Dynamic Light Spanners in Doubling Metrics

Sujoy Bhore
Department of Computer Science & Engineering, Indian Institute of Technology Bombay, India

Jonathan Conroy
Department of Computer Science, Dartmouth College, Hanover, NH, USA

Arnold Filtser
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

Abstract

A t-spanner of a point set X in a metric space $(\mathcal{X}, \delta)$ is a graph G with vertex set P such that, for any pair of points $u, v \in X$, the distance between u and v in G is at most t times $\delta(u, v)$. We study the problem of maintaining a spanner for a dynamic point set X – that is, when X undergoes a sequence of insertions and deletions – in a metric space of constant doubling dimension. For any constant $\varepsilon > 0$, we maintain a $(1+\varepsilon)$-spanner of P whose total weight remains within a constant factor of the weight of the minimum spanning tree of X. Each update (insertion or deletion) can be performed in $\text{poly}(\log \Phi)$ time, where Φ denotes the aspect ratio of X. Prior to our work, no efficient dynamic algorithm for maintaining a light-weight spanner was known even for point sets in low-dimensional Euclidean space.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Routing and network design problems

Keywords and phrases Dynamic data structures, spanners, light-weight, Euclidean metrics, doubling metrics

Digital Object Identifier 10.4230/LIPIcs.SoCG.2026.13

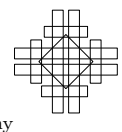
Related Version *Full Version*: https://arxiv.org/abs/2603.23490

Funding *Sujoy Bhore*: Work supported in part by ANRF ARG-MATRICS, Grant 002465.
Jonathan Conroy: Supported in part by U.S. National Science Foundation Grant No. CCF-2443017.
Arnold Filtser: This research was supported by the ISRAEL SCIENCE FOUNDATION (grant No. 1042/22).

1 Introduction

Let $X \subset \mathcal{X}$ be a set of points in the metric $(\mathcal{X}, \delta_X)$. A graph $G = (X, E, \delta)$ with vertex set X is a *geometric graph* if the weight function on the edges is the distance function δ. In other words, the weight of each edge is the distance between its endpoints in $\mathcal{X}$. Observe that if G is a geometric graph, then by the triangle inequality, for every $x, y \in X$, $\delta_G(x, y) \geq \delta(x, y)$. Given a parameter t, called the *stretch*, we say that G is a (geometric) *t-spanner* for X if $\delta_G(x, y) \leq t \cdot \delta(x, y)$ for every $x, y \in X$. The natural goal is to construct, for a given stretch t, a t-spanner with small number of edges and a small total weight.

Lightness and *sparsity* are two central measures for evaluating spanners. For a spanner $G = (X, E)$, the lightness is the ratio $w(G)/w(\text{MST})$ between the total weight of G and the weight of a minimum spanning tree on X. The sparsity of G is the ratio $|E(G)|/|E(\text{MST})| \approx |E(G)|/|X|$, comparing the number of edges of G to that of an MST. Since every spanner (with finite stretch) is connected and therefore contains a spanning tree, the lightness and sparsity of G constitute a measure of how close a spanner (specifically $w(G)$ and $|E(G)|$) to the optimal weight and the optimal number of edges of a spanner with finite stretch. A long line of research has studied spanners in Euclidean spaces [5, 49, 41, 29, 32, 14, 45],



doubling [35, 21, 51, 36, 33, 19, 13], planar and minor-free metric [18, 39, 25, 12], and general graphs [48, 4, 30, 28, 24, 33, 44, 16]. Of particular interest, we focus on *low-dimensional Euclidean space*, and more generally on *doubling metrics*.

Doubling metrics. Let $B(x,r) = \{y|\delta(x,y) \leq r\}$ be a ball of radius $r \geq 0$ centered at a point $x \in \mathcal{X}$. A metric $(\mathcal{X}, \delta)$ has *doubling dimension* d if for every $r > 0$, every ball of radius r can be covered by at most 2^d balls of radius $r/2$: $\forall x \in X, r > 0,\ \exists Y \subseteq X : |Y| \leq 2^d$ and $B_X(x,r) \subseteq \cup_{y \in Y} B_X(y, r/2)$. The study of spanners in doubling metrics was initiated by Gao, Guibas, and Nguyen [35], who proved that any n-point set in a metric space of doubling dimension d admits a $(1+\varepsilon)$-spanner with $\varepsilon^{-O(d)}n$ edges. Smid [51], via an analysis of the *greedy algorithm*, showed that greedy $(1+\varepsilon)$-spanners have $\varepsilon^{-O(d)}n$ edges and achieve lightness $\varepsilon^{-O(d)} \log n$. While it has long been known that d-dimensional Euclidean point sets admit $(1+\varepsilon)$-spanners with lightness $\varepsilon^{-O(d)}$, it remained a major open problem whether an analogous result holds for metric spaces of low doubling dimension. In a major breakthrough, Gottlieb [36] proved that any metric space of doubling dimension d admits a $(1+\varepsilon)$-spanner with lightness $(d/\varepsilon)^{O(d)}$. Gottlieb devised a complicated spanner construction instead of the classic greedy spanner. Later, Filtser and Solomon [33] showed that the greedy spanner is existentially optimal, concluding that it has lightness $(d/\varepsilon)^{O(d)}$ as well. Finally, Borradaile, Le, and Wulff-Nilsen [19] showed the greedy spanner has weight $\varepsilon^{-O(d)}$, which is optimal.

Dynamic Spanners. The problem of maintaining a sparse $(1+\varepsilon)$-spanner for a point set X under insertions and deletions has also been studied over the years. Already back in 1999, Arya et al. [6] obtained polylogarithmic update time in a restricted model with random updates (a deletion removes a uniformly random point from S, and an insertion results in a uniformly random point in the updated set). Later, Bose et al. [20] studied the problem in the context of θ-graphs, and presented an algorithm that supports insertions only and achieves polylogarithmic update time. Gao et al. [35] studied kinetic and dynamic geometric spanners for points in bounded doubling dimensions, and obtained a fully dynamic spanner with worst-case update time $O(\log \Phi)$, where Φ is the *aspect ratio* of the point set X, defined as the ratio between the maximum and minimum pairwise distances in X. Krauthgamer and Lee [43] obtained similar results for proximity search, where the time bounds of their dynamic updates depend also in the aspect ratio. For proximity search, later, Cole and Gottlieb [26] removed the dependency on the aspect ratio. For the Euclidean plane, Abam et al. [1] presented an algorithm with $O(\log n)$ update time. For doubling metrics, Roditty [50] presented the first fully dynamic algorithm for maintaining a geometric $(1+\varepsilon)$-spanner, with update time depending only on the number of points in X. The algorithm obtained in [50] supports insertions in amortized $O(\log n)$ time and deletions in amortized $\tilde{O}(n^{1/3})$ time where $\tilde{O}$ hides polylogarithmic factors, and maintains spanner with $O(n/\varepsilon^d)$ edges. Gotllieb and Roditty [37] gave a dynamic spanner that supports insertions in $O\big(\varepsilon^{-O(d)} \log^2 n\big)$ amortized time and deletions in $O\big(\varepsilon^{-O(d)} \log^3 n\big)$ amortized time. Later, in [38],[1] they gave a $(1+\varepsilon)$-spanner of constant degree that supports updates in $\varepsilon^{-O(d)} \log n$ worst-case time.

Roughly speaking, these "classic" constructions of dynamic spanners for doubling metric are based on a *net tree* (see Section 2). While it is straightforward to construct a sparse spanner from a net tree, this tool is too rough to obtain lightness. Indeed, in Figure 1 we illustrate the net tree of the path graph, and show that it has lightness $\Omega(\log n)$. A more

[1] Gotllieb and Roditty [38] additionally assume that they can query the distance between a deleted point and a non-deleted point, which (for example) is possible for Euclidean metrics.

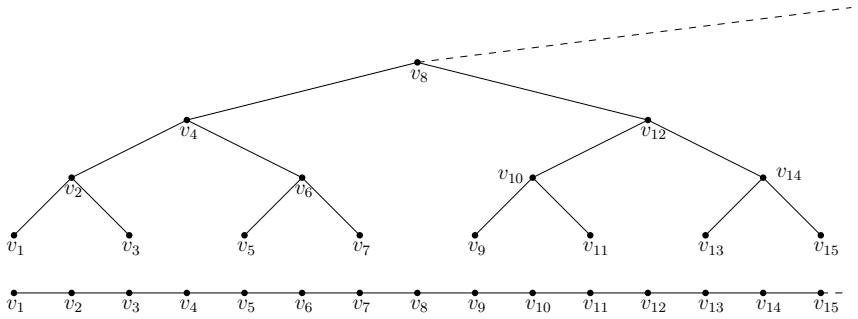


Figure 1 On bottom: The path graph P_n. All edges are unit weight. On top: a depiction of the net tree T of P_n. There is a laminar hierarchy of nets; the 2^i-net is $N_i = \{v_j \in P_n \mid j \mod 2^i = 0\}$ (all vertices at hight i in the tree and above). The net-tree spanner contains all the tree edges, and many additional edges. The weight of the edge $\{v_i, v_j\}$ is $|i-j|$. The tree T has depth $\log n$, and the total weight of the edges going from depth i vertices to depth $i+1$ is $\Theta(n)$. Thus $w(T) = \Theta(n \cdot \log n)$. As the weight of the MST of P_n is $n-1$, the net-tree spanner has lightness $\Omega(\log n)$.

recent approach to constructing dynamic sparse spanners is using *locality-sensitive orderings (LSO)* (introduced by Chan, Har-Peled, and Jones [22]). In the full version of the paper, we summarize the history of LSO, and prove that LSO-based spanners have logarithmic lightness (even for $\mathbb{R}^1$).

While there is a substantial body of work on dynamic spanners in Euclidean metrics and, more generally, in doubling metrics, as summarized above, there is comparatively little work on dynamic *light* spanners. Controlling lightness is arguably a significantly harder objective, as evidenced by the extensive literature in the offline setting (see, e.g., [45, 5, 49, 18, 44, 47, 13, 33, 36, 29, 14, 23] and the references therein).

▶ **Question 1.** *Can we design a dynamic algorithm that maintains a* light-weight *spanner for a point set under dynamic updates in doubling metrics with polylogarithmic update time?*

Partial progress toward Question 1 was made by Eppstein and Khodabandeh [31], who showed that one can maintain a *low-recourse* light spanner for dynamic point sets in low-dimensional Euclidean space. The *recourse* of a dynamic spanner is the number of edges that change after each insertion or deletion. They obtained a construction with amortized $O(1)$ recourse per insertion and amortized $O(\log \Phi)$ recourse per deletion. However, their result addresses recourse rather than efficiency: the actual time to update the spanner can be very large (no faster than just recomputing a spanner from scratch), and therefore their work does not resolve the question of achieving polylogarithmic (or even sublinear) update time.

1.1 Our Contribution

We resolve Question 1 in the affirmative. We say a set of points X is (a, b)-bounded if $\min_{u,v \in X} \delta(u, v) \geq a$ and $\max_{u,v \in X} \delta(u, v) \leq b$.

▶ **Theorem 2.** *Let $(\mathcal{X}, \delta)$ be a metric space with doubling dimension d, and let $\varepsilon > 0$ and $\Phi > 0$ be parameters. Let $X \subseteq \mathcal{X}$ be a set of points undergoing insertions and deletions, such that at all times X is $(1, \Phi)$-bounded. We maintain a $(1+\varepsilon)$-spanner L of X with lightness at most $\varepsilon^{-O(d)}$, where each insertion/deletion to X can be processed in time $(\varepsilon^{-1} \cdot \log \Phi)^{O(d)}$.*

For the sake of simplicity, we state our theorem as though the maximum distance Φ must be given in advance, and the minimum distance between points in X must be at least 1. Using standard techniques [43, 31] we could remove these restrictions at no loss in the running

time: we obtain a data structure whose update time depends adaptively on the aspect ratio Φ of the current point set X (see the full version of the paper). Additionally, we remark that we do not need to know the doubling dimension d in advance; the value d only appears in the runtime analysis.

Techniques. Our proof begin somewhat similarly to Eppstein and Khodabandeh [31]: roughly speaking, we seek to maintain a *greedy spanner* on top of a *dynamic sparse spanner* of points X (crucially, this spanner is much more structured than just the greedy spanner on the complete geometric graph on X). In Section 3, we first give a new (and arguably much simpler[2]) algorithm and analysis that such a spanner can be maintained with bounded recourse. After each insertion/deletion, we identify $O(\log \Phi)$ edges (u, v) of our greedy spanner L which might have to change, to satisfy the greedy spanner invariants: roughly speaking, we examine the distance $\delta_{L<}(u, v)$ between u and v in the graph consisting of all of edges of L that are shorter than (u, v)[3], and we insert or delete (u, v) to L based on whether $\delta_{L<}(u, v) \leq (1 + \varepsilon)\delta(u, v)$. Our key technical contribution (in Section 4) is to show that we can efficiently maintain estimates of $\delta_{L<}(u, v)$ even as we insert or delete edges into L.

1.2 Related Work

Dynamic Graph Spanners. In general graphs, the goal is to maintain, under edge updates, for any integer $k \geq 2$, a spanner with stretch $2k - 1$ and $\widetilde{O}(n^{1+1/k})$ edges. Assuming the Erdős–Girth conjecture, this trade-off is essentially optimal. The dynamic spanner problem was initiated by Ausiello et al. [7], and has since been studied extensively; see, for example, [8, 17, 27, 10, 34]. However, these works focus primarily on *sparsity*, and the resulting techniques do not naturally extend to maintaining *light* spanners. Moreover, due to inherent degree barriers in general graphs, this line of work exclusively considers *edge updates*. In contrast, in geometric and other structured metric spaces, vertex updates are often the more natural model, since the underlying graph need not be maintained explicitly. This distinction is crucial for light spanners, whose construction and analysis rely fundamentally on geometric structure rather than purely combinatorial sparsification.

Online Spanners. Spanners have also been studied extensively in the closely related *online* model. In this setting, points from a metric space arrive one by one, and at each step, the algorithm is given the subset of points that arrived so far. The goal is to maintain a t-spanner at all times. Unlike in the fully dynamic setting, the algorithm is allowed to *add* edges when a new point arrives, but may never remove previously added edges. In addition, the total number of points n is not known in advance. A number of algorithms and lower bounds have been obtained for this model, which addressed both sparsity and lightness guarantees along with stretch, under online constraints; see, e.g., [11, 15, 2, 3, 9, 40, 46]. These results achieve near-optimal trade-offs in the online regime and highlight the fundamental differences between incremental constructions and fully dynamic settings.

---

[2] We avoid the *bucketing* technique in the algorithm of [31] and instead work with all spanner edges at once; we maintain spanner invariants that work in doubling metrics and not just Euclidean metrics (unlike the *leapfrog property* used in [31]); and finally, we give a simple worst-case bound on the recourse of $O(\log \Phi)$, rather than using a potential function to bound the amortized recourse.

[3] Our description here assumes we want to maintain a greedy spanner, but actually we maintain a slightly more robust type of spanner which we call the *delayed greedy spanner*: we work with distances $\delta^*(u, v)$ rather than $\delta_{L<}(u, v)$, as defined in Section 2.2. We do this to control cascading changes to L.

2 Net trees and greedy spanners

Preliminaries. Throughout this paper, let $M = (\mathcal{X}, \delta)$ be a metric space with constant doubling dimension $d = O(1)$. For any vertex $x \in \mathcal{X}$ and radius $r > 0$, we let $B(x, r)$ denote the set of points within distance r of x; that is, $B(x, r) \coloneqq \{y \in \mathcal{X} : \delta(x, y) \leq r\}$. If G is a (geometric) graph, we use the notation $\delta_G(\cdot, \cdot)$ to denote shortest-path distances on G. For any positive integer n, we write $[n]$ to mean $\{i \in \mathbb{Z} : 0 \leq i \leq n\}$. Throughout this paper, we fix some sufficiently small positive integer $\varepsilon > 0$ and aim to construct a light $(1 + O(\varepsilon))$-spanner; our Theorem 2 will follow after rescaling $\varepsilon \leftarrow \varepsilon/O(1)$.

2.1 Net-Tree Spanners

In this subsection, we review the classic net-tree spanner [35].

Net tree. Let $X \subseteq \mathcal{X}$ be a set of points. Assume that the smallest distance between points in X is 1, and the largest distance is Φ; for simplicity we assume $\Phi = 2^i$ for some integer i. A *Δ-net* of X is a set of points N such that every point $x \in X$ is within distance Δ of some point in N, and any two net points in N are at distance greater than Δ. A Δ-net can be constructed greedily. A *net hierarchy* of X is a hierarchical sequence of nets $(N_0, N_1, \ldots, N_{\log \Phi})$ such that every N_i is a subset of N_{i-1}, and N_i is a 2^i-net of N_{i-1}; we take $N_0 = X$. Observe that (by a geometric series) every point in X is within distance 2^{i+1} of some net point in N_i. Moreover, the *packing bound* of doubling metrics implies that there are not too many points close together in each Δ-net.

▶ **Observation 3** (Packing bound). *Let N be a set of points that are pairwise at distance at least Δ. For any $\alpha \geq 1$ and $x \in \mathcal{X}$, there are at most $\alpha^{O(d)}$ points in $B(x, \alpha \cdot \Delta) \cap N$.*

An implicit representation of the net hierarchy $\mathcal{N}$ can be maintained using a data structure called the *net tree* [35, 43, 42]. We will sometimes abuse terminology and call $\mathcal{N}$ a net tree.

Net-tree spanner. The net-tree spanner was introduced (under the name "deformable spanner") by Gao, Guibas, and Nguyen [35]; similar ideas were independently designed by Krauthgamer and Lee [43]. The *ε-net-tree spanner* of X with respect to the net tree $\mathcal{N} = (N_0, \ldots, N_{\log \Phi})$ is the graph with vertex set X and with the following edge set: for every scale $i \in [\log \Phi]$, add an edge between any two vertices $u, v \in N_i$ if $\delta(u, v) \leq c \cdot 2^i$, where $c \coloneqq 4 + 16\varepsilon^{-1}$.

▶ **Lemma 4** ([35, Theorem 3.2]). *An ε-net-tree spanner of X is a $(1 + \varepsilon)$-spanner of X.*

For any pair of vertices (u, v), we say the *scale* of (u, v) is the value i such that $\delta(u, v) \in [2^{i-1}, 2^i)$. Observe that an edge (u, v) at scale i in an ε-net-tree spanner has both endpoints in the $O(\varepsilon) \cdot 2^i$-net of the net tree (not the 2^i-net). The packing bound implies that every vertex in the ε-net-tree spanner has degree at most $\varepsilon^{-O(d)} \cdot \log \Phi$. In fact, using a charging argument, one can show that the ε-net-tree spanner contains at most $\varepsilon^{-O(d)} \cdot n$ edges: that is, it is *sparse*. We frequently denote the ε-net-tree spanner as S.

Dynamic maintenance. Gao, Guibas, and Nguyen [35] showed that the net tree spanner can be maintained dynamically. We summarize their guarantee below.

▶ **Lemma 5** (Rephrasing of [35, Theorem 4.2]). *There is a data structure that implicitly maintains a hierarchy of nets $\mathcal{N} = (N_0, N_1, \ldots, N_{\log \Phi})$ of a point set X undergoing insertions and deletions. Each insertion or deletion can be processed in $O(\log \Phi)$ time. Moreover*[4]*, for any constant $\varepsilon > 0$, one can maintain an ε-net-tree spanner of X with respect to $\mathcal{N}$, in $O(\log \Phi)$ time per insertion or deletion.*

We will make use of Lemma 5, but we will have to slightly unwrap the black box – we will need some control on how the net tree $\mathcal{N}$ changes after each insertion or deletion. Fortunately, [35] update the net tree $\mathcal{N}$ in a simple and structured way. Inserting a point x into X has the following result, described in Figure 2.

$i_0 \leftarrow$ smallest scale such that the scale-i_0 net $N_{i_0} \in \mathcal{N}$ satisfies $\delta(x, N_{i_0}) < 2^{i_0}$
update $\mathcal{N}$ by inserting x into net $N_i \in \mathcal{N}$ for all $i < i_0$

Figure 2 Pseudocode for [35] algorithm to update net $\mathcal{N}$ after inserting point x into X.

Deleting a point x from X is described in Figure 3.

update $\mathcal{N}$ by deleting x from every net $N_i \in \mathcal{N}$
for every scale $i \leftarrow 1, 2, \ldots, \log \Phi$:
  while there exists some point $y \in N_{i-1}$ with $\delta(u, N_i) > 2^i$, add y to N_i

Figure 3 Pseudocode for [35] algorithm to update net $\mathcal{N}$ after deleting point x from X.

We will not discuss the details of the data structures or algorithms that [35] use to implement these two operations efficiently. We have unwrapped the [35] black box so that we can prove that every insertion/deletion of a point x only changes $\mathcal{N}$ "locally" nearby x.

▶ **Lemma 6.** *Let $\mathcal{N}^{\text{old}} = (N_0^{\text{old}}, \ldots, \mathcal{N}_{\log \Phi}^{\text{old}})$ be a net tree for point set X. Let $\mathcal{N} = (N_0, \ldots, N_{\log \Phi})$ be the net tree produced from $\mathcal{N}^{\text{old}}$ by the [35] algorithm after a point x is inserted into (resp. deleted from) X. For any scale $i \in [\log \Phi]$, every point y in the symmetric difference of N_i and N_i^{old} satisfies $\delta(y, x) \le 2^i$.*

Proof. When x is inserted into X (according to the pseudocode in Figure 2), the lemma follows trivially from the [35] algorithm description: the only point that could differ between N_i and N_i^{old} is x. We now argue that the lemma holds when x is deleted from X (according to the pseudocode in Figure 3). By construction, the only point that could be removed from N_i^{old} is x; that is, the set $N_i^{\text{old}} \setminus N_i$ is either $\varnothing$ or $\{x\}$. So it remains to show that every point y in $N_i \setminus N_i^{\text{old}}$ satisfies $\delta(y, x) \le 2^i$. The proof is by induction on i. When $i = 0$, every point in $\texttt{DS}.X$ is contained in N_0^{old}, so $N_0 \setminus N_0^{\text{old}} = \varnothing$ and the claim holds trivially. Now consider $i > 0$. Consider some point $y \in N_i \setminus N_i^{\text{old}}$. By the algorithm description, the point y satisfies

$$\delta(y, N_i^{\text{old}} \setminus \{x\}) > 2^i.$$

There are two cases. In the first case, suppose $y \in N_{i-1}^{\text{old}}$. Because N_i^{old} is a 2^i-net for N_{i-1}^{old}, we have $\delta(y, N_i^{\text{old}}) \le 2^i$. We conclude that $\delta(y, x) \le 2^i$ as desired. In the second case, suppose $y \notin N_{i-1}^{\text{old}}$; thus $y \in N_{i-1} \setminus N_{i-1}^{\text{old}}$, and by induction we have that $\delta(y, x) \le 2^{i-1} < 2^i$. ◀

[4] Note that the phrasing of [35, Theorem 4.2] does not explicitly separate the maintenance of the net hierarchy $\mathcal{N}$ from the maintenance of the spanner (they just state the existence of a dynamic spanner); however, our phrasing of Lemma 5 is immediate from their proof. In our application, we will need to dynamically maintain an ε_1-net-tree spanner of X and an ε_2-net-tree spanner of X (for two different values $\varepsilon_1 \neq \varepsilon_2$) *with respect to the same net tree* $\mathcal{N}$. This is possible using the data structure of [35].

We need one more guarantee from [35] (see full paper for details).

▷ Claim 7. Consider the data structure of [35] that maintains a hierarchy of nets $(N_0, N_1, \ldots, N_{\log \Phi})$ of X and an ε-net-tree spanner. Let $c_\varepsilon = 4 + 16\varepsilon^{-1}$. Given a scale $i \in [\log \Phi]$ and a query point $x \in \mathcal{X}$, one can find all vertices in $N_i \cap B(x, 100 \cdot c_\varepsilon \cdot 2^i)$ in time $\varepsilon^{-O(d)} = O(1)$.

2.2 Invariants for Our Light Spanner: Delayed Greedy Spanner

While the net-tree spanner S is sparse and can be maintained dynamically, it could have lightness $\Omega(\log n)$ (see Figure 1). We aim to find a subgraph of S with $O(1)$ lightness. Intuitively, we want to maintain a greedy $(1+\varepsilon)$-spanner H of the graph S: the greedy spanner (and even the *approximate* greedy spanner) are light [19, 33]. Recall that the greedy spanner initializes $H \leftarrow \varnothing$ and processes edges in S from smallest to largest; when we process (u, v), we add (u, v) to H iff $\delta_H(u, v) > (1+\varepsilon)\delta_S(u, v)$. In other words, the greedy spanner satisfies: for every edge (u, v) in S, letting $H_{<(u,v)}$ denote the set of edges in H with smaller[5] lengths than (u, v), we have $\delta_{H_{<(u,v)}}(u, v) > (1+\varepsilon)\delta_S(u, v)$ iff $(u, v) \in L$.

Delayed greedy spanner. We maintain a light spanner, denoted $S \subseteq S$ that doesn't obey the greedy spanner invariants exactly; rather, we define a more robust set of invariants. For any scale $i \in [\log \Phi]$, we define the set of edges $L[i] \subseteq L$ to be the set of all edges in L with *scale strictly less than* i; that is, every edge in $L[i]$ has length at most 2^{i-1}. For any edge (u, v) of the ε-net-tree spanner at scale i, we define $\delta^*(u, v) := \delta_{L[i]}(u, v)$; recall that $\delta_{L[i]}(\cdot, \cdot)$ denotes the shortest-path metric in the geometric graph induced by $L[i]$. To construct our light spanner, we maintain a subset of edges $L \subseteq S$ with the following properties:

▶ **Invariant 8.** *Every edge $(u, v) \in S \setminus L$ satisfies $\delta^*(u, v) \leq (1+\varepsilon) \cdot \delta(u, v)$.*

▶ **Invariant 9.** *Every edge $(u, v) \in L$ satisfies $\delta^*(u, v) > (1 + \frac{\varepsilon}{3}) \cdot \delta(u, v)$.*

If L satisfies Invariants 8 and 9, we call L a *delayed greedy spanner* of S (because there is a delay between the time an edge is added to the spanner and the time it is taken into account when making the next decision). We emphasize that the delayed greedy spanner invariants are different than the greedy spanner invariants for H: in our invariants, the choice of adding (u, v) to the spanner depends on $L[i]$ (the set of edges in L with *scale* strictly smaller than the scale of (u, v)), rather than $H_{<(u,v)}$ (the set of edges in the greedy spanner H that are strictly shorter than $\delta(u, v)$). Nevertheless, we show that any set of edges L that satisfies Invariants 8 and 9 is a $(1 + O(\varepsilon))$-spanner (Lemma 10) and has constant lightness (Lemma 11).

▶ **Lemma 10.** *Let S be an ε-net-tree spanner for X, and let $L \subseteq S$. If Invariant 8 holds, then L is a $(1+3\varepsilon)$-approximate spanner for (X, δ).*

Proof. By Lemma 4, the ε-net-tree spanner S is a $(1+\varepsilon)$-spanner for (X, δ). Moreover, Invariant 8 implies that L is a $(1+\varepsilon)$-spanner for S: by triangle inequality it suffices to check that every edge of S is preserved up to $1+\varepsilon$ approximation in L, and indeed, for every edge $(u, v) \in S$, we have $\delta_L(u, v) \leq \delta^*(u, v) \leq (1+\varepsilon)\delta(u, v) \leq (1+\varepsilon)\delta_S(u, v)$. We conclude that L is a spanner for (X, δ) with stretch $(1+\varepsilon)^2 \leq (1+3\varepsilon)$. ◀

[5] assume all edge lengths in S are distinct, by breaking ties arbitrarily

▶ **Lemma 11.** *Let S be an ε-net-tree spanner for X, and let $L \subseteq S$. If Invariants 8 and 9 hold, then L has lightness $\varepsilon^{-O(d)}$.*

Proof. We use, as a black box, the fact that the greedy $(1+\frac{\varepsilon}{3})$-spanner in a metric space of doubling dimension d has lightness $\varepsilon^{-O(d)}$ [19]. We consider the shortest-path metric δ_L induced by the edges of L; because (X, δ) has doubling dimension d, the stretch bound of Lemma 10 together with the definition of doubling dimension implies that the doubling dimension of (X, δ_L) is $O(d)$ (see [33, Observation 7]). Consider building the greedy $(1+\frac{\varepsilon}{3})$-spanner $\hat{L}$ of L: that is, we initialize $\hat{L} \leftarrow \varnothing$, then iterate over the edges of L from smallest to largest[6], and add an edge $(u,v) \in L$ into the spanner $\hat{L}$ iff $\delta_{\hat{L}}(u,v) > (1+\frac{\varepsilon}{3})\delta_L(u,v)$. The greedy spanner $\hat{L}$ has lightness $\varepsilon^{-O(d)}$. We now *charge* every edge of L to an edge of $\hat{L}$. If an edge $(u,v) \in L$ is added to $\hat{L}$, then charge (u,v) to itself. Otherwise, suppose $(u,v) \in L$ is not added to $\hat{L}$. This means that there is a path P between u and v in $\hat{L}$ (at the instant that (u,v) is considered) with length at most $(1+\frac{\varepsilon}{3})\delta_L(u,v)$. Because S is a geometric graph, $\delta_S(u,v) = \delta(u,v)$; i.e., P has length at most $(1+\frac{\varepsilon}{3})\delta(u,v)$. Let i be the scale of (u,v), meaning that $\delta(u,v) \in [2^{i-1}, 2^i)$. The path P contains some scale-i edge (u',v') in the greedy spanner $\hat{L}$: otherwise, if P contained only scale-j edges with $j < i$, we would have $\delta^*(u,v) \le (1+\frac{\varepsilon}{3}) \cdot \delta(u,v)$, contradicting Invariant 9. Charge (u,v) to the scale-i edge (u',v').

We now claim that every edge (u',v') in $\hat{L}$ is charged at most $\varepsilon^{-O(d)}$ times. Indeed, whenever a scale-i edge (u',v') is charged by a scale-i edge (u,v), we have $\delta(u,u') \le (1+\frac{\varepsilon}{3})2^i$; this is because, by definition of charging, u' lies on a path u and v with length at most $(1+\frac{\varepsilon}{3})\delta(u,v) \le (1+\frac{\varepsilon}{3})2^i$. The packing bound (Observation 3) implies that there are only $\varepsilon^{-O(d)}$ scale-i edges that could charge (u',v'). Because we only charge scale-i edges in L to scale-i edges in $\hat{L}$, we conclude that L is at most $\varepsilon^{-O(d)}$ times heavier than L, which is $\varepsilon^{-O(d)}$ times heavier than the minimum spanning tree on X. ◀

3 Light Spanner with Worst-Case Recourse Bounds

We now describe how to maintain our light spanner L, satisfying Invariants 8 and 9. The current section focuses on describing a spanner with a small *recourse* bound, and proving that our updates maintain Invariants 8 and 9. In Section 4, we explain how to perform these updates quickly. We maintain a data structure *DS*. It stores:

- a set of points *DS*.X
- a net-tree *DS*.$\mathcal{N}$ on the points DS.X
- an ε-net-tree spanner *DS*.S of DS.X with respect to DS.$\mathcal{N}$
- a subgraph *DS*.$L \subseteq$ DS.S called the *light spanner*, such that Invariants 8 and 9 hold.

All these variables are initialized as $\varnothing$. The data structure supports two operations – insertion and deletion of a point x. We say that a procedure Insert(x) *correctly implements insertion* of point x if the procedure assigns DS.$X \leftarrow$ DS.$X \cup \{x\}$ and updates DS.$\mathcal{N}$, DS.S, and DS.L to satisfy the descriptions above. Similarly, a procedure Delete(x) *correctly implement deletion* of a point x if it assigns DS.$X \leftarrow$ DS.$X \setminus \{x\}$ and updates DS.$\mathcal{N}$, DS.S, and DS.L.

3.1 Insertion

To insert a point x, we first update the net-tree according to [35] (as summarized in our Lemma 5) and then recompute the spanner edges DS.L nearby x to satisfy the invariants. The pseudocode is given below. Our key insight is that when we recompute spanner edges at level i, we only need to recompute edges with endpoints in $B(x, O(2^i))$.

[6] by triangle inequality, it suffices to examine only the edges of L when building the greedy spanner, rather than all pairs of vertices in L

```
DS.INSERT(x):
  DS.X ← DS.X ∪ {x}
  update DS.𝒩 and DS.S according to Lemma 5
  DS.RECOMPUTE(x)

DS.RECOMPUTE(x):
  for every scale i ← 0, 1, ..., log Φ:
      for every scale-i edge (u, v) of DS.S with u, v ∈ B(x, 4 · 2^i):
          if δ*(u, v) > (1 + ε) · δ(u, v):
              DS.L ← DS.L ∪ {(u, v)}
          else:
              DS.L ← DS.L \ {(u, v)}
```

▶ **Lemma 12.** *The procedure DS.INSERT(x) correctly implements the insertion of a point x.*

Proof. Let $\mathcal{N}^{\text{old}}$, S^{old}, L^{old}, and X^{old} denote $\texttt{DS}.\mathcal{N}$, $\texttt{DS}.S$, $\texttt{DS}.L$, and $\texttt{DS}.X$ (respectively) before the insertion process. For brevity, we write $\mathcal{N}$, S, L, and X instead of $\texttt{DS}.\mathcal{N}$, $\texttt{DS}.S$, $\texttt{DS}.L$, and $\texttt{DS}.X$. By Lemma 5, the variables $\texttt{DS}.X$, $\texttt{DS}.\mathcal{N}$ and $\texttt{DS}.S$ are correctly maintained. Moreover, we have that $L \subseteq S$, because (by description of the algorithm) $S^{\text{old}} \subseteq S$ and (by assumption) $L^{\text{old}} \subseteq S^{\text{old}}$.

It remains to show that that every edge (u, v) in S satisfies Invariants 8 and 9. Suppose that edge (u, v) is at scale i (that is, $\delta(u, v) \in (2^{i-1}, 2^i]$). Note that the distance $\delta^*(u, v) = \delta_{L[i]}(u, v)$ depends only on the edges in L with scale strictly less than i; thus, it suffices to show that (u, v) satisfies the two invariants at the moment after the for-loop processes scale i. If $u, v \in B(x, 4 \cdot 2^i)$, then (u, v) satisfies the two invariants by description of the algorithm. Now suppose that some endpoint, without loss of generality u, is not in $B(x, 4 \cdot 2^i)$. In this case, (u, v) is in L if and only if (u, v) is in L^{old}. There are two cases:

- **Suppose that (u, v) is not in L.** Thus (u, v) is not in L^{old}. To satisfy Invariant 8, we must show $\delta_{L[i]}(u, v) \leq (1+\varepsilon)\cdot\delta(u, v)$. In fact, we show that either $\delta_{L[i]}(u, v) \leq \delta_{L^{\text{old}}[i]}(u, v)$ or $\delta_{L^{\text{old}}[i]}(u, v) > 2\delta(u, v)$; see Figure 4. This suffices, as L^{old} satisfies Invariant 8: we conclude that $\delta_{L[i]}(u, v) \leq \delta_{L^{\text{old}}[i]}(u, v) \leq (1 + \varepsilon)\delta(u, v)$ as desired. Consider a shortest path P in $L^{\text{old}}[i]$ between u and v. The description of the algorithm implies that the only edges that that change between $L[i]$ and $L^{\text{old}}[i]$ are in the ball $B(x, 4 \cdot 2^{i-1})$. Because we assumed $\delta(u, x) > 4 \cdot 2^i$, triangle inequality implies that endpoints of any edge (u', v') in $L^{\text{old}}[i] \setminus L[i]$ are far from u: specifically, $\delta(u, u') \geq 4 \cdot 2^i - 4 \cdot 2^{i-1} = 2 \cdot 2^i$, and so $\delta_{L^{\text{old}}}(u, u') \geq 2 \cdot 2^i$. If P has length $\geq 2 \cdot 2^i$ then we are done; otherwise, P does not include any edges in $L^{\text{old}}[i] \setminus L[i]$, and so P is also a path in $L[i]$ as desired.
- **Suppose (u, v) is in L.** Thus (u, v) is in L^{old}. To satisfy Invariant 9, we must show that $\delta_{L[i]}(u, v) > (1 + \frac{\varepsilon}{3}) \cdot \delta(u, v)$. We claim that either $\delta_{L[i]}(u, v) \geq \delta_{L^{\text{old}}[i]}(u, v)$, or $\delta_{L[i]}(u, v) \geq 2\delta(u, v)$. This suffices to prove the claim, because L^{old} satisfies Invariant 9 and so $\delta_{L^{\text{old}}[i]}(u, v) > (1 + \frac{\varepsilon}{3})\delta(u, v)$; thus, in either case, $\delta_{L[i]}(u, v) > (1 + \frac{\varepsilon}{3})\delta(u, v)$. Let P be a shortest path between u and v in $L[i]$. If $\delta_{L[i]}(u, v) \geq \delta_{L^{\text{old}}[i]}(u, v)$ then we are done. Otherwise, if $\delta_{L[i]}(u, v) < \delta_{L^{\text{old}}[i]}(u, v)$, then P must contain some edge of $L[i]$ that is not in $L^{\text{old}}[i]$. But the description of the algorithm implies that any such edge has endpoints in $B(x, 4 \cdot 2^{i-1})$. By triangle inequality and our assumption that $\delta(u, x) \geq 4 \cdot 2^i$, we have $\delta(u, u') \geq 2 \cdot 2^i$, and so P has length at least $2 \cdot 2^i$ as desired. ◀

We remark that the INSERT procedure has bounded recourse: at most $\varepsilon^{-O(d)} \log \Phi$ edges change in the spanner L. This is because, for every scale $i \in \lceil \log \Phi \rceil$, there are at most $\varepsilon^{-O(d)}$ scale-i edges in the ε-net-tree spanner in $B(x, 4 \cdot 2^i)$, by definition of net-tree spanner and packing bound (Observation 3).

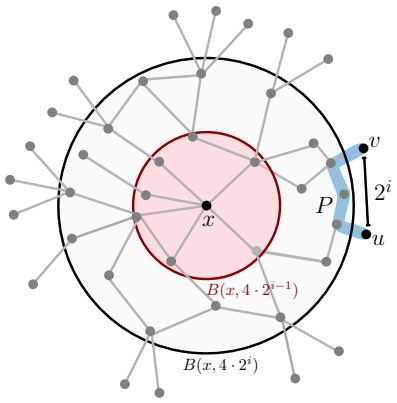


Figure 4 A light spanner L^{old}. For a given scale i, the spanners $L[i]$ and $L^{\text{old}}[i]$ differ only within the ball $B(x, 4 \cdot 2^{i-1})$. A scale-i pair (u, v) outside of $B(x, 4 \cdot 2^i)$. A short path P between u and v in L^{old} does not wander inside $B(x, 4 \cdot 2^{i-1})$, so P is also in L.

3.2 Deletion

The procedure for deleting a point x is similar to insertion. The procedure *DS*.DELETE(x) first updates the net tree according to [35], then deletes all edges in L that were incident to x, and finally recompute the spanner edges nearby x using the DS.RECOMPUTE procedure. The pseudocode and analysis are fairly similar to insertion procedure (albeit slightly more complicated). We defer all details to the full version of the paper.

▶ **Lemma 13.** *The procedure DS.DELETE(x) correctly implements the deletion of a point x.*

We pause to remark that in our proof of Lemma 12 (and also Lemma 13), we have actually proved the following claim, which will be helpful later.

▶ **Lemma 14.** *Consider the data structure DS before and after running DS.INSERT(x) or DS.DELETE(x). Let S^{old} (resp. L^{old}) denote DS.S (resp. DS.L) before the insertion/deletion is performed. Let (u, v) be a scale-i edge in DS.S, with some endpoint outside of $B(x, 4 \cdot 2^i)$. Then (u, v) is an edge in S^{old}, and either (1) $\delta_{DS.L[i]}(u, v) = \delta_{L^{\text{old}}[i]}(u, v)$, or (2) both $\delta_{DS.L[i]}(u, v)$ and $\delta_{L^{\text{old}}[i]}(u, v)$ are larger than $2\delta(u, v)$.*

4 Implementing Updates Quickly

In this section, we show how to modify the data structure so that the DS.INSERT and DS.DELETE procedures can be executed quickly. By prior work on the net-tree spanner (Lemma 5), the net-tree DS.$\mathcal{N}$ and the ε-net-tree spanner DS.S can be maintained in $O(\log \Phi)$ time. The DS.RECOMPUTE procedure iterates over each of the $\log \Phi$ scales. For each scale i, it examines the $\varepsilon^{-O(d)} \log \Phi$ many scale-i edges in $B(x, 4 \cdot 2^i)$; by Claim 7 these edges can be found in $\varepsilon^{-O(d)} = O(1)$ time. Each examined edge (u, v) is added or removed from DS.L depending on $\delta^*(u, v)$. This means that the bottleneck for a fast update time is the computation of $\delta^*(u, v)$:

▶ **Lemma 15.** *The procedure DS.INSERT and DS.DELETE run in $\varepsilon^{-O(d)} \cdot \tilde{O}(\log \Phi)$ time, plus the time needed for $\varepsilon^{-O(d)} \cdot \log \Phi$ many computations of $\delta^*(u, v)$.*

Unfortunately, we don't know how to design a data structure that maintains these distances. To get around this, we first observe that it suffices to maintain an *approximation* for $\delta^*(u,v)$.

▶ **Definition 16.** *We say a value $\tilde{d}$ is an α-approximation for a value d if $d \le \tilde{d} \le \alpha \cdot d$. For any function $d(u,v)$ on pairs of points (we will take $d(u,v) = \delta^*(u,v)$ and $d(u,v) = \delta_{\mathtt{DS}.L}(u,v)$), we say that $\tilde{d}(u,v)$ is a coarse α-approximation for $d(u,v)$ if:*

$$\tilde{d}(u,v) = \begin{cases} \text{if } d(u,v) \le 2\delta(u,v) \text{ then} & d(u,v) \le \tilde{d}(u,v) \le \alpha \cdot d(u,v) \\ \text{if } d(u,v) > 2\delta(u,v) \text{ then} & \tilde{d}(u,v) \ge 2 \cdot \delta(u,v) \end{cases}$$

In other words, a coarse α-approximation $\tilde{d}(u,v)$ for $\delta^*(u,v)$ lets us either detect when $\delta^*(u,v)$ is very large or provides an α-approximation for $\delta^*(u,v)$. Throughout this section, we set $\alpha := 1 + \frac{\varepsilon}{3}$. As we show in Lemma 19, even if we used coarse α-approximations $\tilde{\delta}^*(u,v)$ instead of the real distances $\delta^*(u,v)$ in the $\mathtt{DS}$.RECOMPUTE procedure, our spanner L still satisfies Invariants 8 and 9 after the $\mathtt{DS}$.INSERT and $\mathtt{DS}$.DELETE procedures.

Augmenting the data structure with distance estimates. We now show that we can modify our data structure to efficiently maintain coarse α-approximations of $\delta^*(u,v)$ for every edge (u,v) in the ε-net-tree $\mathtt{DS}.S$. In fact, we maintain something stronger: we need to store several auxiliary values in order to maintain the coarse approximations. Let $\kappa \ge \frac{1024}{3}$ be a sufficiently large constant, and let $\varepsilon_{\text{small}} \leftarrow \frac{\varepsilon}{3 \cdot \kappa \cdot \log \Phi}$. Our data structure $\mathtt{DS}$ maintains (in addition to the points $\mathtt{DS}.X$, the net-tree $\mathtt{DS}.\mathcal{N}$, the ε-net-tree spanner $\mathtt{DS}.S$, and the light spanner $\mathtt{DS}.L$) the following objects:

- An $\varepsilon_{\text{small}}$-net tree spanner $\mathtt{DS}.S_{\text{small}}$ for $\mathtt{DS}.\mathcal{N}$
- For every scale $i \in [\log \Phi]$ and every edge (u,v) in $\mathtt{DS}.S_{\text{small}}$ at scale i, a value $\mathtt{DS}.\tilde{\delta}^*(u,v)$ which is a *coarse* $(1 + \kappa \cdot i \cdot \varepsilon_{\text{small}})$-approximation for $\delta^*(u,v)$.
- For every scale $i \in [(\log \Phi) - 2]$ and every edge (u,v) in $\mathtt{DS}.S_{\text{small}}$ at scale i, a value $\mathtt{DS}.\tilde{\delta}_L(u,v)$ which is a $(1 + \kappa \cdot i \cdot \varepsilon_{\text{small}})$-approximation for $\delta_{\mathtt{DS}.L}(u,v)$.

Observe that this data structure does indeed maintain a coarse $\frac{\varepsilon}{3}$-approximation for every edge in $\mathtt{DS}.S$: this is because $\mathtt{DS}.S \subset \mathtt{DS}.S_{\text{small}}$, and $\kappa \cdot i \cdot \varepsilon_{\text{small}} < \frac{\varepsilon}{3}$ so the estimates $\mathtt{DS}.\tilde{\delta}^*(u,v)$ are coarse $\frac{\varepsilon}{3}$-approximations for $\delta^*(u,v)$. The reason we maintain estimates for every edge in $\mathtt{DS}.S_{\text{small}}$ and not just every edge in $\mathtt{DS}.S$ is to control the distortion, which will accumulate by a $(1 + O(\varepsilon_{\text{small}}))$ factor at each scale.

We highlight two differences between the estimates $\mathtt{DS}.\tilde{\delta}^*(u,v)$ and $\mathtt{DS}.\tilde{\delta}_L(u,v)$, for a scale-$i$ edge (u,v). First, the value $\mathtt{DS}.\tilde{\delta}^*(u,v)$ seeks to estimate $\delta^*(u,v)$, which is the distance between u and v in the subgraph $\mathtt{DS}.L[i]$ which consists of all edges in $\mathtt{DS}.L$ with scale strictly smaller than i. On the other hand, $\mathtt{DS}.\tilde{\delta}_L(u,v)$ estimates distances in the graph $\mathtt{DS}.L$; note that the shortest path between u and v in $\mathtt{DS}.L$ may include edges in $\mathtt{DS}.L$ at scale i and $i+1$, in addition to edges in $\mathtt{DS}.L[i]$. The second key difference is that $\mathtt{DS}.\tilde{\delta}^*(u,v)$ is only required to be a $(1 + \kappa \cdot i \cdot \varepsilon_{\text{small}})$-approximation of $\delta^*(u,v)$ when $\delta^*(u,v) \le 2\delta(u,v)$, whereas $\mathtt{DS}.\tilde{\delta}_L(u,v)$ is always a $(1 + \kappa \cdot i \cdot \varepsilon_{\text{small}})$-approximation of $\delta_{\mathtt{DS}.L}(u,v)$. Why do we need these two types of estimates? Ultimately, our algorithm for INSERT and DELETE will only use the estimates $\mathtt{DS}.\tilde{\delta}^*(u,v)$, to determine whether or not the edge (u,v) should be added. But in order to compute $\mathtt{DS}.\tilde{\delta}^*(u,v)$ for some pair (u,v) at scale i, we will need to estimate $\delta_{\mathtt{DS}.L}(u',v')$ for pairs (u',v') at scale $\le i - 3$.

The fast insert/delete procedures. Below, we give pseudocode for a modified DS.RECOMPUTE procedure called *DS.RECOMPUTEFAST*, which uses the estimates $\mathtt{DS}.\tilde{\delta}^*(u,v)$ instead of the true distances $\delta^*(u,v)$. We define the procedures *DS.INSERTFAST* and *DS.DELETEFAST* to be identical to DS.INSERT and DS.DELETE, except that they use DS.RECOMPUTEFAST instead of DS.RECOMPUTE, and they also maintain a $\varepsilon_{\text{small}}$-net-tree spanner $\mathtt{DS}.S_{\text{small}}$ in addition to the ε-net-tree spanner $\mathtt{DS}.S$.

```
DS.RecomputeFast(x):
  for every scale i ← 0, 1, ..., log φ:
      DS.UpdateDistEstimates(x, i) ⟨⟨recomputes estimates DS.δ̃*(·,·) and DS.δ̃_L(·,·).⟩⟩
      for every scale-i edge (u, v) of DS.S with u, v ∈ B(x, 8 · 2^i):
          d̃ ← DS.δ̃*(u, v) ⟨⟨a ε/3-good estimate for δ*(u, v)⟩⟩
          if d̃ > (1 + ε) · δ(u, v):
              DS.L ← DS.L ∪ {(u, v)}
          else:
              DS.L ← DS.L \ {(u, v)}
```

In Section 4.1, we describe the procedure DS.UPDATEDISTESTIMATES and prove that it correctly maintains estimates for $\delta^*(u,v)$ and $\delta_L(u,v)$. For now, we just state lemmas which summarize the guarantees of DS.UPDATEDISTESTIMATES (Lemmas 17 and 18). Assuming the lemmas hold, we prove the correctness of the fast insert/delete procedures (Lemma 19).

▶ **Lemma 17.** *DS.UPDATEDISTESTIMATES runs in time* $(\varepsilon^{-1} \cdot \log \Phi)^{O(d)}$.

▶ **Lemma 18.** *Consider the execution of DS.INSERTFAST*(x) *or DS.DELETEFAST*(x). *Let* $i \in [\log \Phi]$ *be a scale, and consider the data structure DS at the instant after the execution of the call DS.UPDATEDISTESTIMATES*(x,i) *during the insertion/deletion process. Assume that every edge* (u,v) *of DS.S with scale strictly smaller than* i *satisfies Invariants 8 and 9. Then:*

- *for every scale-*i *edge* (u,v) *in the* $\varepsilon_{\text{small}}$*-net-tree spanner* $\mathtt{DS}.S_{\text{small}}$, *the value* $\mathtt{DS}.\tilde{\delta}^*(u,v)$ *is a coarse* $(1+\kappa \cdot i \cdot \varepsilon_{\text{small}})$*-approximation for* $\delta^*(u,v)$.
- *for every scale-*$(i-2)$ *edge* (u,v) *in the* $\varepsilon_{\text{small}}$*-net-tree spanner* $\mathtt{DS}.S_{\text{small}}$, *the value* $\mathtt{DS}.\tilde{\delta}_L(u,v)$ *is a* $(1+\kappa \cdot i \cdot \varepsilon_{\text{small}})$*-approximation* $\delta_{\mathtt{DS}.L}(u,v)$.

▶ **Lemma 19.** *The procedure DS.INSERTFAST*(x) *(resp. DS.DELETEFAST*(x)*) correctly implements insertion (resp. deletion) of* x. *It runs in time* $(\varepsilon^{-1} \cdot \log \Phi)^{O(d)}$.

Proof. The runtime bound follows from Lemmas 15 and 17. The proof of correctness is almost identical to that of Lemmas 12 and 13 except in one place: we need to argue that every scale-i edge (u,v) in $\mathtt{DS}.S$ with both endpoints in $B(x, 4 \cdot 2^i)$ satisfies Invariants 8 and 9.

For every such edge (u,v), the algorithm checks if $\mathtt{DS}.\tilde{\delta}^*(u,v) > (1+\varepsilon)\delta(u,v)$ and includes (u,v) in L iff the inequality holds. At the instant the algorithm makes the check, we may assume (by induction) that every edge (u',v') in $\mathtt{DS}.S$ with scale strictly smaller than i satisfies Invariants 8 and 9. Thus, by Lemma 18, the value of $\mathtt{DS}.\tilde{\delta}(u,v)$ is a coarse $(1+\kappa \cdot i \cdot \varepsilon_{\text{small}})$-approximation for $\delta^*(u,v)$. As $i \in [\log \Phi]$, in particular we conclude that $\mathtt{DS}.\tilde{\delta}(u,v)$ is a coarse $(1+\frac{\varepsilon}{3})$-approximation for $\delta^*(u,v)$. There are two cases.

Case 1: $\mathtt{DS}.\tilde{\delta}(u,v) \le (1+\varepsilon)\delta(u,v)$. In this case, (u,v) is not in $\mathtt{DS}.L$. We claim that $\delta^*(u,v) \le (1+\varepsilon)\delta(u,v)$, so that (u,v) satisfies Invariant 8. Indeed, we have $\delta^*(u,v) < 2\delta(u,v)$; otherwise, the definition of coarse α-approximation means that $\mathtt{DS}.\tilde{\delta}(u,v) > 2\delta(u,v)$, which contradicts our assumption that $\mathtt{DS}.\tilde{\delta}(u,v) \le (1+\varepsilon)\delta(u,v)$. Thus, the definition of coarse α-approximation implies $\delta^*(u,v) \le \mathtt{DS}.\tilde{\delta}(u,v)$ and so $\delta^*(u,v) \le (1+\varepsilon)\delta(u,v)$ as desired.

Case 2: $\texttt{DS}.\tilde{\delta}(u,v) > (1+\varepsilon)\delta(u,v)$. In this case, (u,v) is added to $\texttt{DS}.L$. We claim that $\delta^*(u,v) > (1+\frac{\varepsilon}{3})\delta(u,v)$, so that (u,v) satisfies Invariant 9. For contradiction, suppose that $\delta^*(u,v) \leq (1+\frac{\varepsilon}{3})\delta(u,v)$. The definition of coarse $(1+\frac{\varepsilon}{3})$-approximation implies $\texttt{DS}.\tilde{\delta}(u,v) \leq (1+\frac{\varepsilon}{3})\cdot\delta^*(u,v)$. Combining these two inequalities, we conclude that $\texttt{DS}.\tilde{\delta}(u,v) \leq \left(1+\frac{\varepsilon}{3}\right)\left(1+\frac{\varepsilon}{3}\right)\delta(u,v) \leq (1+\varepsilon)\delta(u,v)$, contradicting the Case 2 assumption. ◀

Theorem 2 follows from Lemmas 19, 10, and 11 (after rescaling $\varepsilon \leftarrow \varepsilon/3$).

4.1 The procedure UpdateDistEstimates

We defer the details of UpdateDistEstimates to the full version of our paper.

5 Conclusion and Open Questions

We have designed the first dynamic light spanner for point sets in Euclidean and doubling metrics, with update time $(\log \Phi)^{O(d)}$. Our work leaves open two interesting open questions. First, is it possible to maintain a dynamic light spanner with recourse or time bounds *independent* of the aspect ratio Φ and instead depends only on the number of points $n = |X|$? Second, can we obtain a runtime bound where the exponent of $\log \Phi$ (or $\log n$) is independent of d? Ideally, we would like to handle updates in time $\varepsilon^{-O(d)} \cdot \log n$, which would match the best runtime for dynamic sparse spanners.

References

1. Mohammad Ali Abam, Mark de Berg, and Joachim Gudmundsson. A simple and efficient kinetic spanner. In *Proceedings of the twenty-fourth annual symposium on Computational geometry*, pages 306–310, 2008. doi:10.1145/1377676.1377729.
2. Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. A general approach to online network optimization problems. *ACM Trans. Algorithms*, 2(4):640–660, 2006. doi:10.1145/1198513.1198522.
3. Noga Alon and Yossi Azar. On-line steiner trees in the euclidean plane. In David Avis, editor, *Proceedings of the Eighth Annual Symposium on Computational Geometry, Berlin, Germany, June 10-12, 1992*, pages 337–343. ACM, 1992. doi:10.1145/142675.142744.
4. Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993. doi:10.1007/BF02189308.
5. Sunil Arya, Gautam Das, David M Mount, Jeffrey S Salowe, and Michiel Smid. Euclidean spanners: short, thin, and lanky. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 489–498, 1995. doi:10.1145/225058.225191.
6. Sunil Arya, David M Mount, and Michiel Smid. Dynamic algorithms for geometric spanners of small diameter: Randomized solutions. *Computational Geometry*, 13(2):91–107, 1999. doi:10.1016/S0925-7721(99)00014-0.
7. Giorgio Ausiello, Paolo G Franciosa, and Giuseppe F Italiano. Small stretch spanners on dynamic graphs. In Gerth Stølting Brodal and Stefano Leonardi, editors, *13th Annual European Symposium (ESA 2005)*, volume 3669 of *Lecture Notes in Computer Science*, pages 532–543. Springer, Springer, 2005. doi:10.1007/11561071_48.
8. Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Transactions on Algorithms (TALG)*, 8(4):1–51, 2012. doi:10.1145/2344422.2344425.
9. Piotr Berman and Chris Coulston. On-line algorithms for steiner tree problems (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 344–353. ACM, 1997. doi:10.1145/258533.258618.

10 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. *ACM Transactions on Algorithms (TALG)*, 17(4):1–51, 2021. doi:10.1145/3469833.

11 Sujoy Bhore, Arnold Filtser, Hadi Khodabandeh, and Csaba D. Tóth. Online spanners in metric spaces. *SIAM J. Discret. Math.*, 38(1):1030–1056, 2024. doi:10.1137/22m1534572.

12 Sujoy Bhore, Balázs Keszegh, Andrey Kupavskii, Hung Le, Alexandre Louvet, Dömötör Pálvölgyi, and Csaba D. Tóth. Spanners in planar domains via steiner spanners and non-steiner tree covers. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4292–4326. SIAM, 2025. doi:10.1137/1.9781611978322.145.

13 Sujoy Bhore and Lazar Milenkovic. Light spanners with small hop-diameter. In *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, Aarhus, Denmark, July 8-11, 2025*, volume 334 of *LIPIcs*, pages 30:1–30:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ICALP.2025.30.

14 Sujoy Bhore and Csaba D. Tóth. Euclidean steiner spanners: Light and sparse. *SIAM J. Discret. Math.*, 36(3):2411–2444, 2022. doi:10.1137/22m1502707.

15 Sujoy Bhore and Csaba D. Tóth. Online euclidean spanners. *ACM Trans. Algorithms*, 21(1):5:1–5:22, 2025. doi:10.1145/3681790.

16 Greg Bodwin. An alternate proof of near-optimal light spanners. In Merav Parter and Seth Pettie, editors, *2024 Symposium on Simplicity in Algorithms, SOSA 2024, Alexandria, VA, USA, January 8-10, 2024*, pages 39–55. SIAM, 2024. doi:10.1137/1.9781611977936.5.

17 Greg Bodwin and Sebastian Krinninger. Fully dynamic spanners with worst-case update time. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *LIPIcs*, pages 17:1–17:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.17.

18 Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. Minor-free graphs have light spanners. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 767–778. IEEE, 2017. doi:10.1109/FOCS.2017.76.

19 Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. Greedy spanners are optimal in doubling metrics. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2371–2379. SIAM, 2019. doi:10.1137/1.9781611975482.145.

20 Prosenjit Bose, Joachim Gudmundsson, and Pat Morin. Ordered theta graphs. *Computational Geometry*, 28(1):11–18, 2004. doi:10.1016/J.COMGEO.2004.01.003.

21 T-H Hubert Chan and Anupam Gupta. Small hop-diameter sparse spanners for doubling metrics. *Discrete & Computational Geometry*, 41(1):28–44, 2009. doi:10.1007/S00454-008-9115-5.

22 Timothy M Chan, Sariel Har-Peled, and Mitchell Jones. On locality-sensitive orderings and their applications. *SIAM Journal on Computing*, 49(3):583–600, 2020. doi:10.1137/19M1246493.

23 Hsien-Chih Chang, Jonathan Conroy, Hung Le, Shay Solomon, and Cuong Than. Light tree covers, routing, and path-reporting oracles via spanning tree covers in doubling graphs. In *STOC'25—Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 2257–2268. ACM, 2025. doi:10.1145/3717823.3718312.

24 Shiri Chechik and Christian Wulff-Nilsen. Near-optimal light spanners. *ACM Transactions on Algorithms (TALG)*, 14(3):1–15, 2018. doi:10.1145/3199607.

25 Vincent Cohen-Addad, Arnold Filtser, Philip N Klein, and Hung Le. On light spanners, low-treewidth embeddings and efficient traversing in minor-free graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 589–600. IEEE, 2020. doi:10.1109/FOCS46700.2020.00061.

26 Richard Cole and Lee-Ad Gottlieb. Searching dynamic point sets in spaces with bounded doubling dimension. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 574–583, 2006. doi:10.1145/1132516.1132599.

27 Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Transactions on Algorithms (TALG)*, 7(2):1–17, 2011. doi:10.1145/1921659.1921666.

28 Michael Elkin, Ofer Neiman, and Shay Solomon. Light spanners. *SIAM J. Discret. Math.*, 29(3):1312–1321, 2015. doi:10.1137/140979538.

29 Michael Elkin and Shay Solomon. Steiner shallow-light trees are exponentially lighter than spanning ones. *SIAM J. Comput.*, 44(4):996–1025, 2015. doi:10.1137/13094791X.

30 Michael Elkin and Shay Solomon. Fast constructions of lightweight spanners for general graphs. *ACM Trans. Algorithms*, 12(3):29:1–29:21, 2016. doi:10.1145/2836167.

31 David Eppstein and Hadi Khodabandeh. Maintaining light spanners via minimal updates. In *Proceedings of the 36th Canadian Conference on Computational Geometry (CCCG)*, pages 1–15, 2024.

32 Arnold Filtser and Ofer Neiman. Light spanners for high dimensional norms via stochastic decompositions. *Algorithmica*, 84(10):2987–3007, 2022. doi:10.1007/s00453-022-00994-0.

33 Arnold Filtser and Shay Solomon. The greedy spanner is existentially optimal. *SIAM J. Comput.*, 49(2):429–447, 2020. doi:10.1137/18M1210678.

34 Sebastian Forster and Gramoz Goranci. Dynamic low-stretch trees via dynamic low-diameter decompositions. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 377–388, 2019. doi:10.1145/3313276.3316381.

35 Jie Gao, Leonidas J. Guibas, and An Thai Nguyen. Deformable spanners and applications. *Comput. Geom.*, 35(1-2):2–19, 2006. doi:10.1016/j.comgeo.2005.10.001.

36 Lee-Ad Gottlieb. A light metric spanner. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 759–772. IEEE, 2015. doi:10.1109/FOCS.2015.52.

37 Lee-Ad Gottlieb and Liam Roditty. Improved algorithms for fully dynamic geometric spanners and geometric routing. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 591–600. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347148.

38 Lee-Ad Gottlieb and Liam Roditty. An optimal dynamic spanner for doubling metric spaces. In *European Symposium on Algorithms*, pages 478–489. Springer, 2008. doi:10.1007/978-3-540-87744-8_40.

39 Michelangelo Grigni and Papa Sissokho. Light spanners and approximate tsp in weighted graphs with forbidden minors. In *SODA*, volume 2, pages 852–857, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545492.

40 MohammadTaghi Hajiaghayi, Vahid Liaghat, and Debmalya Panigrahi. Online node-weighted steiner forest and extensions via disk paintings. *SIAM J. Comput.*, 46(3):911–935, 2017. doi:10.1137/14098692X.

41 Sariel Har-Peled, Piotr Indyk, and Anastasios Sidiropoulos. Euclidean spanners in high dimensions. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 804–809. SIAM, 2013. doi:10.1137/1.9781611973105.57.

42 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006. doi:10.1137/S0097539704446281.

43 Robert Krauthgamer and James R. Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 798–807. ACM, New York, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982913.

44 Hung Le and Shay Solomon. A unified framework for light spanners. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 295–308. ACM, 2023. doi:10.1145/3564246.3585185.

45 Hung Le and Shay Solomon. Truly optimal euclidean spanners. *SIAM J. Comput.*, 54(4):S19–135, 2025. doi:10.1137/20m1317906.

46 Joseph Naor, Debmalya Panigrahi, and Mohit Singh. Online node-weighted steiner tree and related problems. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, (FOCS)*, pages 210–219. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.65.

47 Giri Narasimhan and Michiel Smid. *Geometric spanner networks*. Cambridge University Press, Cambridge, 2007. doi:10.1017/CBO9780511546884.

48 David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989. doi:10.1002/JGT.3190130114.

49 Satish Rao and Warren D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 540–550. ACM, 1998. doi:10.1145/276698.276868.

50 Liam Roditty. Fully dynamic geometric spanners. *Algorithmica*, 62(3):1073–1087, 2012. doi:10.1007/S00453-011-9504-7.

51 Michiel HM Smid. The weak gap property in metric spaces of bounded doubling dimension. *Efficient Algorithms*, 5760:275–289, 2009. doi:10.1007/978-3-642-03456-5_19.