# Expected weight of the minimum spanning trees of Euclidean and random complete graphs

**Swati Goel**[1] **and Lev Kruglyak**[2]

sgoel@college.harvard.edu[1], levkruglyak@college.harvard.edu[2]

## 1 Introduction

In this report, we would like to experimentally determine the asymptotic behavior of the average total weight of a minimum spanning tree of some random graphs as the number of vertices becomes very large. Specifically, we consider the following two types of graphs:

- **Random Complete Graphs:** A complete undirected graph on $n$ vertices where the weight of each edge $(u, v)$ is chosen randomly and uniformly from the real interval $[0, 1]$.
- **Random Euclidean Graphs:** (EG) A complete undirected graph on $n$ vertices in the unit hypercube $[0, 1]^d$, where the weight of an edge $(u, v)$ is the Euclidean distance $\|u - v\|$.

For the sake of generality, we'll refer to a random complete graph as a 0-dimensional EG. Since the 1-dimensional case isn't very interesting, we are thus interested in the 0-dimensional case (i.e. just a random complete graph), and the $2, 3, 4$-dimensional cases. Let $f_d(n)$ be the expected total weight of a random $d$-dimensional Euclidean graph. (Including the 0-dimensional case). Our goal is then to find an asymptotic expansion of $f_d(n)$.

## 2 Random complete graphs

The naive approach to computing the MST of a random complete graph would be to simply build up the graph in its entirety and then run an MST algorithm on the generated graph, which may be optimized for very dense graphs. Using something like Prim's algorithm with a Fibonacci heap, we could get a runtime of $O(V^2 + V \log V)$ where $V$ is the number of vertices in the graph. This turns out to be highly unfeasible since it requires storing $V^2$ edges, which can easily surpass the maximum memory of any reasonable computer. There are a few ways to get around this limitation.

### 2.1 Procedural random graphs

Using procedural random functions we can actually dispense with storing the graph at all, reducing the space complexity to constant. As opposed to an iterative random function which outputs a sequence of random numbers based on the order in which it was called, a procedural random function takes in some input, and consistently returns the same random number every time it is called.

So a procedural random graph need only contain the number of verticies in the graph and a seed value. The graph comes with some procedural random function which takes in a seed and two edge indices. (We can pick the random function so that it is symmetric in the edge indicies, so that the graph is property undirected) Then every time we want to fetch an edge, we simply call this procedural random function, and it will return the same random edge weight. Such procedural random functions can be easily created by composing several cryptographic integer functions and then using some bit manipulations to convert this to a float. We do some fun stuff with shadertoy to visualize the efficacy of various cryptographic functions.

To find the MST of such a procedural graph, we apply Prim's algorithm to this graph interface, since Prim's algorithm does not require any storing of edges. In practice, we found that this approach was too slow, so we switched to a different approach.

### 2.2 Sparsifying a random complete graph

A secondary approach to this problem is to build a sparse graph on $n$ vertices, by only adding edges to the graph when the weight of the edge is less than some threshold $\kappa(n)$. Ideally, this bound would be chosen not to change the minimum spanning tree. For our implementation, we experimentally determined that an upper bound of the form

$$\kappa(n) = c\frac{\log_2 n}{n}$$

works, where $c$ is some fixed constant. We deduced the value of $c$ by starting it at some very high number so the generated graph would be complete, any slowly lowering it until the weight of the minimum spanning tree changes significantly. We found that a $c$-value of approximately $c \approx 4.0$ seems to work.

We then run Kruskal's algorithm on this reduced graph, giving us a minimum spanning tree. Assuming a uniformly distributed random function, the number of edges

is $O(\kappa(n) \cdot n^2) = O(n \log n)$. Thus the time complexity of Kruskal's algorithm applied to this sparse graph is $O(n \log^2 n)$. This is a more reasonable time frame.

### 2.3 Numerical results

For each $n$, we ran 8 trials, each in a separate thread for maximum performance, and then the results were averaged. Computing all these values took about 30 minutes on a 2.3 GHz 8-Core Intel Core i9 Macbook Pro with 16 GB 2667 MHz DDR4 memory, which was fairly expected based on the algorithm's performance for smaller values and the asymptotic time complexity.

Table 1. Random complete graph MST weights

| $n$ | $f_0(n)$ | $n$ | $f_0(n)$ |
|---|---|---|---|
| $2^7$ | 1.10562 | $2^{13}$ | 1.21397 |
| $2^8$ | 1.09412 | $2^{14}$ | 1.2037 |
| $2^9$ | 1.0523 | $2^{15}$ | 1.21013 |
| $2^{10}$ | 1.15964 | $2^{16}$ | 1.2026 |
| $2^{11}$ | 1.16696 | $2^{17}$ | 1.20162 |
| $2^{12}$ | 1.17769 | $2^{18}$ | 1.20655 |

The asymptotic expansion of $f_0(n)$ appears to be constant, converging to $\lim_{n\to\infty} f_0(n) \approx 1.206$. This is somewhat intuitively understandable, since given any node $u$, we can on average find an edge to some node $v$ with weight approximately equal to $1/n$. Then the MST should have about $n$ nodes and so the total weight is some constant factor.

### 2.4 Theoretical asymptotic expansion for $f_0(n)$

There already exists a theoretical limit for the expected weight of a random complete graph, somewhat surprisingly it is related to the Riemann zeta function from analytic number theory. See [1] for a proof.

**Theorem 1** *Given a complete graph on n verticies, where the edge weights are uniformly distributed in the* $[0, 1]$ *interval, the expected weight of an MST of this graph approaches the constant* $\zeta(3)$ *as* $n \to \infty$ *where* $\zeta(s)$ *is the Riemann zeta function:*

$$\zeta(s) = \sum_{k=0}^{\infty} \frac{1}{k^s}$$

*The constant* $\zeta(3)$ *is known as Apéry's constant.*

This theoretical result is inline with our experimental result, since

$$\zeta(3) = 1.202\ 056\ 903\ldots$$

In dimensions $d \geq 2$, the asymptotics of $f_d(n)$ look very different, however there is much more optimization to be had in these Euclidean cases.

## 3 Random Euclidean Graphs

As opposed to a totally random complete graph, the edge weights on a Euclidean graphs have significantly more structure, which can lend itself to some incredible optimizations. For our implementation, we used the algorithm presented in [2].

This algorithm doesn't give an exact MST but an approximate one. More formally, given some point set $P \subset \mathbb{R}^d$ and error parameter $\varepsilon > 0$, the algorithm produces a Euclidean spanning tree for $P$ whose weight is at most $(1 + \varepsilon)$ times the weight of the true minimum spanning tree. The algorithm does this with time complexity $O\left(n \log n + \left(\varepsilon^{-2} \log^2 \frac{1}{\varepsilon}\right) n\right)$. This algorithm is incredibly efficient and also quite simple.

### 3.1 Overview of the algorithm

The general idea behind the algorithm is to first build a quadtree (resp. higher order trees in $d > 2$) containing all of the points in the graph. Building such a quadtree takes only $O(n \log n)$ time complexity, and it is actually the fastest part of the algorithm. Then we find an $s$-well separated pair decomposition ($s$-WSPD) of this quadtree into a set of pairs of well separated nodes. (Choosing this $s$ is described later.)

If we choose $s$ carefully, we can construct a 2-WSPD of size $O(n)$ in $O(n \log n)$ time. Finally, we select a minimum edge connecting each pair of well separated nodes using some depth criteria to ensure we don't iterate throuh too many pairs of edges. We add these minimum edges to a graph, and so we now have a graph with $O(n)$ edges, and it can be proven that it must contain a spanning tree whose weight is at most $(1 + \epsilon)$ times the actual minimum spanning tree. Running any MST algorithm on this reduced graph gives us our desired result, again in $O(n \log n)$ time.

**Algorithm 1:** Approx-EMST($P, \varepsilon$)

```
Q(P) ← a quadtree for P
Ψ(P) a s-WSPD for P
G ← (P, ∅)
for (u, v) ∈ Ψ(P) do
    U ← {u}; V ← {v}
    k ← 1
    while k ≤ λ(ε) and max(|U|, |V|) ≤ γ/ε do
        U ← expand(U); V ← expand(V)
        k ← k + 1
    end
    C ← {(rep(u'), rep(v')) : (u', v') ∈ U × V}
    (p, q) ← argmin_{(p,q) ∈ C} ||pq||
    add edge (p, q) to G
end
return kruskals_mst (G);
```

Here, $\lambda(\varepsilon) = 2 - \log_2 \varepsilon$ and $\gamma > 0$ is some constant involved in the accuracy of the graph. Choosing

it must be done experimentally, we found that $\gamma \approx 1$ works well. Lastly, $s$ is a constant involved in determining how large/accurate the $s$-WSPD construction is, we found that $s < 1$ works well. Increasing it significantly past this to say, $s = 2$ as the official algorithm suggests leads to diminishing returns in the accuracy of the algorithm, yet increases runtime significantly.

### 3.2 Numerical results

As before, for each $n$ we ran 8 trials, multithreaded for maximum performance and averaged the results. The algorithm ran very quickly; for a benchmark, it was able to calculate $f_4(2^{18})$ in under 20 seconds. Letting the algorithm run for only a few minutes, we were able to obtain the following values:

Table 2. Random Euclidean graph MST weights

| $n$ | $f_2(n)$ | $f_3(n)$ | $f_4(n)$ |
|---|---|---|---|
| $2^7$ | 8.0888 | 18.4344 | 29.008 |
| $2^8$ | 11.607 | 29.234 | 49.9654 |
| $2^9$ | 16.5037 | 46.3574 | 83.9279 |
| $2^{10}$ | 23.4344 | 73.5039 | 141.87 |
| $2^{11}$ | 33.169 | 116.624 | 236.841 |
| $2^{12}$ | 46.9872 | 186.252 | 401.888 |
| $2^{13}$ | 66.5872 | 293.833 | 676.886 |
| $2^{14}$ | 94.2845 | 467.511 | 1132.79 |
| $2^{15}$ | 133.569 | 742.982 | 1898.65 |
| $2^{16}$ | 188.905 | 1177.78 | 3223.1 |
| $2^{17}$ | 267.479 | 1872.09 | 5415.15 |
| $2^{18}$ | 378.2 | 2969.83 | 9060.04 |
| $2^{19}$ | 534.991 | 4713.59 | 15187.7 |
| $2^{20}$ | 756.245 | 7484.55 | 25787 |

If we plot these data points on a log-log graph (see Figure 1), we see that the data becomes almost perfectly linear, so we can run a linear regression to get an asymptotic approximation for $f_d(n)$. We then have the following asymptotic expessions for $f_d(n)$: $f_2(n) \in \Theta(n^{0.503})$, $f_3(n) \in \Theta(n^{0.667})$ and $f_4(n) \in \Theta(n^{0.752})$. This leads to the conjecture that for $d \geq 2$, $f_d(n) \in \Theta(n^{(d-1)/d})$.

### 3.3 Theoretical asymptotic expansion for $f_d(n)$

In [3], an asymptotic expression for $f_d(n)$ is given in terms of the density of the probability distribution for picking points in the unit hypercube.

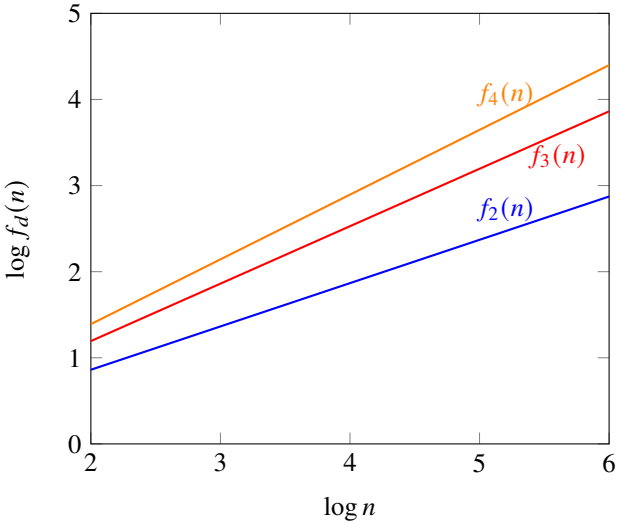
**Theorem 2** *Let $f$ be the density of the probability function for picking points in the hypercube. Then for any $d > 1$, the expected weight of the Euclidean minimum spanning tree approaches*

$$c(d)n^{\frac{d-1}{d}} \int_{\mathbb{R}^d} f(x)^{\frac{d-1}{d}} dx$$

*as $n \to \infty$.*

In particular, for a uniform probability distribution we have $f_d(n) \in \Theta(n^{\frac{d-1}{d}})$, confirming our conjecture.

Figure 1. log-log graph of $f_d(n)$ for $d = 2, 3, 4$




## 4 Conclusion

As mentioned, our algorithm is quite efficient, calculating the MST for $f_4(2^{18})$ in under 20 seconds. Storage was not an issue. A notable issue we had at first when implementing our own procedural random functions was that $f_0(n)$ seemingly converged to 0.6, but this was because our random function wasn't uniformly distributed. Once we switched to using the mt19937 uniform distribution library, these issues disappeared.

## References


[1] A.M. Frieze. On the value of a random minimum spanning tree problem. *Discrete Applied Mathematics*, 10(1):47–56, 1985. ISSN 0166-218X. doi:https://doi.org/10.1016/0166-218X(85)90058-7. URL `https://www.sciencedirect.com/science/article/pii/0166218X85900587`.

[2] Sunil Arya and David Mount. A fast and simple algorithm for computing approximate euclidean minimum spanning trees. pages 1220–1233, 01 2016. doi:10.1137/1.9781611974331.ch85.

[3] J. Michael Steele. Growth rates of euclidean minimal spanning trees with power weighted edges. *The Annals of Probability*, 16(4):1767–1787, 1988. ISSN 00911798. URL `http://www.jstor.org/stable/2243991`.