4
Low-Stretch Spanning Trees

Given that shortest paths from a single source node s can be represented by a single shortest-path tree, can we get an analog for all-pairs shortest paths? Given a graph can we find a tree T that gives us the shortest-path distances between every pair of nodes? Does such a tree even exist? Sadly, the answer is negative—and it remains negative even if we allow this tree to stretch distances by a small factor, as we will soon see. However, we show that allowing randomization will allow us to circumvent the problems, and get low-stretch spanning trees in general graphs.

In this chapter, we consider *undirected* graphs $G = (V, E)$, where each edge e has a *non-negative* weight/length w_e. For all u, v in V, let $d_G(u, v)$ be the *distance* between u, v, i.e., the length of a shortest path in G from u to v. Observe that the set V along with the distance function d_G forms a metric space.

A metric space is a set V with a distance function d satisfying *symmetry* (i.e., $d(x, y) = d(y, x)$ for all $x, y \in V$) and the *triangle inequality* ($d(x, y) \leq d(x, z) + d(z, y)$ for all $x, y, z \in V$). Typically, the definition also asks for $x = y \iff d(x, y) = 0$, but we will merely assume $d(x, x) = 0$ for all x.

4.1 *Towards a Definition*

The study of low-stretch spanning trees is guided by two high level hopes:

1. Graphs have spanning trees that preserve their distances. That is, given G there exists a subtree $T = (V, E_T)$ with $E_T \subseteq E$ such that

$$d_G(u, v) \approx d_T(u, v) \qquad \text{for all } u, v \in V.$$

We assume that the weights of edges in E_T are the same as those in G.

2. Many **NP**-hard problems are much easier to solve on trees.

Supposing these are true, we have a natural recipe for designing algorithms to solve problems that depend only on distances in G: (1) find a spanning tree T preserving distances in G, (2) solve the problem on T, and then (3) return the solution (or some close cousin) with the hope that it is a good solution for the original graph.

4.1.1 *An All-Pairs Shortest Path Tree?*

The boldest hope would be to find an *all-pairs shortest path tree* T, i.e., one that ensures $d_T(u,v) = d_G(u,v)$ for all u,v in V. However, such a tree may not exist: consider K_n, the clique of n nodes, with unit edge lengths. The distance d_G satisfies $d_G(x,y) = 1$ for all $x \neq y$, and zero otherwise. But any subtree T contains only $n-1$ edges, so most pairs of vertices $x,y \in V$ lack an edge between them in T. Any such pair has a shortest-path distance $d_T(x,y) \geq 2$, whereas $d_G(x,y) = 1$.

4.1.2 *A First Relaxation: Low-Stretch Spanning Trees*

To remedy the snag above, let us not require distances in T be equal to those in G, but instead be within a small multiplicative factor $\alpha \geq 1$ of those in G.

Definition 4.1. Let T be a spanning tree of G, and let $\alpha \geq 1$. We call T a (deterministic) *α-stretch spanning tree* of G if

Exercise: show that if T is any subtree of G with the same edge weights, then $d_G(x,y) \leq d_T(x,y)$.

$$d_G(u,v) \leq d_T(u,v) \leq \alpha\, d_G(u,v).$$

holds for all $u,v \in V$.

Supposing we had such a low-stretch spanning tree, we could try our meta-algorithm out on the ***traveling salesperson problem (TSP)***: given a graph, find a closed tour that visits all the vertices, and has the smallest total length. This problem is **NP**-hard in general, but let us see how an α-stretch spanning tree of G gives us an an α-approximate TSP solution for G. The algorithm is simple:

Algorithm 7: TSP via Low-Stretch Spanning Trees

```
7.1 Find an $\alpha$-stretch spanning tree $T$ of $G$.
7.2 Solve TSP on $T$ to get an ordering $\pi_T$ on the vertices.
7.3 return the ordering $\pi_T$.
```

Solving the TSP problem on a tree T is trivial: just take an Euler tour of T, and let π_T be the order in which the vertices are visited. Let us bound the quality of this solution.

Claim 4.2. π_T is an α-approximate solution to the TSP problem on G.

Proof. Suppose that the permutation π_G minimizes the length of the TSP tour for G. The length of the resulting tour is

$$OPT_G := \sum_{i \in [n]} d_G(\pi_G(i), \pi_G(i+1)).$$

Since distances in the tree T are stretched by only a factor of α,

$$\sum_{i\in[n]} d_T(\pi_G(i), \pi_G(i+1)) \leq \alpha \cdot \sum_{i\in[n]} d_G(\pi_G(i), \pi_G(i+1)). \qquad (4.1)$$

Now, since π_T is the optimal ordering for the tree T, and π_G is some other ordering,

$$\underbrace{\sum_{i\in[n]} d_T(\pi_T(i), \pi_T(i+1))}_{OPT_T} \leq \sum_{i\in[n]} d_T(\pi_G(i), \pi_G(i+1)). \qquad (4.2)$$

Finally, since distances were only stretched in going from G to T,

$$\sum_{i\in[n]} d_G(\pi_T(i), \pi_T(i+1)) \leq \sum_{i\in[n]} d_T(\pi_T(i), \pi_T(i+1)). \qquad (4.3)$$

Putting it all together, the length of the tour given by π_T is

$$\sum_{i\in[n]} d_G(\pi_T(i), \pi_T(i+1)) \leq \alpha \cdot \sum_{i\in[n]} d_G(\pi_G(i), \pi_G(i+1)),$$

which is $\alpha \cdot OPT_G$. $\square$

Hence, if we had low-stretch spanning trees T with $\alpha \leq 1.49$, we would get the best approximation algorithm for the TSP problem. (Assuming we can find T, but we defer this for now.) However, you may have already noticed that the K_n example above shows that $\alpha < 2$ is impossible. But can we achieve $\alpha = 2$? Indeed, is there any "small" value for α such that for any graph G we can find an α-stretch spanning tree of G?

Sadly, things are terrible: take the cycle C_n, again with unit edge weights. Now any subtree T is missing one edge from C_n, say uv. The endpoints of this edge are at distance 1 in C_n, but $d_T(u,v) = n - 1$, since we have to go all the way around the cycle. Hence, getting $\alpha < (n-1)$ is impossible in general.

Exercise: show how to find, for any graph G, a spanning tree T with stretch $\alpha \leq n-1$.

4.1.3 *A Second Relaxation: Randomization to the Rescue*

Since we cannot get trees with small stretch deterministically, let us try to get trees with small stretch "on average". We amend our definition as follows:

Definition 4.3. A ***(randomized) low-stretch spanning tree*** of *stretch* α for a graph $G = (V, E)$ is a probability distribution $\mathcal{D}$ over spanning trees of G such that for all $u, v \in V$, we have

Henceforth, all references to low-stretch trees will only refer to this randomized version, unless otherwise specified.

$$d_G(u,v) \leq d_T(u,v) \qquad \text{for all } T \text{ in the support of } \mathcal{D}, \text{ and}$$
$$\mathbb{E}_{T\sim\mathcal{D}}[d_T(u,v)] \leq \alpha\, d_G(u,v) \qquad (4.4)$$

Observe that the first property must hold with probability 1 (i.e., it holds for all trees in the support of the distribution), whereas the second property holds only on average. Is this definition any good for our TSP example above? If we change the algorithm to sample a tree T from the distribution and then return the optimal tour for T, we get a randomized algorithm that is good in expectation. Indeed, (4.1) becomes

$$\sum_{i\in[n]} \mathbb{E}[d_T(\pi_G(i), \pi_G(i+1))] \le \alpha \cdot \sum_{i\in[n]} d_G(\pi_G(i), \pi_G(i+1)), \qquad (4.5)$$

because the stretch guarantees hold in expectation (and linearity of expectation). The rest of the inequalities hold unchanged, including (4.3)—which requires the probability 1 guarantee of Definition 4.6 (Do you see why?). Hence, we get

$$\underbrace{\sum_{i\in[n]} \mathbb{E}[d_G(\pi_T(i), \pi_T(i+1))]}_{\text{expected algorithm's tour length}} \le \alpha \cdot \underbrace{\sum_{i\in[n]} d_G(\pi_G(i), \pi_G(i+1))}_{OPT_G}. \qquad (4.6)$$

Even a randomized better-than-1.49 approximation for TSP would still be amazing! And the algorithmic template here works not just for TSP: any **NP**-hard problem whose objective is a linear function of distances (e.g., many other vehicle routing problems, or the k-median clustering problem) can be solved in this way. Indeed, the first approximation algorithms for many such problems came via low-stretch spanning trees.

Moreover, (randomized) low-stretch spanning trees arise in many different contexts, some of which are not obvious at all. E.g., they can be used to more efficiently solve "Laplacian" linear systems of the form $A\vec{x} = \vec{b}$, where A is the ***Laplacian matrix*** of some graph G. To do this, we let P be the Laplacian matrix of a low-stretch spanning tree of G, and then we solve the system $P^{-1}A\vec{x} = P^{-1}\vec{x}$ instead. This is called *preconditioning* with P. It turns out that this preconditioning allows certain algorithms for solving linear systems to converge faster to a solution. Time permitting, we will discuss this application later in the course.

4.2 *Low-Stretch Spanning Tree Construction*

But first, given a graph G, how can we find a randomized low-stretch spanning tree for G with a small value of α (and efficiently)? As a sanity check, let us check what we can do on the two examples from before:

1. For the complete graph K_n, choose a star graph centered at a uniformly random vertex of G. For any pair of vertices u, v, they are

A natural first attempt (at least for unweighted graphs) would be to try a uniformly random spanning tree. This does not work very well (which I think is not that surprising), even for the complete graph K_n (which I think is somewhat surprising). A result of Moon and Moser shows that for any pair of vertices $u, vinV(K_n)$, if we choose T to be one of the n^{n-2} spanning trees uniformly at random, the expected distance is $d_T(u,v) = \Theta(\sqrt{n})$.

at distance 1 in this star if either u or v is the center, else they are at distance 2. Hence the expected distance is $\frac{2}{n} \cdot 1 + \frac{n-2}{n} \cdot 2 = 2 - \frac{2}{n}$.

2. For the cycle C_n, choose a tree by dropping a single edge uniformly at random. For any edge uv in the cycle, there is only a 1 in n chance of deleting the edge from u to v. But when it is deleted, u and v are at distance $n-1$ in the tree. So

$$\mathbb{E}[d_T(u,v)] = \frac{n-1}{n} \cdot 1 + \frac{1}{n} \cdot (n-1) = 2 - \frac{2}{n}.$$

And what about an arbitrary pair of nodes u, v in C_n? We can use the exercise on the right to show that the stretch on other pairs is no worse!

Exercise: Given a graph G, suppose the stretch on all edges is at most α. Show that the stretch on all pairs of nodes is at most α. (Hint: linearity of expectation.)

While we will not manage to get $\alpha < 1.49$ for general graphs (or even for the above examples, for which the bounds of $2 - \frac{2}{n}$ are the best possible), we show that $\alpha \approx O(\log n)$ can indeed be achieved. The following theorem is the current best result, due to Ittai Abraham and Ofer Neiman:

Theorem 4.4. *For any graph G, there exists a distribution $\mathcal{D}$ over spanning trees of G with stretch $\alpha = O(\log n \log\log n)$. Moreover, the construction is efficient: we can sample trees from this distribution $\mathcal{D}$ in $O(m \log n \log\log n)$ time.*

Moreover, the stretch bound of this theorem is almost optimal, up to the $O(\log\log n)$ factor, as the following lower bound due to Alon, Peleg, Karp, and West shows.

Theorem 4.5. *For infinitely many n, there exist graphs G on n vertices such that any α-stretch spanning tree distribution $\mathcal{D}$ on G must have $\alpha = \Omega(\log n)$. In fact, G can be taken to be the n-vertex square grid, the n-vertex hypercube, or any n-vertex constant-degree expander.*

4.3 Bartal's Construction

The algorithm underlying Theorem 4.4 is quite involved, but we can give the entire construction of low-stretch trees for finite metric spaces.

Definition 4.6. A (randomized) low-stretch tree with stretch α for a metric space $M = (V, d)$ is a probability distribution $\mathcal{D}$ over trees over the vertex set V such that for all $u, v \in V$, we have

$$\begin{aligned} d(u,v) &\le d_T(u,v) \qquad \text{for all } T \text{ in the support of } \mathcal{D}, \text{ and} \\ \mathbb{E}_{T\sim\mathcal{D}}[d_T(u,v)] &\le \alpha\, d(u,v). \end{aligned} \tag{4.7}$$

The difference of this definition from Definition 4.6 is slight: we now have a metric space instead of a graph, and we are allowed to output any tree on the vertex set V (since the concept of subtrees doesn't make sense now). Note that given a graph G, we can compute its shortest-path metric (V, d_G) and then find a distribution over (non-spanning) trees that approximate the distance in G. So if we don't really need the spanning aspect in our low-stretch trees—e.g., as in the TSP example—we can use results for this definition.

We need one more piece of notation: for a metric space $M = (V, d)$, define its ***aspect ratio*** Δ to be

$$\Delta_M := \frac{\max_{u \neq v \in V} d(u,v)}{\min_{u \neq v \in V} d(u,v)}.$$

We will show the following theorem, due to Yair Bartal:

Theorem 4.7. *For any metric space $M = (V, d)$, there exists an efficiently sampleable α_B-stretch spanning tree distribution $\mathcal{D}_B$, where*

$$\alpha_B = O(\log n \log \Delta_M).$$

The proof works in two parts: we first show a good *low-diameter decomposition*. This will be a procedure that takes a metric space and a diameter bound D, and randomly partitions the metric space into clusters of diameter $\leq D$, in such a way that close-by points are unlikely to be separated. Then we show how such a low-diameter decomposition can be used recursively to constuct a low-stretch tree.

The ***diameter*** of a set S is $\max_{u,v \in S} d(u,v)$, i.e., the maximum distance between any two points in it.

4.3.1 *Low-Diameter Decompositions*

The notion of a low-diameter decomposition has become ubiquitous in algorithm design, popping up in approximation and online algorithms, and also in distributed and parallel algorithms. It's something worth understanding well.

Definition 4.8 (Low-Diameter Decomposition)**.** A ***low-diameter decomposition scheme*** (or LDD scheme) with parameter β for a metric $M = (V, d)$ is a randomized algorithm that, given a bound $D > 0$, partitions the point set V into "clusters" $C_1, \ldots, C_t$ such that

(i) for all $i \in \{1, \ldots, t\}$, the *diameter* of C_i is at most D, and

(ii) for all $x, y \in V$ such that $x \neq y$, we have

$$\Pr[x, y \text{ in different clusters}] \leq \beta \cdot \frac{d(x,y)}{D}.$$

Let's see a few examples, to get a better sense for the definition:

1. Consider a set of points on the real line. One way to partition the line into pieces of diameter D is simple: imagine making notches

on the line at distance D from each other, and then randomly shifting them. Formally, pick a random value $R \in [0, D]$ uniformly at random, and partition the line into intervals of the form $[Di + R, D(i+1) + R)$, for $i \in \mathbb{Z}$. A little thought shows that points x, y are separated with probability exactly $\frac{d(x,y)}{D}$.

2. The infinite 2-dimensional square grid with unit edge-lengths. One way to divide this up is to draw horizontal and vertical lines which are $D/2$ apart, and randomly shift as above. A pair x, y is separated with probability exactly $\frac{d(x,y)}{D/2}$ in this case. Indeed, this approach works for k-dimensional hypergrids (and k-dimensional ℓ_1-space) with probability $k \cdot \frac{d(x,y)}{D}$ — in this case the β parameter is at most the dimension of the space.

3. What about lower bounds? One can show that for the k-dimensional hypergrid, we cannot get $\beta = o(k)$. Or for a constant-degree n-vertex expander, we cannot get $\beta = o(\log n)$. Details to come soon.

Since the aspect ratio of the metric space is invariant to scaling all the edge lengths by the same factor, it will be convenient to assume that the smallest non-zero distance in d is 1, so the largest distance is Δ. The basic algorithm is then quite simple:

```
Algorithm 8: LDD($M = (V, d), D$)
8.1 $p \leftarrow \min(1, \frac{4 \log n}{D})$.
8.2 while there exist unmarked point do
8.3     $v \leftarrow$ any unmarked point.
8.4     sample $R_v \sim \text{Geometric}(p)$.
8.5     cluster $C_v \leftarrow \{\text{unmarked } u \mid d(v,u) < R_v\}$.
8.6     mark points in $C_v$.
8.7 return the resulting set of clusters.
```

Lemma 4.9. *The algorithm above ensures that*

1. *the diameter of every cluster is at most D with probability at least $1 - 1/n$, and*

2. *any pair $x, y \in V$ is separated with probability at most $2p\, d(x,y)$.*

Proof. To show the diameter bound, it suffices to show that $R_v \leq D/2$ for each cluster C_v, because then the triangle inequality shows that for any $x, y \in C_v$,

$$d(x,y) \leq d(x,v) + d(v,y) < D/2 + D/2 = D.$$

Now the probability that $R_v > D/2$ for one particular cluster is

We use that $1 - z \leq e^z$ for all $z \in \mathbb{R}$.

$$\Pr[R_v > D/2] = (1-p)^{D/2} \le e^{-pD/2} \le e^{-2\log n} = \frac{1}{n^2}.$$

By a union bound, there exists a cluster with diameter $> D$ with probability

$$1 - \Pr[\exists v \in V,\, R_v > D/2] \ge 1 - \frac{n}{n^2} = 1 - \frac{1}{n}.$$

To bound the probability of some pair u, v being separated, we use the fact that sampling from the geometric distribution with parameter p means repeatedly flipping a coin with bias p and counting the number of flips until we see the first heads. Recall this process is memoryless, meaning that even if we have already performed k flips without having seen a heads, the time until the first heads is still geometrically distributed.

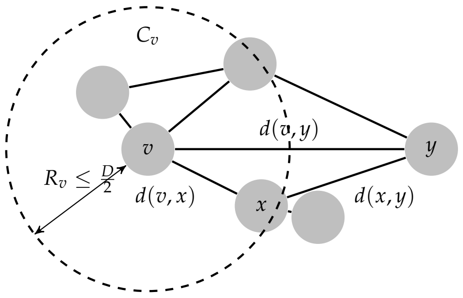

Figure 4.1: A cluster forming around v in the LDD process, separating x and y. To reduce clutter, only some of the distances are shown.

Hence, the steps of drawing R_v and then forming the cluster can be viewed as starting from v, where the cluster is a unit-radius ball around v. Each time we flip a coin of bias p: it is comes up heads we set the radius R_v to the current value, form the cluster C_v (and mark its vertices) and then pick a new unmarked point v; on seeing tails, we just increment the radius of v's cluster by one and flip again. The process ends when all vertices lie in some cluster.

For x, y, consider the first time when one of these vertices lies inside the current ball centered at some point, say, v. (This must happen at some point, since all vertices are eventually marked.) Without loss of generality, let the point inside the current ball be x. At this point, we have performed $d(v,x)$ flips without having seen a heads. Now we will separate x, y if we see a heads within the next $\lceil d(v,y) - d(v,x) \rceil \le \lceil d(x,y) \rceil$ flips—beyond that, both x, y will have been contained in v's cluster and hence cannot be separated. But the probability of getting a heads among these flips is at most (by a union bound)

$$\lceil d(x,y) \rceil\, p \le 2d(x,y)\, p \le 8 \log n \, \frac{d(x,y)}{D}.$$

(Here we used that the minimum distance is 1, so rounding up distances at most doubles things.) This proves the claimed probability of separation. $\square$

Recall that we wanted the diameter bound with probability 1, whereas Lemma 4.9 only ensures it with high probability. Here's a quick fix to this problem: repeat the above process until the returned partition has clusters of diameter at most D. The probability of any pair u, v being separated by this last run of Algorithm 8 is at most the probability of u, v being separated by any of the runs, which is at most $p\, d(u,v)$ times the expected number of runs,

$$p\, d(u,v) \cdot (1/(1-1/n)) \le 2p\, d(u,v) = O(\log n)\, \frac{d(u,v)}{D}.$$

Lemma 4.10. *The low-diameter decomposition scheme above achieves parameter $\beta = O(\log n)$ for any metric M on n points.*

4.3.2 *Low-Stretch Trees Using LDDs*

Now we can use the low-diameter decomposition scheme to get a low-stretch tree (LST). Here's the high-level idea: given a metric with diameter Δ, use an LDD to decompose it into clusters with diameter $D \le \Delta/2$. Build a tree recursively for each of these clusters, and then combine these trees into one tree for the entire metric.

Recall we assumed that the metric had minimum distance 1 and maximum distance Δ. Formally, we invoke the procedure LST below with the parameters LST(metric M, $\lceil \log_2 \Delta \rceil$).

Algorithm 9: LST(metric M = (V,d), $D = 2^\delta$)

```
    Input: Invariant: diameter$(M) \le 2^\delta$
9.1 if $|V| = 1$ then
9.2     return tree containing the single point in $V$.
9.3 $C_1, \dots, C_t \leftarrow$ LDD$(M, D = 2^{\delta-1})$.
9.4 for $j$ in $\{1, \dots, t\}$ do
9.5     $M_j \leftarrow$ metric $M$ restricted to the points in $C_j$.
9.6     $T_j \leftarrow$ LST$(M_j, \delta - 1)$.
9.7 Add edges of length $2^\delta$ from root $r_1$ for tree $T_1$ to the roots of
      $T_2, \dots, T_t$.
9.8 return resulting tree rooted at $r_1$.
```

We are ready to prove Theorem 4.7; we will show that the tree has expected stretch $O(\beta \log \Delta)$, and that it does not shrink any distances. In fact, we show a slightly stronger guarantee.

Lemma 4.11. *If the random tree T returned by some call LDD(M', δ) has root r, then (a) every vertex x in T has distance $d(x, r) \le 2^{\delta+1}$, and (b) the expected distance between any $x, y \in T$ has $\mathbb{E}[d_T(x, y)] \le 8\delta\beta\, d(x, y)$.*

Proof. The proof is by induction on δ. For the base case, the tree has a single vertex, so the claims are trivial. Else, let x lie in cluster C_j, so inductively the distance to the root of the tree T_i is $d(x, r_i) \le 2^{(\delta-1)+1}$. Now the distance to the new root r is at most 2^δ more, which gives $2^\delta + 2^\delta = 2^{\delta+1}$ as claimed.

Moreover, any pair x, y is separated by the LDD with probability $\beta\, \frac{d(x,y)}{2^{\delta-1}}$, in which case their distance is at most

$$d(x, r) + d(r, y) \le 2^{\delta+1} + 2^{\delta+1} = 4 \cdot 2^\delta.$$

Else they lie in the same cluster, and inductively have expected distance at most $8(\delta-1)\beta\, d(x,y)$. Hence the expected distance is

$$\begin{aligned}\mathbb{E}[d(x,y)] &\leq \Pr[x,y \text{ separated}] \cdot 4 \cdot 2^{\delta} + \\ &\quad \Pr[x,y \text{ not separated}] \cdot 8(\delta-1)\beta\, d(x,y) \\ &\leq \beta\, \frac{d(x,y)}{2^{\delta-1}} \cdot 4\, 2^{\delta} + 8(\delta-1)\beta\, d(x,y) \\ &= 8\,\delta\,\beta\, d(x,y). \quad \square\end{aligned}$$

This proves Theorem 4.7 because $\beta = O(\log n)$, and the iniitial call on the entire metric defines $\delta = O(\log \Delta)$. In fact, if we have a better LDD (with smaller β), we immediately get a better low-stretch tree. For example, shortest-path metrics of planar graphs admit an LDD with parameter $\beta = O(1)$; this shows that planar metrics admit (randomized) low-stretch trees with stretch $O(\log \Delta)$.

It turns out this factor of $O(\log n \log \Delta)$ can be improved to $O(\log n)$—this was done by Fakcharoenphol, Rao, and Talwar. Moreover, the bound of $O(\log n)$ is tight: the lower bounds of Theorem 4.5 continue to hold even for low-stretch non-spanning trees.

4.4 *Metric Embeddings: a.k.a. Simplifying Metrics*

We just how to approximate a finite metric space with a simpler metric space, defined over a tree. (Loosely, "every metric space is within $O(\log n)$ of some tree metric".) And since trees are simpler metrics, both conceptually and algorithmically, such an embedding can help design algorithms for problems on metric spaces.

This idea of approximating metric spaces by simpler ones has been extensively studied in various forms. For example, another famous result of Jean Bourgain (with an extension by Jirka Matoušek) shows that any finite metric space on n points can be embedded into ℓ_p-space with $O((\log n)/p)$ distortion [1]. Moreover, the ***Johnson-Lindenstrauss Lemma***, which we will see in a future chapter, shows that any n point-submetric of Euclidean space can be embedded into a (low-dimensional) Euclidean space of dimension at most $O(\log n/\epsilon^2)$, such that distances between points are distorted by a factor of at most $1 \pm \epsilon$ [2]. Since geometric spaces, and particularly, low-dimensional Euclidean spaces, are easier to work with and reason about, these can be used for algorithm design as well.

[1]

[2]

4.4.1 *Historical Notes*

To be cleaned up. Elkin et al. [3] gave the first polylog-stretch *spanning* trees, which took eight years following Bartal's construction. (The first low-stretch spanning trees had stretch $2^{O(\sqrt{\log n \log\log n})}$ by Alon et al. [4], which is smaller than n^{ϵ} for any $\epsilon > 0$ but larger than

[3]

[4]

polylogarithmic, i.e., $(\log n)^C$ for any $C > 0$.)