PA 1 | Vladimir Petrov

Part 1: Experiment

I got following tables:

dim = 0 (random weights):

$n \to$	128	256	512	1024	2048	4096	8192	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
average MST	1.19	1.22	1.17	1.19	1.21	1.21	1.20	1.20	1.20	1.20	1.20	1.20

So, my guess here $f_0(n) \approx 1.20$

dim=2 (unit square):

$n \to$	128	256	512	1024	2048	4096	8192	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
average MST	7.77	10.68	14.85	21.07	29.58	41.75	58.73	83.22	117.53	166.0	234.67	331.64

Noticing that each second time (i.e. from 2^k to 2^{k+2}) the results grows approx. twice, I assume $\sqrt{n}$ dependency

So, my guess here $f_2(n) \approx c_2(n) \cdot n^{\frac{1}{2}} = 0{,}65 \cdot n^{\frac{1}{2}}$ $\left(0{,}65 \approx \frac{331.64}{512}\right)$

dim=3:

$n \to$	128	256	512	1024	2048	4096	8192	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
average MST	17.68	27.53	43.53	67.78	107.29	169.2	267.32	423.11	668.6	1058.6	1675.5	2658.0

Scanned with CamScanner

Assuming some polynomial dependency n^α, I find $\alpha \approx 0,66$

so, my gues here $\alpha = \frac{2}{3}$ and thus

$$f_3(n) \approx c_3(n) \cdot n^{\frac{2}{3}} = 0,65 \cdot n^{\frac{2}{3}}$$ (again, $0,65 \approx \frac{2658,0}{(2^{18})^{2/3}} = \frac{2658}{4096}$)

dim = 4:

$n \rightarrow$	128	256	512	1024	2048	4096	8192	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}
avrg MST	28.26	47.06	78.43	130.42	216.75	361.68	602.6	1008.0	1687.3	2829,6	4742.8	7950.7

Assuming polynomial dependency as well, I find n^α with $\alpha \approx 0,75$

so, my guess here $f_4(n) \approx c_4(n) \cdot n^{\frac{3}{4}} = 0,69 \cdot n^{3/4}$ $\left(0,69 \approx \frac{2829.6}{(2^{16})^{3/4}}\right)$

Part 2. Analysis

comparing dim = 2,3,4 my general guess $f_d(n) \approx C(d) \cdot n^{\frac{d-1}{d}}$

This makes sense intuitively:

suppose we are in dimension d and $n \approx m^d$ (with $m \in \mathbb{N}$)

Let's split unit cube into m^d "cells" of smaller sub-cubes with size = $(1/m)$.

Then, there is exactly one point in each cell on average

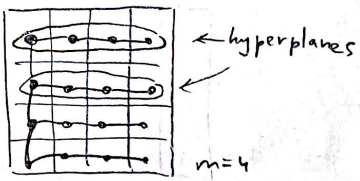
(since # points = $N = m^d$ = # cell and they're all isometric)

Now, our MST in this case looks like connecting points in adjacent cells (not all pairs, of course) since the further the cells the greater the distance btw the points inside

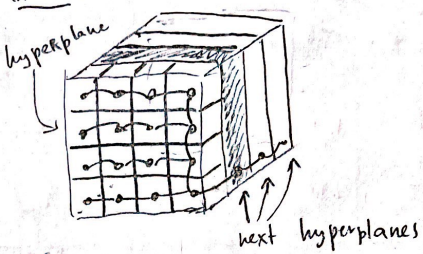
Scanned with CamScanner

One implementation of MST is just connecting points in "the same hyperplane" and then connecting hyperplanes as shown in cases dim = 2 & 3:

dim = 2:



dim = 3:



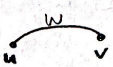
There will be $|E| = N-1 = m^d - 1$ edges & each of them $\approx$

$\approx$ (const) · size(cell) = (const) · $\frac{1}{m}$

$\Rightarrow$ weight(MST) $\approx$ (const) · $\frac{1}{m}(m^d - 1) \approx c_d \cdot m^{d-1} = c_d \cdot n^{\frac{d-1}{d}}$

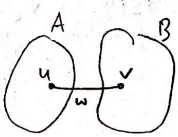
As for dim = 0, I will provide thoughts on why form should converge to some const.

First, let's write some bound(n) of an edge that exists in MST

Suppose w =  $\in$ MST

let's through it away so the MST splits into 2 parts A & B ($u \in A$, $v \in B$)

$A \sqcup B = G$, A & B = slice



$\Rightarrow$ $w \le$ all edges between A & B

there are $|A| \cdot |B|$ of them $\ge N-1$ $(N = |V|)$

this happens with probability $(1-w)^{|A||B|} \le (1-w)^{N-1}$

Now, suppose we have an edge $w \ge \frac{c}{N}$ (where c = const chosen later)

Scanned with CamScanner

$\Rightarrow$ this happens with prob. $\leq \left(1-\frac{c}{N}\right)^{N-1} \leq 2\cdot\left(1-\frac{c}{N}\right)^{N} =$

$= 2\left(1-\frac{1}{\left(\frac{N}{c}\right)}\right)^{N} = 2\left(1-\frac{1}{\left(\frac{N}{c}\right)}\right)^{\left(\frac{N}{c}\right)\cdot c} \xrightarrow[N\to+\infty]{} 2\cdot\left(\frac{1}{e}\right)^{c}$

Thus, taking c quite large (like, 10) we can guarantee that we don't have edges $> \frac{c}{N}$ in our MST with very high probability

This, in turn, explains while overall MST weight =

$= \sum_{w_i \in MST} w_i \leq (\max w_i)\cdot(\#MST) \leq \frac{c}{N}\cdot N = c$

(of course, smaller edges will be way less than $\frac{c}{N}$ as well as factor 2 and assumption $|A|\cdot|B| = N-1$ are very rude approximations).

More precise approach would consider

$$E[w(MST)] \leq (N-1)\,E[\max w \text{ in } MST] \leq (N-1)\cdot\int_0^1 w\cdot p(w)\,dw$$

where $p(w)$ is density of distribution of r.v. = max weight

we know that $\mathbb{P}(\max \text{ weight} > w) \leq (1-w)^{N-1}$

thus $\int_0^w p(x)\,dx = \mathbb{P}(\max \text{ weight} \leq w) \geq 1-(1-w)^{N-1}$

thus we could substite $p(w)$ with $p^*(w) = \left(1-(1-w)^{N-1}\right)' = (N-1)(1-w)^{N-2}$

in calculating $\int_0^1 w\,p(w)\,dw$ (which would increase result)

$\Rightarrow$ we have $(N-1)\cdot\int_0^1 (N-1)\,w\,(1-w)^{N-2} = (N-1)^2\int_0^1 w\underbrace{(1-w)^{N-2}}_{(1-w)^{N-2}-(1-w)^{N-1}}dw =$

$= (N-1)^2\left|_0^1 -\frac{(1-w)^{N-1}}{N-1} + \frac{(1-w)^N}{N}\right. = (N-1)^2\left(\frac{1}{N-1}-\frac{1}{N}\right) = \frac{N-1}{N}.$

Scanned with CamScanner

most probably, I made a mistake in this substitution (since we know that real answer is about 1.2) — I just wanted to give a try for a more rigorous approach

Also note that I use this bound in my code (in generate_graph() (int n)).

Same logics applies for dim = 2, 3, 4:

↳ (Bounds on max weights in these dimensions):

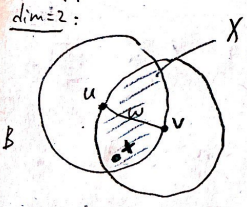
suppose we have $u, v \in D^{dim}$ = unit cube in dimension d

$dist(u,v) = w$

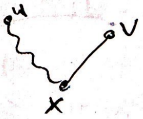
Let's draw 2 balls $B_w(u)$ & $B_w(v)$ and $X = B_w(u) \cap B_w(v)$

suppose there is a vertex x in X:

dim=2:



then either x and u or x and v are not connected in MST without w (otherwise there would be a cycle)

wlog it's v. =>  change 

since $dist(x,v) \le w$ this change would decrease MST weight

=> X doesn't contain any $x \in V$

this happens with prob. $(1 - Volume(X))^N$

Volume $\sim const(d) \cdot w^d$ (in case d=2 it's just $2w^2\left(\frac{\pi}{3} - \frac{\sqrt{3}}{4}\right)$)

Now, choosing some const k we have

$\mathbb{P}(\text{max weight} \ge k \cdot N^{\frac{1}{d}}) \le$

Scanned with CamScanner

$\leq \left(1 - \text{const}(d) \cdot \left(kN^{\frac{1}{d}}\right)^d\right)^N = \left(1 - \text{const}(d)\, k^d \cdot N^{-1}\right)^N \to \exp(-c(d)k^d)$ [6]

Therefore, weight (MST) $\leq kN^{-\frac{1}{d}} \cdot N = kN^{\frac{d-1}{d}}$ with high probability (how much close to 1 depends on k)

Part 3. Implementation / Algorithms

I chose Kruskall's algorithm.
Regardless of algorithm (Krusk. vs Prim) I would have to sort the edges.
For $N \leq 2^{11}$ I took full graph since this was both feasible in terms of time and memory.
However, for greater N taking full graph of distances would require $C_N^2 = O(N^2)$ memory at least to store them.
So, I had to optimize this: throwing away all long (heavy) edges according to the bounds I found in part 2.
Understanding that I will be working with a sparse graph because of this optimization I chose Kruskall's since it works fine on sparse graphs and easier to implement.

Optimization (Throwing away edges):

dim=0: here I explicitly use the bound described in Part 2
it can be found Bound(long long n);

Scanned with CamScanner

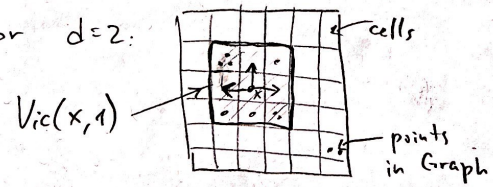
In cases of dim ≥ 2 I do the following:

(1) Split cube into base^d regular sub-cubes (size = 1/base)
base ∈ ℤ chosen in way that there are avrg-per-cell points in each of sub-cubes
(avrg-per-cube depends on d because we will be interested in the number of points of the vicinity of each sub-cubes, which will be $= 3^d \cdot$ avrg-per-cell)

(2) assign each point in Graph to the corresponding cell where it got into. I constructed struct cell for that

(3) for each $u \in G$ let's define v's vicinity as the set of all points that got into close / neighboring / same cell
This means Vic(x, dist) = observes all cells which coordinates do not differ more than distance (= 1 or 2) from coordinates of cell(x)

Picture for d=2:



(4) Our procedure for constructing graph with edges follows:
> Look over each point $x \in G$
> Observe Vic(x, dist) for dist = 1, 2 and count number

Scanned with CamScanner

of vertexes that appear to be "close" to x:
guaranteed closer than points outside the vicinity
(so approximately dist ≤ size(cell), but there is more precise bound)

> in case there are at least limit(d) of them
I put all these edges in our graph and continue
At this point I assume that this method provides great probability that we inserted all edges with x that could potentially be part of MST

> In case we found less than limit(d) of close points, I "play safe": push all edges (x,1) (x,2).. (x,N) in our graph so not to lose anything 100%

Why is this method with limit(d) reasonable / helps (?)
- First, splitting cube into sub-cubes helps a lot since this is optimization in terms of distance: I only observe vicinities assuming that big edges won't get into graph
- Second, method with limit(d) provides more robustness to exceptional cases when we could lose edges: in the case there are not enough points in the vicinity I just push all edges (from x)
The only subtle thing here is definition of „enough"

Scanned with CamScanner

For this, to define limit(d), I'm using Bounds from what's known "Kissing number" (wikipedia):

$k(2) = 6$
$k(3) = 12$
$k(4) = 24$

This is essentially the maximal (theoretical) possible degree for one vertex in our graph.

In other words, on plain ($\mathbb{R}^2$) it's not possible that some point has degree ≥ 7 in our MST.

(≥ 13 & ≥ 25 in $\mathbb{R}^3$ & $\mathbb{R}^4$)

Thus, after finding $\geq$ limit(d) vertexes that are guaranteed to be the closest to x it's even less (considerably less, but still probable) probable that we are loosing some edge than when just throwing away long edges

Note that in coding I take limit = $\frac{\text{limit(d)}}{2} + 1$

Because I only consider "shifts" in nonnegative direction so only points in upper half-plane will be observed

(+1 because we have 0 shift giving our point itself)

RunTime

After throwing away all long edges, we are left with (hopefully) linear number of edges: $M = O(N)$

this aligns with my Bounds: since practically for all vertexes (except for outliners) their degree in optimized graph

Scanned with CamScanner

is limited by limit(d), then $M = \frac{\sum deg(v_i)}{2} \le \frac{limit(d)}{2} N = O(N)$

(+ limited number of outliners gives + O(N) edges)

thus, eventually, $M = O(N)$ and since Kruskall works in $O(M \log n) = O(N \log n)$ time

while sorting of G will work in $O(M \log M) = O(N \log N)$ time

This aligns with practical results:

time for sorting + kruskall in case $N = 2^{18}$ $d = 4$ is near 6,3s

(while $2^{18} \cdot 18 \approx (2^{10})^2 \cdot 10 = (10^3)^2 \cdot 10 = 10^7$)

of course, it's kinda slow because our const in $O(N \log N)$ is big: we leave way more than N edges (approximately 40N in case $d = 4$)

Yet, the slowest part is generating the graph itself! I mean, generating and gathering all required edges because I'm doing it „safe"

For time statistics and explicit values of all trials you can just use flag = 1 (instead of 0):

For example, ./randmst 1 262144 5 4

says that average time for generating graph = 26.82 s
for sorting = 6.07 s
for finding MST = 0.24 s

And average task RunTime = 33.23 s (with 1 CPU usage ☺)

(So, maybe, this could be potentially faster if I knew how to parallel the process and speed up in ~8 times)

Scanned with CamScanner