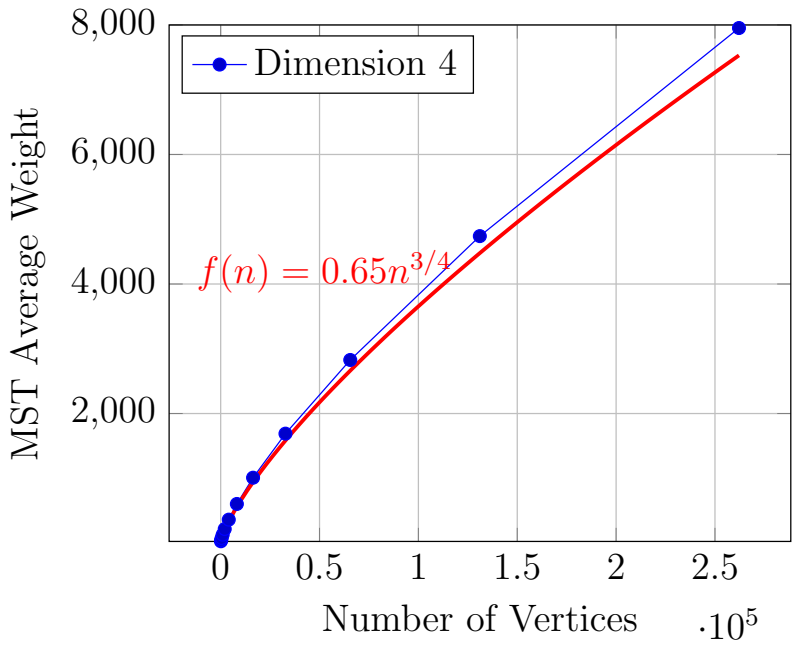


3.4 Dimension 4 $f(n) = 0.65n^{3/4}$

My experimental results for dimension = 4 are provided in the table below. The average weight of MST is the average of 10 trials.

No	No of Vertices	Average Weight of MST	Runtime (sec)
1	128	27.7515373230	0.000374
2	256	46.1811752319	0.001176
3	512	77.7852325439	0.011145
4	1024	129.5397338867	0.049760
5	2048	216.9770050049	0.333785
6	4096	361.4254760742	1.494581
7	8192	603.6107177734	5.445861
8	16384	1009.1958007812	4.902279
9	32768	1689.3171386719	5.056851
10	65536	2827.0375976562	14.962617
11	131072	4739.3750000000	87.358170
12	262144	7952.5415039062	243.105301

The plot of the above experimental results is shown in blue below. Based on the results and the plot, my guess function for dimension = 4 is $f(n) = 0.65n^{3/4}$, which is shown in red.



4 Discussion

My implementation of the Prim's algorithm is based on the pseudocode provided in the lecture notes #6, page #4. Therefore, the correctness follows from the proof of the correctness for the original Prim's algorithm. I used two main strategies to optimize my implementation: (1) Using an adjacency list to maintain the vertices as all the vertices can be traversed in $O(m+n)$ time; (2) Using a Min Heap to implement the priority queue required to maintain the elements in the set $V-S$, where V is the set of initial vertices in G, and S is the growing MST at any point in time of the algorithm. The **extractMin** operation of the Prim's algorithm, which is used to extract the least-cost edge, can be implemented in $O(\log n)$ time using a Min Heap. My implementation of a priority queue as a Min Heap allows both the extraction of the node with the minimum edge weight (including the reorganization of the Min Heap) and the decrease key operation $O(\log n)$ time. Since there will be $O(n)$ **extractMin** operations and $O(m)$ **decreaseKey** operations, this gives us an overall runtime of $O((m+n)\log n)$.

Since the number of edges, m, in a fully connected graph is equal to $\binom{n}{2} = \frac{1}{2}n(n-1)$, we notice that the number of edges grows by $O(n^2)$. Therefore, each time when we increase the number of vertices by a factor of 2, the number of edges will increase approximately by a factor of 4. For example, in the 3 dimension case, the average time taken to find the MST (ignoring the bookkeeping costs such as finding the average weight) for $n = 131072$ is 46.88 seconds, while the average time taken to find the MST for $n = 262144$ is 183.57 seconds, which is nearly 3.92 times larger. Given that the actual ratio of $(m+n)\log n$ for $n = 262144$ and $n = 131072$ is approximately 4.23, and the fact that I pruned edges with large weights to optimize my code, the observed runtime increase as n was being increased is consistent with the theoretical runtime of my implementation, which is $O((m+n)\log n)$. This behavior is more apparent for larger values of n, proving that the implementation indeed has an asymptotic runtime of $O((m+n)\log n)$.

The plot of the experimental results for dimension = 0 shows that as n becomes large, $f(n)$ is converging approximately to $f(n) = 1.205$. A constant $f(n)$ makes sense in this case as we are not placing any constraints on the vertices locations in this case, except that the edge weights should be in [0, 1]. As n increases, since the edge weights are uniformly distributed in [0,1], the edge choices for the Prim's algorithm become limited as the number of equally (closely) weighted edges increases, and therefore the expected weight of the MST will converge to a constant.

The shape of the plot in a higher dimension is explained by the fact that initially for small n, the Euclidean distances between vertices tend to be larger. For example, for dimensions 2, 3, and 4, the maximum possible distances between two vertices are $\sqrt{2}$, $\sqrt{3}$, and 2 respectively. The plots of the experimental results for dimension = 2, 3 and 4 show that as n becomes large, $f(n)$ is increasing at a monotonically decreasing rate (i.e: $\frac{\mathrm{d}^2y}{\mathrm{d}x^2} < 0$). This is because, as n becomes large, there will be a lot more shorter edges between vertices (as we have to compact more points within the same volume or hyperspace). So, Prim's algorithm naturally ends up choosing the smaller weighted edges to grow the MST, which explains why the expected

weight is only slowly growing.

5 Asymptotic Analysis of MST Size

Consider n random points in a unit square. Using the pigeonhole principle, by dividing the unit square into small enough squares so that we can see that there are at least two points that are at a distance of at most $\sqrt{n}$. Let us construct a spanning tree with these n points as vertices, where we add an edge between two such closest points at distance at most $\sqrt{n}$. Excluding the starting point, now we have $n-1$ points left. Applying the pigeonhole principle again, we see that there are at least two points that at a distance of at most $\sqrt{n-1}$. We can grow our spanning tree in this manner by adding each edge until we have only 2 points left. Therefore, the total weight of the spanning tree is at most,

$$w = n^{1/2} + (n-1)^{1/2} + (n-2)^{1/2} + \cdots + 2^{1/2}$$

We can see that the above summation is bounded by $w \leq \int_2^{n+1} x^{1/2}\,\mathrm{d}x = n^{1/2}$. Therefore, a spanning tree connecting n uniformly distributed random points in a unit square has a size at most $O(n^{1/2})$, which is consistent with our experimental results for dimension = 2, where we found the expected weight of the MST to be $f(n) = 0.65\sqrt{n}$.

In a similar manner, we can show that the expected weights of the spanning tree in a 3-dimensional unit cube and a 4-dimensional unit hypercubes are as follows:

$$w_{\text{unit cube}} = n^{1/3} + (n-1)^{1/3} + (n-2)^{1/3} + \cdots + 2^{1/3} \leq n^{2/3}$$
$$w_{\text{hypercube}} = n^{1/4} + (n-1)^{1/4} + (n-2)^{1/4} + \cdots + 2^{1/4} \leq n^{3/4}.$$

The above upper bounds for the weights are consistent with our experimental findings of the asymptotic functions $f(n) = 0.65n^{2/3}$ for the unit cube (dimension = 3) and $f(n) = 0.65n^{3/4}$ for the hypercube (dimension = 4).

For any arbitrary higher dimension d, where each dimension has unit length, the above results becomes $w_d \leq n^{(d-1)/d}$, so the weight of the spanning tree is bounded by $O\left(n^{(d-1)/d}\right)$.

Using a similar argument, when we divide a d-dimensional hypercube into smaller boxes, for any $\epsilon > 0$, there could be only ϵn edges whose weight is $O(n^{1/d})$ with a probability of $1 - \epsilon$. This leads us to a lower bound of $\Omega(n^{(d-1)/d})$ for the weight of the spanning tree.

The idea for the above asymptotic bounds was obtained from the paper, "Growth rates of euclidean minimal spanning trees with power weighted edges" by Michael Steele [1]. My experimental results for up to $n = 2^{18} = 262144$ helped confirm the above theoretical asymptotic bounds as well as to obtain the constants (≈ 0.65 for dimensions = 2, 3, and 4) associated with them.

6 Conclusion

In this programming assignment, we determined the expected weight, $f(n)$, of the minimum spanning tree (MST) as a function of the number of vertices, n, in certain completely connected undirected graphs. Specifically, we determined the expected weight of the MST for dimensions = 0, 2, 3 and 4 by building MST for increasingly large values of n and studying their asymptotic behavior.

I decided to choose Prim's algorithm for generating MSTs, because using a binary Min Heap and an adjacency list to maintain the vertex/edge information, one can achieve an $O((m + n) \log n)$ runtime. My implementation followed the standard Prim's algorithm presented as a pseudocode in the lecture notes and in the textbooks. My only optimizations to the naive algorithm are (1) using a Min Heap to speed up the **extractMin** and **decreaseKey** operations and (2) using an adjacency list to improve the storage complexity. I further optimized my code by discarding the edges that are not likely to be part of the eventual MST. I found that edge weights above a certain threshold, $k(n)$, were not being used in the MST and therefore can be discarded upfront. This allowed me to reduce the memory required to store my data structures, and to build my graph faster, although its contribution to the speedup of the program was not substantial.

I found that calling **srand()** using current time as a seed can result in the same random numbers for a few trials at lower values of n. I suspect this was the culprit which caused a few tests to fail on the Autograder in my initial uploads. I resolved this issue by seeding **srand()** at the granularity of a microsecond.

In conclusion, my experiments showed that expected size $f(n)$ of the MST depends on the dimension of the problem. For dimension 0, as n asymptotically becomes large, $f(n)$ approximately converges to 1.20. For higher dimensions, $d = 2, 3$, and 4, my experiments found that as n asymptotically becomes large, $f(n) \approx 0.65n^{(d-1)/d}$. I believe a more accurate value for the constant can be found by conducting experiments on even larger values of n (i.e: $n > 2^{18}$). I also further learned from this exercise that both the choice of the algorithm as well as the choice of the appropriate data structures (such as Min Heap and adjacency list) for the chosen algorithm are extremely important when implementing the solution in order to obtain an optimal asymptotic runtime on large inputs.

References

[1] J. Michael Steele, "Growth rates of Euclidean minimal spanning trees with power weighted edges", The Annals of Probability, 1988, Vol. 16, No. 4, 1767-1787.