Programming Assignment 1

Bin Yan

1 Experiment results

I have simulated the average size of minimum spanning tree of four types of complete undirected graphs for number of vertices $n = 16, 32, 64, \dots, 32768, 65536, 131072, 262144$. I have run 40 trials for each graph when $n \leq 1024$. The results for dimension d=1, 2, 3 and 4 are listed in Table 1. The results are also plotted in Figure 1, Figure 2(a), Figure 3(a) and Figure 4(a).

Table 1 Average size of minimum spanning tree for four types of graphs

numpoints	d = 1	d=2	d=3	d=4	numtrials
16	1.193	2.64	4.38	6.15	40
32	1.156	3.83	7.09	10.22	40
64	1.192	5.40	11.21	17.06	40
128	1.169	7.67	17.57	28.57	40
256	1.191	10.69	27.57	47.12	40
512	1.195	15.05	43.36	78.19	40
1024	1.202	21.12	68.03	129.96	40
2048	1.196	29.68	107.16	216.42	20
4096	1.202	41.78	169.32	361.10	20
8192	1.204	59.01	267.31	602.56	20
16384	1.202	83.20	422.26	1008.71	20
32768	1.206	117.39	669.25	1688.58	5
65536	1.199	165.96	1058.60	2827.48	5
131072	1.202	234.62	1677.85	4739.94	5
262144	1.203	331.60	2658.32	7952.42	5

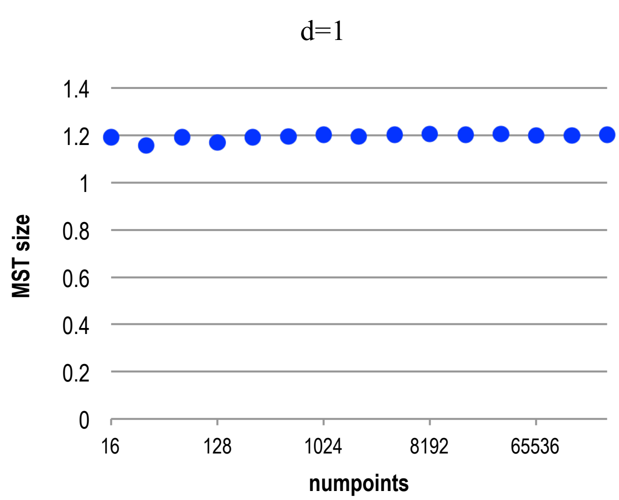

Figure 1 MST size vs. number of vertices, dimension = 1

d=2

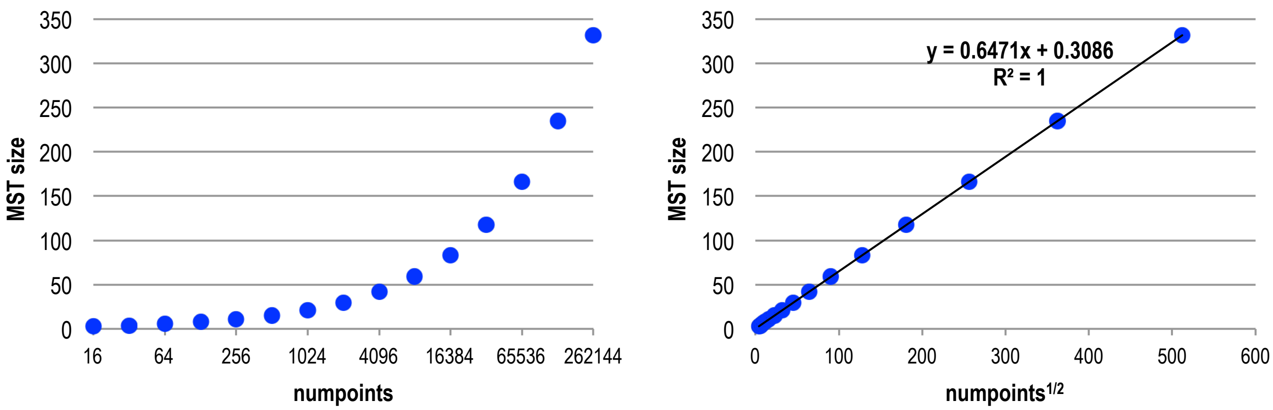

Figure 2 (a) MST size vs. number of vertices n (b) MST size vs. number of vertices $n^{\frac{1}{2}}$, dimension = 2

d=3

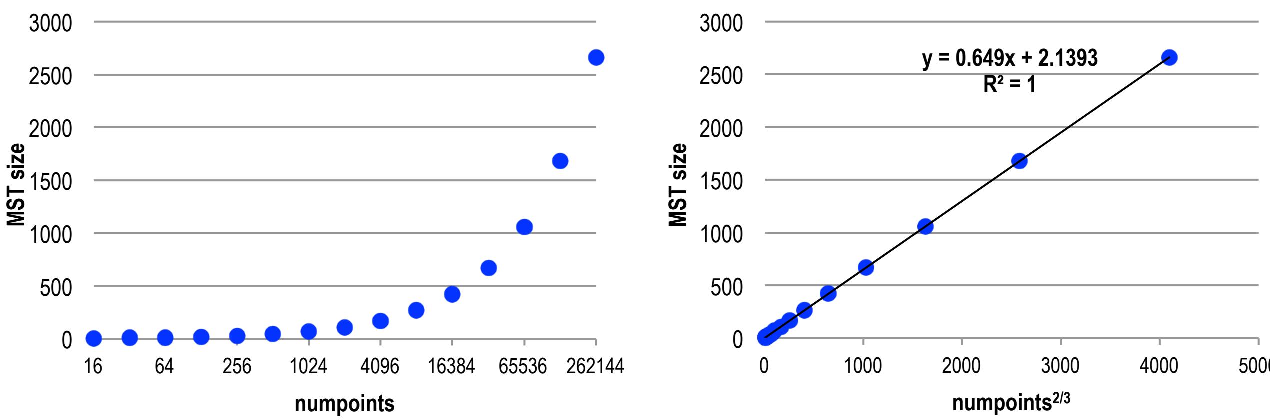

Figure 3 (a) MST size vs. number of vertices n (b) MST size vs. number of vertices $n^{\frac{2}{3}}$, dimension = 3

d=4

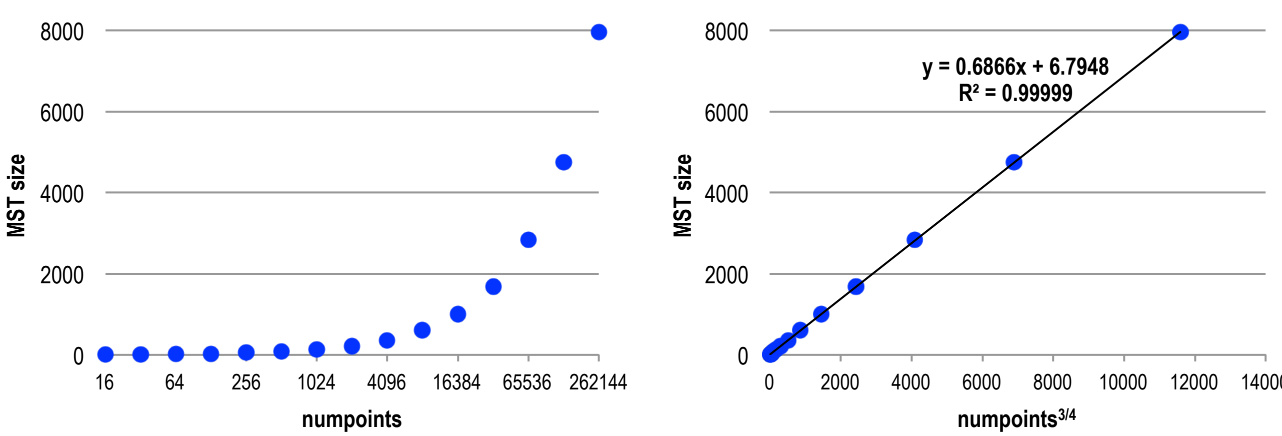

Figure 4 (a) MST size vs. number of vertices n (b) MST size vs. number of vertices $n^{\frac{3}{4}}$, dimension = 4

2 $f(n)$

As shown in in Figure 1, Figure 2(b), Figure 3(b) and Figure 4(b),

$$f(n) = \begin{cases} 1.20, & d = 1 \\ 0.6471n^{1/2} + 0.3086, & d = 2 \\ 0.6490n^{2/3} + 2.1393, & d = 3 \\ 0.6866n^{3/4} + 6.7948, & d = 4 \end{cases}$$

where n is the number of vertices in the graph, and the d is the dimension of the graph type.

Therefore, my guess of $f(n)$ is,

$$f(n) = c_1 + c_2 n^{\frac{d-1}{d}}$$

where c_1 and c_2 are constants.

3 Discussion

Prim vs. Kruskal

I use Prim's algorithm. In particular, I use adjacency matrix to represent a graph in Prim's algorithm.

For Prim's algorithm, if using adjacency matrix searching, the running time is $O(|V|^2)$. If using binary heap and adjacency list, the running time is $O(|E| \log|V|)$, where $|V|$ is the number of the vertices and $|E|$ is the number of edges. For Kruskal's algorithm, the running time is $O(|E| \log|E|)$. Since we are dealing with complete graph, $|E| = |V|^2$. Therefore, I choose Prim's algorithm and adjacency matrix to represent the graph. In this particular case, heap will not help. I simply use two nested for loops which takes $O(|V|^2)$.

Memory problem when n is large

When n (n corresponds to $|V|$) grows larger than 30,000, memory issue becomes a challenge. If we store all the edges in the memory, we need sizeof(float) $\times$ $30{,}000^2 = 3.6$ GB. My laptop memory is 4 GB. It still can handle $n = 30{,}000$ as I tested, but already takes very long. To solve this problem, I define three types of graphs in C++, AdjacencyMatrixGraph, HashGraph and EuclideanGraph.

AdjacencyMatrixGraph is used for one dimensional random graph when $n \leq 33{,}000$. how many elements in total, edge (i, i) not exists, (i, j) and (j, i) in …