Fast Construction of Nets in Low-Dimensional Metrics and Their Applications

Sariel Har-Peled* Manor Mendel†

October 21, 2014

Abstract

We present a near linear time algorithm for constructing hierarchical nets in finite metric spaces with constant doubling dimension. This data-structure is then applied to obtain improved algorithms for the following problems: approximate nearest neighbor search, well-separated pair decomposition, spanner construction, compact representation scheme, doubling measure, and computation of the (approximate) Lipschitz constant of a function. In all cases, the running (preprocessing) time is near linear and the space being used is linear.

1. Introduction

Given a data set, one frequently wants to manipulate it and quickly compute some properties of it. For example, one would like to cluster the data into similar clusters, or measure similarity of items in the data, etc. One possible way to do this is to define a distance function (i.e., metric) on the data items and perform the required task using this metric. Unfortunately, in general, the metric might be intrinsically complicated ("high-dimensional"), and various computational tasks on the data might require high time and space complexity. This is known in the literature as "the curse of dimensionality."

One approach receiving considerable attention recently is that of defining a notion of dimension on a finite metric space and developing efficient algorithms for this case. An example of this approach is the notion of doubling dimension [Ass83, Hei01, GKL03]. The *doubling constant* of metric space $\mathcal{M}$ is the maximum, over all balls $\mathbf{b}$ in the metric space $\mathcal{M}$, of the minimum number of balls needed to cover $\mathbf{b}$, using balls with half the radius of $\mathbf{b}$. The logarithm of the doubling constant is the *doubling dimension* of the space. The doubling dimension can be thought of as a generalization of the Euclidean dimension, as $\Re^d$ has $\Theta(d)$ doubling dimension. Furthermore, the doubling dimension extends the notion of growth restricted metrics of Karger and Ruhl [KR02].

Understanding the structure of such spaces (or similar notions) and how to manipulate them efficiently received considerable attention in the last few years [Cla99, KR02, GKL03, HKMR04, KL04b, KL04a, Tal04].

The low doubling metric approach can be justified at the following two levels:

*Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801 (sariel@uiuc.edu, http://www.uiuc.edu/~sariel). The work of this author was partially supported by NSF CAREER award CCR-0132901.

†Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801. Current affiliation: The Open University of Israel, Raanana, Israel (mendelma@gmail.com).

1. Arguably, non-Euclidean, low (doubling) dimensional metric data appear in practice and deserve an efficient algorithmic treatment. Even high-dimensional Euclidean data may have some low doubling dimension structure, which makes it amenable to this approach.

   This view seems to be shared by many recent algorithmic papers on doubling metrics, but it still awaits convincing empirical and/or theoretical support.

2. Even if one is only interested in questions on Euclidean point sets, it makes sense to strip the techniques being used to their bare essentials, obtaining a better understanding of the problems and conceptually simpler solutions.

More arguments along these lines can be found in [Cla99], where the author advocates this approach.

In general, it is impossible to directly apply algorithmic results developed for fixed dimensional Euclidean space to doubling metrics, since there exist doubling metrics that cannot be embedded in Hilbert space with low distortion of the distances [Sem96, Laa02]. Hence, some of the aforementioned works apply notions and techniques from fixed dimensional computational geometry and extend them to finite metric spaces.

In particular, Talwar [Tal04] showed that one can extend the notion of *well-separated pairs decomposition* (WSPD) of [CK95] to spaces with low doubling dimension. Specifically, he shows that for every set P of n points having doubling dimension dim, and every $\varepsilon > 0$, there exists WSPD, with separation $1/\varepsilon$ and $O(n\varepsilon^{-O(\dim)} \log \Phi)$ pairs, where dim is the doubling dimension of the finite metric space, and Φ is the *spread* of the point set, which is the ratio between the diameter of P and the distance between the closest pair of points in P. This is weaker than the result of Callahan and Kosaraju [CK95] for Euclidean space, which does not depend on the spread of the point set.

Krauthgamer and Lee [KL04b] showed a data structure for answering $(1+\varepsilon)$-approximate nearest neighbor queries on point set P with spread Φ. Their data structure supports insertions in $O(\log \Phi \log\log \Phi)$ time. The preprocessing time is $O(n \log \Phi \log\log \Phi)$ (this is by inserting the points one by one), and the query time is $O(\log \Phi + \varepsilon^{-O(\dim)})$. In $\Re^d$ for fixed d, one can answer such queries in $O(\log\log(\Phi/\varepsilon))$ time, using near linear space; see [Har01] and references therein. (In fact, it is possible to achieve constant query time using slightly larger storage [HM04].) Note, however, that the latter results strongly use the Euclidean structure. Recently, Krauthgamer and Lee [KL04a] overcame the restriction on the spread, presenting a data-structure with nearly quadratic space, and logarithmic query time.

Underlining all those results is the notion of hierarchical nets. Intuitively, hierarchical nets are sequences of larger and larger subsets of the underlining set P, such that in a given resolution, there is a subset in this sequence that represents well the structure of P in this resolution (a formal definition is given in Section 2). Currently, the known algorithms for constructing those nets require running time which is quadratic in n.

An alternative way of constructing those nets is by the clustering algorithm of Gonzalez [Gon85]. The algorithm of Gonzalez computes 2-approximate k-center clustering by repeatedly picking the point furthest away from the current set of centers. Setting $k = n$, this results in a permutation of the points in the metric space. It is easy to verify that, by taking different prefixes of this permutation, one gets hierarchical nets for the metric. However, the running time of Gonzalez's algorithm in this case is still quadratic. Although in fixed dimensional Euclidean space the algorithm of Gonzalez was improved to $O(n \log k)$ time by Feder and Greene [FG88], and to linear time by Har-Peled [Har04], those algorithms require specifying k in advance, they do not generate the permutation of the points, and as such they cannot be used in this case.

Our results. In this paper, we present for the aforementioned applications improved algorithms having near linear preprocessing time and linear space. We also remove the dependency on the spread.

As such, we (almost) match the known results in computational geometry for low-dimensional Euclidean spaces.

We assume that the input is given via a black box that can compute the distance between any two points in the metric space in constant time. Since the matrix of all $\binom{n}{2}$ distances has quadratic size, this means that in some sense our algorithms have sublinear running time. This is not entirely surprising since subquadratic time algorithms exist for those problems in fixed dimensional Euclidean space. Thus, our paper can be interpreted as further strengthening the perceived connection between finite spaces of low doubling dimensions and Euclidean space of low dimension. Furthermore, we believe that our algorithms for the well-separated pair decomposition and approximate nearest neighbor are slightly cleaner and simpler than the previous corresponding algorithms for the Euclidean case.

Net-tree. In Section 3 we present a $2^{O(\dim)} n \log n$ expected time randomized algorithm for constructing the hierarchical nets data-structure, which we call a net-tree.

Approximate nearest neighbor (ANN). In Section 4 we show a new data-structure for the $(1+\varepsilon)$-approximate nearest neighbor query. The expected preprocessing time is $2^{O(\dim)} n \log n$, the space used is $2^{O(\dim)} n$, and the query time is $2^{O(\dim)} \log n + \varepsilon^{-O(\dim)}$.

This query time is almost optimal in the oracle model since there are examples of point sets in which the query time is $2^{\Omega(\dim)} \log n$ [KL04b], and examples in which the query time is $\varepsilon^{-\Omega(\dim)}$.[1]

Our result also matches the results of Arya et al. [AMN+98] in Euclidean settings. Furthermore, our result improves upon the recent work of Krauthgamer and Lee, which either assumes bounded spread [KL04b] or requires quadratic space [KL04a]. The algorithms in [AMN+98, KL04b, KL04a] are deterministic, in contrast to ours.

WSPD. In Section 5 we show that one can construct an ε^{-1} WSPD of P in near linear time. The number of pairs is $n\varepsilon^{-O(\dim)}$. The size of the WSPD is tight, as there are examples of metrics in which the size of the WSPD is $n\varepsilon^{-\Omega(\dim)}$. Our result improves upon Talwar's work [Tal04] and matches the results of Callahan and Kosaraju (the algorithms of both [CK95, Tal04] are deterministic, though).

Spanners. A t-spanner of a metric is a sparse weighted graph whose vertices are the metric's points and in which the graph metric is a t-approximation to the original metric. Spanners were first defined and studied in [PS89]. Construction of $(1+\varepsilon)$-spanners for points in low-dimensional Euclidean space is considered in [Kei88, Cal95]. Using Callahan's technique [Cal95], the WSPD construction also implies a near linear time construction of $(1+\varepsilon)$-spanners having linear number of edges for such metrics. Independently of our work, Chan et al. [CGMZ05] show a construction of a $(1+\varepsilon)$-spanner for doubling metrics with a linear number of edges. Their construction is stronger in the sense that the degrees in their spanner graph are bounded by constant. However, they do not specify a bound on the running time of their construction.

Compact representation scheme (CRS). In Section 6 we construct in near linear time a data-structure of linear size that can answer approximate distance queries between pairs of points in essentially constant time. CRSs were coined "approximate distance oracles" in [TZ01]. Our result extends recent results of Gudmunsson et al. [GLNS02a, GLNS02b], who showed the existence of CRSs with similar parameters for metrics that are "nearly" fixed dimensional Euclidean (and that are a subclass of fixed doubling dimension metrics). We also mention in passing that our CRS technique can be applied to improve and unify two recent results [Tal04, Sli05b] on distance labeling.

Doubling measure. A doubling measure μ is a measure on the metric space with the property that for every $x \in P$ and $r > 0$, the ratio $\mu(\mathbf{b}(x,2r))/\mu(\mathbf{b}(x,r))$ is bounded, where $\mathbf{b}(x,r) = \{y : d(x,y) \le r\}$.

[1] Consider the set $\mathbb{Z}_m^n$ with the ℓ_∞ norm, where $m = \lceil \varepsilon^{-1}/2 \rceil$, $n = \lceil \dim \rceil$. Consider a query at point q satisfying $\exists x_0 \in \mathbb{Z}_m^n$ such that $d(q,x_0) = m-1$, and for all $x \in \mathbb{Z}_m^n$, $x \neq x_0 \Rightarrow d(q,x) = m$. Since x_0 can be chosen in an adversarial way, any randomized $(1+\varepsilon)$-ANN query algorithm would have to make $\Omega(m^n)$ distance queries before hitting x_0.

Vol′berg and Konyagin [VK87] proved that for finite metrics (and in fact for complete metrics [LS98]) the existence of a doubling measure is quantitatively equivalent to the metric being doubling. This measure has found some recent algorithmic applications [Sli05b], and we anticipate more applications. Following the proof of Wu [Wu98], we present in Section 7 a near linear time algorithm for constructing a doubling measure.

Lipschitz constant of a mapping. In Section 8 we study the problem of computing the Lipschitz constant of a mapping $f : P \to B$. In particular, we show how using WSPD makes it possible to approximate the Lipschitz constant of f in near linear time (in $|P|$) when P has a constant doubling dimension (and B is an arbitrary metric). We also obtain efficient exact algorithms, with near linear running time, for the case where P is a set of points in one- or two-dimensional Euclidean space.

Computing the doubling dimension. Although not stated explicitly in what follows, we assume in sections 2–8 that the doubling dimension of the given metric is either known a priori or given as part of the input. This assumption is removed in Section 9, where we remark that a constant approximation of the doubling dimension of a given metric $\mathcal{M}$ can be computed in $2^{O(\dim)} n \log n$ time. It is therefore possible to execute the algorithms of sections 2–8 with the same asymptotic running time, using the approximation of the doubling dimension. (In all the cases where the doubling dimension is needed, any upper bound on it will do, with accordingly degraded running time.)

Most of the algorithms in this paper are randomized. However, our use of randomness is confined to Lemma 2.4 (except for Section 8.1). This means that the algorithms always return the desired result, with bounds on the *expected* running time. This also gives the same asymptotic bound with constant probability, using Markov inequality. Furthermore, in the ANN and CRS schema, randomness is used only in the preprocessing, and the query algorithms are deterministic. Lemma 2.4 can be easily derandomized in $O(n^2)$ time, thus giving $n^2\text{polylog}(n)$ deterministic algorithms for all problems discussed here. We do not know whether a nontrivial derandomization is possible.

2. Preliminaries

Denote by $\mathcal{M}$ a metric space and by P a finite subset $P \subset \mathcal{M}$. The *spread* of P, denoted by $\Phi(P)$, is the ratio of the diameter of P and the distance between the closest pair of points in P. For a point $p \in \mathcal{M}$ and a number $r \geq 0$, we denote by $\mathbf{b}(p, r) = \{q \in \mathcal{M} |\ d_{\mathcal{M}}(p, q) \leq r\}$ the ball of radius r around p. The *doubling constant* λ of P, defined as the minimum over $m \in \mathbb{N}$ such that every ball $\mathbf{b}$ in P, can be covered by at most m balls of at most half the radius. The doubling dimension of the metric space is defined as $\log_2 \lambda$. A slight variation of the doubling constant is that any subset can be covered by λ' subsets of at most half the diameter. It is not hard to see that $\log_2 \lambda$ and $\log_2 \lambda'$ approximate each other up to a factor of 2. Since we will ignore constant factors in the dimension, these two definitions are interchangeable. It is clear that $\log_2 \lambda'(P) \leq \log_2 \lambda'(\mathcal{M})$, and thus the doubling dimension of P is "approximately" at most that of $\mathcal{M}$.

A basic fact about the λ doubling metric $\mathcal{M}$ that will be used repeatedly is that if $P \subset \mathcal{M}$ has spread at most Φ, then $|P| \leq \lambda^{O(\log_2 \Phi)}$.

2.1. Hierarchy of nets

An r-net in a metric space $\mathcal{M}$ is a subset $\mathcal{N} \subset \mathcal{M}$ of points such that $\sup_{x \in \mathcal{M}} d_{\mathcal{M}}(x, \mathcal{N}) \leq r$ and $\inf_{x,y \in \mathcal{N};\ x \neq y} d_{\mathcal{M}}(x, y) \geq r/\alpha$ for some constant $\alpha \geq 1$. r-nets are useful "sparse" object that approximately capture the geometry of the metric space at scales larger than $3r$. In this paper we will rely heavily on the following notion of hierarchical nets.

Definition 2.1 (net-tree). Let $P \subset \mathcal{M}$ be a finite subset. A net-tree of P is a tree T whose set of leaves is P. We denote by $P_v \subset P$ the set of leaves in the subtree rooted at a vertex $v \in T$. Associate with each vertex v a point $\mathrm{rep}_v \in P_v$. Internal vertices have at least two children. Each vertex v has a level $\ell(v) \in \mathbb{Z} \cup \{-\infty\}$. The levels satisfy $\ell(v) < \ell(\overline{\mathrm{p}}(v))$, where $\overline{\mathrm{p}}(v)$ is the parent of v in T. The levels of the leaves are $-\infty$. Let τ be some large enough constant, say $\tau = 11$.

We require the following properties from T:

Covering property: For every vertex $v \in T$,

$$\mathbf{b}\left(\mathrm{rep}_v, \frac{2\tau}{\tau - 1} \cdot \tau^{\ell(v)}\right) \supset P_v.$$

Packing property: For every nonroot vertex $v \in T$,

$$\mathbf{b}\left(\mathrm{rep}_v, \frac{\tau - 5}{2(\tau - 1)} \cdot \tau^{\ell(\overline{\mathrm{p}}(v))-1}\right) \bigcap P \subset P_v.$$

Inheritance property: For every nonleaf vertex $u \in T$, there exists a child $v \in T$ of u such that $\mathrm{rep}_u = \mathrm{rep}_v$.

The net-tree can be thought of as a representation of nets from all scales in the following sense.

Proposition 2.2. *Given a net-tree, let*

$$\mathcal{N}_C(l) = \left\{ \mathrm{rep}_u \;\middle|\; \ell(u) < l \leq \ell(\overline{\mathrm{p}}(u)) \right\}.$$

Then the points in $\mathcal{N}_C(l)$ are pairwise $\tau^{l-1}/4$ separated; that is, for any $p, q \in \mathcal{N}_C(l)$, we have $d_{\mathcal{M}}(p, q) \geq \tau^{l-1}/4$. In addition, $P \subseteq \cup_{p \in \mathcal{N}_C(l)} \mathbf{b}(p, 4 \cdot \tau^l)$.

Proof: Let $p, q \in \mathcal{N}_C(l)$, and let u and v be the corresponding nodes in the net-tree, respectively. Consider the balls $\mathbf{b}_p = \mathbf{b}(p, r_p)$ and $\mathbf{b}_q = \mathbf{b}(q, r_q)$, where $r_p = \frac{\tau-5}{2(\tau-1)} \cdot \tau^{\ell(\overline{\mathrm{p}}(u))-1}$ and $r_q = \frac{\tau-5}{2(\tau-1)} \cdot \tau^{\ell(\overline{\mathrm{p}}(v))-1}$. The sets $\mathbf{b}_p \cap P$ and $\mathbf{b}_q \cap P$ are fully contained in P_u and P_v, respectively, by the definition of the net-tree. Since u and v are on different branches of the net-tree, P_u and P_v are disjoint. But then $d_{\mathcal{M}}(p, q) \geq \max\{r_p, r_q\} \geq \frac{\tau-5}{2(\tau-1)} \cdot \tau^{l-1} \geq \tau^{l-1}/4$ by the definition of $\mathcal{N}_C(l)$ and since $\tau = 11$.

Similarly, consider the set of nodes $V_C(l) = \left\{ u \middle| \ell(u) < l \leq \ell(\overline{\mathrm{p}}(u)) \right\}$ realizing $\mathcal{N}_C(l)$. For any $v \in V_C(l)$, we have $P_v \subseteq \mathbf{b}\left(\mathrm{rep}_v, \frac{2\tau}{\tau-1} \cdot \tau^{\ell(v)}\right) \subseteq \mathbf{b}\left(\mathrm{rep}_v, \frac{2\tau^l}{\tau-1}\right) \subseteq \mathbf{b}\left(\mathrm{rep}_v, \tau^l\right)$ since $\tau \geq 3$. Thus, $P \subseteq \cup_{v \in V_C(l)} \mathbf{b}(\mathrm{rep}_v, \tau^l) = \cup_{p \in \mathcal{N}_C(l)} \mathbf{b}(p, \tau^l)$, as required. ∎

Although $\mathcal{N}_C(\cdot)$ are quantitatively weaker nets compared with the greedy approach,[2] they are stronger in the sense that the packing and the covering properties respect the hierarchical structure of the net-tree.

The packing and covering properties easily imply that each vertex has at most $\lambda^{O(1)}$ children. Net-trees are roughly equivalent to compressed quadtrees [AMN+98]. The net-tree is also similar to the sb data-structure of Clarkson [Cla02], but our analysis and guaranteed performance are new.

[2] We have made no attempt to optimize the ratio between the packing and covering radii, and the one reported here can be (substantially) improved. However, some degradation in this ratio seems to be unavoidable.

2.2. The computational model

The model of computation we use is the "unit cost floating-point word RAM model." More precisely, for a given input consisting of poly(n) real numbers at the range $[-\Phi, -\Phi^{-1}] \cup [\Phi^{-1}, \Phi]$, and given an accuracy parameter $t \in \mathbb{N}$, the RAM machine has words of length $O(\log n + \log\log \Phi + t)$. These words can accommodate floating-point numbers from the set

$$\left\{ \pm(1+x)2^y \;\middle|\; x \in [0,1], x2^{-t} \in \mathbb{N}, y \in [-n^{O(1)} \log^{O(1)} \Phi, n^{O(1)} \log^{O(1)} \Phi] \cap \mathbb{Z} \right\}$$

and integers from the set $\left\{-(2^t n \log \Phi)^{O(1)}, \ldots, 0, \ldots, (2^t n \log \Phi)^{O(1)}\right\}$. For simplicity, we assume that the input given in this way is *exact*. All the problems discussed in this paper have an accuracy parameter of $\varepsilon > 0$. We assume that $\varepsilon^{O(1)} > 2^{-t}$ to avoid rounding problems. The space used by an algorithm (or a scheme) is the number of *words* being used. The machine allows arithmetic, floor, ceiling, conversion from integer to floating point, logarithm, and exponent operations in unit time. We further assume that the machine is equipped with a random number generator.

Floating-point computation is a very well studied topic; see [Knu97, Chap. 4] and references therein. However, we were unable to trace a citation that explicitly defines an asymptotic floating-point computational model. We choose this model for the following two related reasons:

1. The algorithms in this paper are supposed to output only an approximate solution. Therefore it makes sense to try to use approximate numbers since they use less resources.

2. An important theme in this paper is developing algorithms that are independent of the spread of the given metrics. Most algorithms that have an explicit dependence on the spread in their time or space complexity have some form of polylog(Φ) dependence. An algorithm that has no dependence on the spread Φ, but relies on words of length $O(\log \Phi)$, may be considered suspicious at best.

Having stated these reasons, for the most part in what follows we will ignore numerical and accuracy issues in our algorithms. The algorithms are simple enough that it is evidently clear that no numerical stability issues arise. A notable exception is Assouad's embedding discussed in Section 6.2. There we have to explicitly add another ingredient (Lemma 6.9) to the algorithm in order to adapt it to the floating-point word RAM model. Indeed, that section is the catalyst for the current discussion.

2.3. Finding a separating ring

We next present a simple argument that helps to overcome the dependence on the spread in the running time.

Proposition 2.3. *Denote by $r_{\mathrm{opt}}(P, m)$ the radius of the smallest ball in P (whose center is also in P) containing m points. Then in a metric space with doubling constant λ, any ball of radius $2r$, where $r \le 2r_{\mathrm{opt}}(P, m)$, contains at most $\lambda^2 m$ points.*

Proof: By the doubling property, the ball of radius $2r$ can be covered by λ^2 balls of radius $r_{\mathrm{opt}}(P, m)$. Each such ball contains at most m points. ∎

Lemma 2.4. *Given an n-point metric space P with doubling constant λ, one can compute a ball $\mathbf{b} = \mathbf{b}(p, r)$, such that $\mathbf{b}$ contains at least $m = n/(2\lambda^3)$ points of P, and $\mathbf{b}(p, 2r)$ contains at most $n/2$ points of P. The expected running time of this algorithm is $O(\lambda^3 n)$.*

Proof: Pick randomly a point p from P, and compute the ball $\mathbf{b}(p, r)$ of smallest radius around p containing at least $n/(2\lambda^3)$ points. Next, consider the ball of radius $\mathbf{b}(p, 2r)$. If it contains $\leq n/2$ points, we are done. Otherwise, we repeat this procedure until we succeed.

To see why this algorithm succeeds with constant probability in each iteration, consider the smallest ball $Q = P \cap \mathbf{b}(q, r_{\text{opt}})$ that contains at least m points of P. Observe that any ball of radius $r_{\text{opt}}/2$ contain less than m points. With probability $1/(2\lambda^3)$ our sample is from Q. If $p \in Q$, then $r \leq 2r_{\text{opt}}$, and by the doubling property the ball $\mathbf{b}(p, 4r_{\text{opt}})$ can be covered by at most λ^3 balls of radius $r_{\text{opt}}/2$. Hence it holds that $|P \cap \mathbf{b}(p, 2r)| < \lambda^3 m \leq n/2$.

Thus, the algorithm succeeds with probability $1/(2\lambda^3)$ in each iteration, and with probability $\geq 1/3$ after $2\lambda^3$ iterations, implying the result, as each iteration takes $O(n)$ time. ∎

Lemma 2.4 enables us to find a sparse ring of radius "not much larger" than its width. For example, by using it we can find an empty ring of width h and radius at most $2nh$ in linear time.

3. Computing nets efficiently

In this section we prove the following theorem.

Theorem 3.1. *Given a set P of n points in $\mathcal{M}$, one can construct a net-tree for P in $2^{O(\dim)} n \log n$ expected time.*

The outline of the proof is as follows. In Section 3.1 we show how to construct the Gonzalez sequence in $2^{O(\dim)} n \log(n + \Phi)$ time. We then eliminate the dependence of the running time on the spread Φ in Section 3.3 by using a tool developed in Section 3.2. In Section 3.4 we conclude the proof of Theorem 3.1 by showing how to construct the net-tree from the Gonzalez sequence. We end by mentioning in Section 3.5 a few data structures for efficient searching on the net-tree.

3.1. Computing greedy clustering quickly

Gonzalez [Gon85] presented a greedy algorithm, denoted by `GreedyCluster`, that when applied to a set of points P computes a permutation of the points $\Pi = \langle p_1, p_2, \ldots, p_m \rangle$, such that $p_1, \ldots, p_k$ are good centers for P, for any $k \geq 1$. We refer to Π as the *greedy permutation* of P. Formally, there are numbers $r_1, \ldots, r_n$, such that $P \subseteq \cup_{l=1}^{k} \mathbf{b}(p_l, r_k)$. Furthermore, $\min_{1 \leq i < j \leq k} d_{\mathcal{M}}(p_i, p_j) = r_{k-1}$.

`GreedyCluster` works by picking an arbitrary point in P to be p_1 and setting r_1 to be the distance of the furthest point in P to p_1. For every point $q \in P$, `GreedyCluster` stores its distance to the closest center picked so far; namely, in the beginning of the kth iteration, for all $q \in P$ we have $\alpha_q^k = \min_{i=1}^{k-1} d_{\mathcal{M}}(q, p_i)$. The algorithm sets the kth center to be $p_k = \arg\max_{p \in P} \alpha_p^k$ (namely, p_k is the point in P furthest away from the centers picked so far). Clearly, $r_{k-1} = \alpha_{p_k}^k$. By implementing this naively, one can compute the first k points $p_1, \ldots, p_k$ in $O(nk)$ time. Thus, this leads to a 2-approximation to k-center clustering in $O(nk)$ time.

Feder and Greene [FG88] improved the running time to $O(n \log k)$ time (this was further improved to linear time by Har-Peled [Har04]). Feder and Greene's main observation was that when updating α_q^{k+1}, one needs to update this value only for points of P, which are in distance $\leq r_{k-1}$ away from p_k, since for points q further away, the addition of p_k cannot change α_q^k.

This suggests the following natural approach for computing the greedy permutation: Associate with each center in $\{p_1, \ldots, p_k\}$ the points of P that it serves (namely, points that are closer to the given center than to any other center). Furthermore, each center p_i maintains a *friends list* that contains all

the centers that are a distance of at most $4r_k$ from it. An "old" center will trim a point from its friends list only when its distance is larger than $8r_k$. Specifically, the friends list of p_i at the kth iteration ($k \geq i$) contains all the centers at a distance of at most $\min\{8r_k, 4r_i\}$ from p_i. Because of the constant doubling dimension property, this list is of size $\lambda^{O(1)}$.

We further maintain a max-heap in which every center p_i, $i < k$, maintains the point p'_i furthest away from p_i in its cluster along with its current $\alpha_{p'_i} = d_{\mathcal{M}}(p_i, p'_i)$ value.

At the kth iteration, the algorithm extracts the maximum value from the heap. It sets p_k to be the corresponding point. Denote by c_{p_k} the closest point among $\{p_1, \ldots, p_{k-1}\}$ to p_k (i.e., the cluster's center of p_k at the end of the $(k-1)$th round). Next, the algorithm scans all the points currently served by the same cluster as c_{p_k}, or by clusters containing points from friends list of c_{p_k}, and updates the α value of those points. Furthermore, it moves all the relevant points to the newly formed cluster. In the process, it also update the points p'_i (of maximum distance from p_i in its cluster) for all p_i in the friends list of c_{p_k}. It also computes the friends list of p_k (how exactly this is done will be described in detail shortly).

We next bound the running time. To this end, a phase starting at the ith iteration of the algorithm terminates at the first $j > i$ such that $r_{j-1} \leq r_{i-1}/2$. A ball of radius $4r_{j-1}$ around each point $q \in P$ contains at most λ^3 points of $p_1, \ldots, p_j$, and as such every point of P is being scanned at most λ^3 times at each phase of the algorithm. Thus, if the spread of the point set is Φ, the number of phases is $O(\log \Phi)$, and scanning takes $\lambda^{O(1)} n \log \Phi$ time overall. Maintaining the max-heap costs an additional $\lambda^{O(1)} n \log n$ time, since in each iteration only $\lambda^{O(1)}$ values in the head are changed.

The only remaining hurdle is the computation of the friends list of a newly formed center p_k. This can be done by maintaining, for every point p_l, $l \in \{1, \ldots, n\}$, the serving center $p_{l'}$ two phases ago (at the end of that phase). The friends list of p_k is constructed by scanning the friends list of $p_{k'}$ and picking those points that are at distance at most $4r_k$ from p_k. This costs $\lambda^{O(1)}$ time for p_k and $O(\lambda^{O(1)} n)$ time overall. To see that this search suffices, note that the set $\{p_i | i < k,\ d_{\mathcal{M}}(p_i, p_k) \leq 4r_k\}$ is scanned. Indeed, fix p_{i_0}, having $i_0 < k$, and $d_{\mathcal{M}}(p_{i_0}, p_k) \leq 4r_k$. Let $p_{k'}$ be the center of p_k two phases ago. From the definition, $2r_k \leq r_{k'} \leq 4r_k$, and thus $d_{\mathcal{M}}(p_k, p_{k'}) \leq 4r_k$. The current (at the end of the $(k-1)$th iteration) friends list of $p_{k'}$ contains all the current centers at a distance of at most $\min\{8r_k, 4r_{k'}\} = 8r_k$ from $p_{k'}$. Furthermore,

$$d_{\mathcal{M}}(p_{i_0}, p_{k'}) \leq d_{\mathcal{M}}(p_{i_0}, p_k) + d_{\mathcal{M}}(p_k, p_{k'}) \leq 8r_k.$$

We are therefore guaranteed that p_{i_0} will be scanned.

Of course, as the algorithm progresses it needs to remove nonrelevant elements from the friends list as the current clustering radius r_i shrinks. However, this can be done in a lazy fashion whenever the algorithm scans such a list.

Theorem 3.2. *Let P be an n-point metric space with doubling constant λ and spread Φ. Then the greedy permutation for P can be computed in $O(\lambda^{O(1)} n \log(\Phi n))$ time and $O(\lambda^{O(1)} n)$ space.*

3.2. Low quality approximation by HST

Here we present an auxiliary tool that will be used in Section 3.3 to extend the net-tree construction of Section 3.1 to metric spaces with large spread.

We will use the following special type of metric spaces.

Definition 3.3. A hierarchically well-separated tree (HST) is a metric space defined on the leaves of a rooted tree T. Associated with each vertex $u \in T$ is a label $\Delta_u \geq 0$ such that $\Delta_u = 0$ if and only if u is a leaf of T. The labels are such that if a vertex u is a child of a vertex v, then $\Delta_u \leq \Delta_v$. The distance

between two leaves x and y of T is defined as $\Delta_{\mathsf{lca}(x,y)}$, where $\mathsf{lca}(x,y)$ is the least common ancestor of x and y in T.

The class of HSTs coincides with the class of finite ultrametrics. For convenience, we will assume that the underlying tree is binary (any HST can be converted into a binary HST in linear time, while retaining the underlying metric). We will also associate with every vertex $u \in T$ an arbitrary leaf rep_u of the subtree rooted at u. We also require that $\mathrm{rep}_u \in \{\mathrm{rep}_v | \ v \text{ is a child of } u\}$.

A metric N is called a t-approximation of the metric $\mathcal{M}$ if N and $\mathcal{M}$ are defined on the same set of points and $\forall u, v \in \mathcal{M}, d_{\mathcal{M}}(u,v) \leq d_N(u,v) \leq t \cdot d_{\mathcal{M}}(u,v)$.

It is not hard to see that any n-point metric is $(n-1)$-approximated by some HST (see, e.g., Lemma 3.6). Here we show the following.

Lemma 3.4. *For an n-point metric space $\mathcal{M}$ with doubling constant λ, it is possible to construct in $O(\lambda^6 n \log n)$ expected time an HST which is a $3n^2$ approximation of $\mathcal{M}$.*

This low quality HST will help us later in eliminating the dependence on the spread of the construction time of the net-tree and in distance queries.

We begin proving Lemma 3.4 by constructing a sparse graph that approximates the original metric (this is sometimes called a *spanner*).

Lemma 3.5. *Given an n-point metric space P with doubling constant λ, one can compute a weighted graph G that $3n$-approximates P in $O(\lambda^6 n \log n)$ expected time. The graph G contains $O(\lambda^3 n \log n)$ edges.*

Proof: The construction is recursive. If $n = O(1)$, we just add all the pairs from P as edges. Otherwise, we compute, using Lemma 2.4, a ball $\mathbf{b}(c,r)$ containing at least $m = n/(2\lambda^3)$ points of P with the additional property that $\mathbf{b}(c,2r)$ contains at most $n/2$ points of P.

As such, there exists two numbers r', h such that $r \leq r' \leq 2r$, $h \geq r/n$, and $P \cap \mathbf{b}(c,r') = P \cap \mathbf{b}(c, r'+h)$ (namely, the ring with outer radius $r'+h$ and inner radius r' around c is empty of points of P). Computing r' and h is done by computing the distance of each point from c and partitioning the distance range $[r, 2r]$ into $2n$ segments of equal length. In each segment, we register the point with minimum and maximum distance from c in this range. This can be easily done in $O(n)$ time using the floor function. Next, scan those buckets from left to right. Clearly, the maximum length gap is realized by a maximum of one bucket together with a consecutive nonempty minimum of another bucket. Thus, the maximum length interval can be computed in linear time, and it yields r and h.

Let $P_{\mathrm{in}} = \mathbf{b}(c,r') \cap P$ and let $P_{\mathrm{out}} = P \setminus P_{\mathrm{in}}$. Observe that $d_{\mathcal{M}}(P_{\mathrm{in}}, P_{\mathrm{out}}) = \min_{p \in P_{\mathrm{in}}, q \in P_{\mathrm{out}}} d_{\mathcal{M}}(p,q) \geq h \geq r/n$. Next, we build recursively a spanner for P_{in} and a spanner for P_{out}. We then add the edges between c and all the points of P to the spanner. Let G denote the resulting graph.

Since there are $n/2 \geq |P_{\mathrm{in}}| \geq n/2\lambda^3$ points of P, the running time of the algorithm is $T(|P|) = T(|P_{\mathrm{in}}|) + T(|P_{\mathrm{out}}|) + O(\lambda^3 n) = O(\lambda^6 n \log n)$. Similarly, the number of edges in G is $O(\lambda^3 n \log n)$.

Remaining is the task of proving that G provides a $3n$-approximation to the distances of P. Let G_{in} and G_{out} be the graphs computed for P_{in} and P_{out}, respectively. Consider any two points $u, v \in P$. If u and v are both in P_{in} or both in P_{out}, then the claim follows by induction. Thus, consider the case that $u \in P_{\mathrm{in}}$ and $v \in P_{\mathrm{out}}$. Observe that $d_{\mathcal{M}}(u,v) \geq h \geq r/n$. On the other hand,

$$\begin{aligned} r/n \leq d_{\mathcal{M}}(u,v) \leq d_G(u,v) &\leq d_{\mathcal{M}}(c,u) + d_{\mathcal{M}}(c,v) \\ &\leq r + r + d_{\mathcal{M}}(u,v) \leq (2n+1) d_{\mathcal{M}}(u,v), \end{aligned}$$

since $d_{\mathcal{M}}(c,v) \leq d_{\mathcal{M}}(c,u) + d_{\mathcal{M}}(u,v) \leq r + d_{\mathcal{M}}(u,v)$. Clearly, this implies that $d_G(u,v) \leq 3n d_{\mathcal{M}}(u,v)$, as claimed. ∎

We will later obtain in Theorem 5.3 a near linear time construction of spanners that $(1+\varepsilon)$-approximate the original metric and have a linear number of edges.

Lemma 3.6. *Given a weighted connected graph G on n vertices and m edges, it is possible to construct in $O(n \log n + m)$ time an HST H that $(n-1)$-approximates the shortest path metric on G.*

Proof: Compute the minimum spanning tree of G in $O(n \log n + m)$ time, and let T denote this tree.

Sort the edges of T in nondecreasing order, and add them to the graph one by one. The HST is built from the bottom up. At each point we have a collection of HSTs, each of which corresponds to a connected component of the current graph. When an added edge merges two connected components, we merge the two corresponding HSTs into one by adding a new common root for the two HSTs and labeling this root with the edge's weight times $n-1$. This algorithm is only a slight variation on the Kruskal algorithm and has the same running time.

We next estimate the approximation factor. Let x and y be two vertices of G. Denote by e the first edge that was added in the process above that made x and y to be in the same connected component C. Note that at that point in time, e is the heaviest edge in C, so $w(e) \leq d_G(x,y) \leq (|C|-1)\, w(e) \leq (n-1)\, w(e)$. Since $d_H(x,y) = (n-1)\, w(e)$, we are done. ∎

The proof of Lemma 3.4 now follows by applying Lemma 3.6 on the spanner from Lemma 3.5.

Note that by applying Lemma 3.6 on the spanner from Theorem 5.3, one can obtain a near linear time construction of an HST which $O(n)$-approximates that original metric.

3.3. Extending greedy clustering to metrics of large spread

The main idea in removing the dependence of the running time on the spread is to apply the algorithm of Section 3.1 to a *dynamic* set of points that will correspond to a level of the HST. In more detail, the set of points will correspond to the representatives rep_v, where $\Delta_v \leq r_{\mathrm{curr}}/n^4 \leq \Delta_{\overline{\mathrm{p}}(v)}$, where r_{curr} is the current greedy radius, Δ_v is the HST label of v (i.e., the diameter of the subtree rooted at v), and $\overline{\mathrm{p}}(v)$ is the parent of v in the HST. The algorithm now needs to handle another type of event since, as the algorithm proceeds, the greedy radius decreases to a level in which $\Delta_v \geq r_{\mathrm{curr}}/n^4$. In this case, v should be replaced with its two children u, w. Specifically, if v belongs to a cluster of a point p_i, we remove rep_v from the list of points associated with the cluster of p_i and add rep_u and rep_w to this list (the case where p_i is equal to rep_v is handled in a similar fashion). Next, we need to compute for the new point its nearest center; namely, compute α_{rep_u} and α_{rep_w} (in fact, since $\mathrm{rep}_v = \mathrm{rep}_u$ or $\mathrm{rep}_v = \mathrm{rep}_w$, we need to compute only one of those values). To this end, we scan the friends list of p_i and compute α_{rep_u} and α_{rep_w} from it. This takes $\lambda^{O(1)}$ time. We also need to insert $\{\mathrm{rep}_u, \mathrm{rep}_w\} \setminus \{\mathrm{rep}_v\}$ into the max-heap.

Thus, the algorithm has two heaps. One is a max-heap maintaining the points according to their distances to the nearest center; that is, for every point $p \in P$ we maintain the values of α_p in a max-heap. The second max-heap maintains the nodes of the HST sorted by their diameters Δ (multiplied by a factor of n^4 for normalization). At every point, the algorithm extracts the larger of two heaps and handles it accordingly. One important technicality is that the algorithm is no longer generating the same permutation as `GreedyCluster`, since we are not always picking the furthest point to add as the next center. Rather, we add the furthest active point. We refer to the new algorithm as `NetPermutAlg`.

Lemma 3.7. *Let $\pi = \langle p_1, \ldots, p_n \rangle$ be the permutation of P generated by* `NetPermutAlg`*. Furthermore, let $r_k = \alpha^{k+1}_{p_{k+1}} = \min_{i=1}^{k} d_{\mathcal{M}}(q, p_i)$. Then, $P \subseteq \cup_{i=1}^{k} \mathbf{b}(p_i, (1+n^{-2})r_k)$ and for any $u, v \in \{p_1, \ldots, p_k\}$ we have $d_{\mathcal{M}}(u,v) \geq (1-n^{-2})r_k$.*

Proof: Clearly, the balls of radius r_k around $p_1, \ldots, p_k$ cover all the active points when p_{k+1} was created. However, every active point might represent points which are a distance of r_k/n^2 from it. Thus, by expanding the radius by $(1 + 1/n^2)$, those balls cover all the points.

Observe, that this implies that for any $i < j$ we have $(1+n^{-2})r_i \geq r_j$. In particular, let $\alpha \leq k$ be the minimum number such that $u, v \in \{p_1, \ldots, p_\alpha\}$. Clearly, $d_{\mathcal{M}}(u, v) \geq r_{\alpha-1} \geq r_k/(1+n^{-2}) \geq (1-n^{-2})r_k$. ■

Lemma 3.8. *The expected running time of* `NetPermutAlg` *is* $O(\lambda^{O(1)} n \log n)$.

Proof: Constructing the HST takes $\lambda^{O(1)} n \log n$ expected time, using Lemma 3.4. As in the bounded spread case, we conceptually divide the execution of the algorithm into phases. In the ith phase, the algorithm handles new clusters with radii in the ranges $\operatorname{diam}(P)/2^{i-1}$ and $\operatorname{diam}(P)/2^i$. Consider a point $p \in P$: It is inserted into the point-set when a node v in the HST is "split" at phase i (since p is the representative point for one of the children of v). Let p and q be the two representative points of the two children of v. We charge v for any work done with p and q for the next $L = 10 \log n$ phases. Consider any work done on p before it undergoes another split event. If p is at most L phases away from the split event of v, the vertex v pays for it.

Otherwise, consider p at $> L$ phases away from its latest split event that happened at v. Let r_{curr} be the current clustering radius, and observe that p represents a set of points which has a diameter $\leq r_{\text{curr}}/n^2$ and that $r_{\text{curr}} \leq \Delta_v/n^{10}$. In particular, this implies that $P \cap \mathbf{b}(p, r_{\text{curr}} \cdot n^2) \subset P \cap \mathbf{b}(p, \Delta_v/n^4) \subset P \cap \mathbf{b}(p, r_{\text{curr}}/n^2)$. Namely, all the points that p represents are very far from the rest of the points of P, in terms of r_{curr}. In particular, it cannot be that the cluster that p represents is in any updated friends list in the current stage. (It can be in a friends list that was not updated lately, since we use lazy evaluation. However, when this friends list is used, it will be updated and p will disappear from it. Note that the work required to update the friends lists is $\lambda^{O(1)} n$ overall; see Section 3.1.) Thus, p does not require any work from the algorithm until it undergoes another split event.

Thus, every node in the HST is charged with $\lambda^{O(1)} \log n$ work. It follows that the overall running time of the algorithm is $\lambda^{O(1)} n \log n$. ■

3.4. Constructing the net-tree

In this section we conclude the description of the algorithm for constructing the net-tree and prove Theorem 3.1.

The construction of the net-tree T is done by adding the points of P according to the permutation of `NetPermutAlg`. As mentioned before, the construction algorithm and the resulting tree are similar to the data-structure of Clarkson [Cla02] (our analysis and the guaranteed performance are new, however). The tree constructed for $p_1, \ldots, p_k$ is denoted by $T^{(k)}$ and $T = T^{(n)}$. We obtain $T^{(k)}$ from $T^{(k-1)}$ as follows.

During the construction, we maintain for every vertex $u \in T^{(k)}$ a set of "close by" vertices $\operatorname{Rel}(u)$. Namely, the set $\operatorname{Rel}(u)$ would be in fact the set

$$\overline{\operatorname{Rel}}(u) = \left\{ v \in T^{(k)} \;\middle|\; \ell(v) \leq \ell(u) < \ell(\overline{\mathrm{p}}(v)), \text{ and } d_{\mathcal{M}}(\mathrm{rep}_u, \mathrm{rep}_v) \leq 13 \cdot \tau^{\ell(u)} \right\},$$

where τ is the packing constant associated with the net-tree; see Definition 2.1. (Since we compute $\operatorname{Rel}(u)$ indirectly, the fact that $\operatorname{Rel}(u) = \overline{\operatorname{Rel}}(u)$ requires a formal proof; see Lemma 3.9(v).) The set $\operatorname{Rel}(u)$ is of size $\lambda^{O(1)}$ throughout the algorithm's execution.

We denote $\overline{r}_i = \min \left\{ r_j \;\middle|\; 1 \leq j \leq i \right\}$.

The algorithm. The kth point in the permutation, p_k, will be added as a leaf to the tree $T^{(k-1)}$ to form the tree $T^{(k)}$. As such, we fix $\ell(p_k) = -\infty$ and $\mathrm{rep}_{p_k} = p_k$. Let $l = \lceil \log_\tau \overline{r}_{k-1} \rceil$.

Let h be the largest index such that $\log_\tau \overline{r}_{h-1} > l$ (i.e., p_h is the last added center in the previous phase). Let $q \in \{p_1, \ldots, p_h\}$ be the closest point to p_k among $\{p_1, \ldots, p_h\}$; namely, q is the nearest neighbor to p_k in all the centers present in the previous phase. Identifying q with the unique leaf of $T^{(k-1)}$ whose representative is q, let $u = \overline{\mathrm{p}}(q)$. We obtain $T^{(k)}$ as follows:

(a) If $\ell(u) > l$, then we make a new vertex v and set $\ell(v) = l$ and $\mathrm{rep}_v = q$. We then connect q and p_k as children of v and make v a child of u.

(b) Otherwise, connect p_k as another child of u.

Finding q. Let c_{p_k} be the closest point among $\{p_1, \ldots, p_{k-1}\}$ to p_k (this information is computed by `NetPermutAlg`; see Section 3.1 for details). Denote $\widehat{u} = \overline{\mathrm{p}}(c_{p_k})$. We consider the following two cases:

(1) If $\ell(\widehat{u}) > l$, then $q = \widehat{u}$; see Lemma 3.9(i) for a proof.

(2) Otherwise, $\ell(\widehat{u}) = l$. In this case, q must be in the set $\left\{ \mathrm{rep}_w \;\middle|\; w \in \mathrm{Rel}(\widehat{u}) \right\}$; see Lemma 3.9(i) for a proof. Thus, we just pick q to be the nearest neighbor to p_k in $\left\{ \mathrm{rep}_w \;\middle|\; w \in \mathrm{Rel}(\widehat{u}) \right\}$.

Updating Rel(·). For each new vertex x added we do the following. Let $y = \overline{\mathrm{p}}(x)$. For each $z \in \mathrm{Rel}(y)$, and for each child z' of z, we traverse *part* of the tree rooted at z' in the following way: When visiting a vertex u, we check whether u should be added to $\mathrm{Rel}(x)$ and whether x should be added to $\mathrm{Rel}(u)$ according to the $\overline{\mathrm{Rel}}(\cdot)$ definition, and update $\mathrm{Rel}(x)$ and $\mathrm{Rel}(u)$ accordingly. If x has been added to $\mathrm{Rel}(u)$, then we continue by traversing the children of u. Otherwise, we skip them.

Note that this might require scanning a large fraction of the net-tree, as x might appear in a large number of Rel() lists.

Lemma 3.9. *For any $k \in [1, \ldots, n]$, the tree $T^{(k)}$ has the following properties:*

(i) *The part of the algorithm that finds q indeed finds it.*

(ii) *If v is a child of u, then $d_{\mathcal{M}}(\mathrm{rep}_u, \mathrm{rep}_v) \leq 2 \cdot \tau^{\ell(u)}$.*

(iii) *For every $t \in \Re$, every pair of points in $\mathcal{N}_C(t)$ is at least τ^{t-1} far apart.*

(iv) *$T^{(k)}$ is a net-tree of $\{p_1, \ldots, p_k\}$.*

(v) *For any $u \in T$, $\mathrm{Rel}(u) = \overline{\mathrm{Rel}}(u)$.*

Since the proof of Lemma 3.9 is tedious, we defer it to the appendix. We next analyze the running time.

Lemma 3.10. *Given the (approximate) greedy permutation $\langle p_1, \ldots, p_n \rangle$ with its "current" cluster's center $\langle c_{p_2}, \ldots, c_{p_n} \rangle$, the algorithm for constructing the net-tree runs in $\lambda^{O(1)} n$ time.*

Proof: By the definition of $\mathrm{Rel}(\cdot)$, the size of each such list is at most $\lambda^{O(1)}$. Assuming the tree is implemented reasonably (with pointers from a vertex to its children and parent), constructing the tree clearly takes $O(\lambda^{O(1)})$ time per new point.

Next we estimate the time to construct $\mathrm{Rel}(\cdot)$. For each vertex x added, we first charge $\lambda^{O(1)}$ visits for visiting the children of $\mathrm{Rel}(\overline{\mathrm{p}}(x))$. All the other visits are charged to the parent of the visited vertex. Each vertex has at most $\lambda^{O(1)}$ children, and its children are visited only if a new entry was inserted into its Rel(). As the total size of the $\mathrm{Rel}(\cdot)$ lists is at most $\lambda^{O(1)} n$, we have just bounded the number of visits of vertices during the update process of $\mathrm{Rel}(\cdot)$ to $\lambda^{O(1)} n$. Thus the time spent is $\lambda^{O(1)} n$. ∎

3.5. Augmenting the net-tree

In order to efficiently search on the net-tree, we will need the following three auxiliary data structures.

The first one, given a vertex v of level l, allows us to find all the vertices of "roughly the same level" that are nearby, i.e., those with a representative at a distance of at most $O(\tau^l)$ from the representative of v. More accurately, we need fast access to $\overline{\mathrm{Rel}}(v)$, as defined in Section 3.4. We have seen in that section how to construct it in near linear time such that the whole list can be accessed in $O(\lambda^4)$ time.

The second data-structure enables the following seek operation: Given a leaf x and a level l, find the ancestor y of x such that $\ell(\overline{\mathrm{p}}(y)) > l \geq \ell(y)$. Bender and Farach-Colton [BFC04] present a data-structure $\mathcal{D}$ that can be constructed in linear time over a tree T such that, given a node x and depth d, it outputs the ancestor of x at depth d at x. This takes constant time per query. Thus, performing the seek operation just requires performing a binary search using $\mathcal{D}$ over the net-tree, and this takes $O(\log n)$ time.

Our third data-structure supports a restricted version of the above seek operation: Given a leaf x, an ancestor z of x, and a level l: If $l \notin [\ell(z) - c\log n, \ell(z)]$, return "don't know." Otherwise, return an ancestor y of x satisfying $\ell(\overline{\mathrm{p}}(y)) > l \geq \ell(y)$ (here $c > 0$ is an absolute constant). The data-structure has $O(n)$ space and $O(n \log n)$ preprocessing time, and the queries can be answered in *constant time*.

As a first step, observe that if for every internal vertex z and a descendant leaf x we add vertices to the tree so as to fill all levels between $\ell(z)$ and $\ell(x) - c\log n$ on the path between z and x, then queries to the l level ancestor, $l \in [\ell(x) - c\log n, \ell(z)]$, can be answered by using the data-structure $\mathcal{D}$ as above to find an ancestor of x at depth $\overline{d}(z) - (\ell(z) - l)$. This construction, however, may blow up the number of vertices in the net-tree (and hence the space) by a $\log n$ factor.

To obtain linear space we do the following: In the preprocessing step we enumerate all the possible patterns of existence/nonexistence of vertices in $0.5\log_2 n$ consecutive levels. For each given pattern and each given level in the pattern, we write the number of actual vertices above this level. Preparing this enumeration takes only $O(\sqrt{n}\log n)$ time. Now, for each vertex u of the net-tree, we hold $2c$ pointers to such patterns that together map the vertices in the $c\log n$ level below v on the path to u, where v is an ancestor of u at *depth* $\overline{d}(u) - c\log n$, if such v exists (note that v is $c\log n$ edges above u in the net-tree, but u holds the pattern of only the first $c\log n$ *levels* below v). This data-structure can be clearly computed in $O(n\log n)$ time using top-down dynamic programming on the net-tree.

Given a query (with x, z, and l as above), we do as follows: Let u be an ancestor of x at depth $\max\{\overline{d}(z) + c\log n, \overline{d}(x)\}$. Vertex u can be accessed in $O(1)$ time using the data-structure $\mathcal{D}$. Using the patterns pointed out by u, we can find the depth of the relevant vertex whose level is just below l in $O(1)$ time, and now using $\mathcal{D}$ again we can access this vertex in constant time.

4. Approximate nearest neighbor search

In the following, ANN stands for approximate nearest neighbor. In this section, we present an ANN scheme that preprocesses a given set of points P in near linear time and produces a linear space data-structure which answers queries of the form "given point q, find $p \in P$ such that $d(q,p) \leq (1+\varepsilon)d(q,P)$" in logarithmic time. See Section 1 for more details.

In Section 4.1, we present a variant of Krauthgamer and Lee's [KL04b] net navigation algorithm for the net-tree. This algorithm allows us to boost an A-ANN solution to a $(1+\epsilon)$-ANN solution in $O(\log n + \log A + \epsilon^{O(\dim)})$ query time. In Section 4.2 we present a fast construction of a variant of the ring separator tree [IM98, KL04a] which supports fast $2n$-ANN queries. We conclude in Section 4.3 with the general scheme which is a combination of the previous two.

4.1. The low spread case

Lemma 4.1. *We are given a net-tree T of P, a query point $q \in \mathcal{M}$, and a vertex $u \in T$ at level $l = \ell(u)$ such that $d_{\mathcal{M}}(\mathrm{rep}_u, q) \leq 5 \cdot \tau^l$ or $\widehat{p} \in P_u$, where $\widehat{p}$ is the nearest neighbor to q in P. Then there is an algorithm that traverses T form u downward such that for any $t \in \mathbb{N}$, after $t+4$ steps, the algorithm reaches a vertex s for which rep_s is a $(1+\tau^{l-f-t})$-ANN, where $f = \log_\tau d_{\mathcal{M}}(\widehat{p}, q)$. The running time of this search is $\lambda^{O(1)} \min\{t, l-f\} + \lambda^{O(\max\{t-(l-f),0\})}$.*

Proof: The query algorithm works as follows. It constructs sets A_i of vertices in T with the following properties:

1. For each $v \in A_i$, $\ell(\overline{\mathrm{p}}(v)) > i \geq \ell(v)$.
2. $\widehat{p} \in \cup_{v \in A_i} P_v \subset \mathbf{b}(q, d_{\mathcal{M}}(q, \widehat{p}) + (13 + \frac{2\tau}{\tau-1}) \cdot \tau^i)$.

The algorithm starts by setting $A_l = \mathrm{Rel}(u)$. If $\widehat{p} \in P_u$, then A_l clearly satisfies the two properties above. If $d_{\mathcal{M}}(\mathrm{rep}_u, q) \leq 5 \cdot \tau^l$, then $d_{\mathcal{M}}(\mathrm{rep}_u, \widehat{p}) \leq 10\tau^l$. Suppose for the sake of contradiction that $\widehat{p} \notin \cup_{v \in A_l} P_v$; then $\exists v'$ such that $\ell(v') \leq l$, $d_{\mathcal{M}}(\mathrm{rep}_u, \mathrm{rep}_{v'}) > 13\tau^l$, and $\widehat{p} \in P_{v'}$. But then from the covering property, $d_{\mathcal{M}}(\mathrm{rep}_{v'}, \widehat{p}) \leq \frac{2\tau}{\tau-1}\tau^l$, which means that $d_{\mathcal{M}}(\mathrm{rep}_u, \widehat{p}) > (13 - \frac{2\tau}{\tau-1})\tau^l > 10\tau^l$, a contradiction.

The set A_{i-1} is constructed from A_i as follows. Let $v \in A_i$ be the closest vertex in A_i to q, i.e., $d_{\mathcal{M}}(\mathrm{rep}_v, q) = \min_{w \in A_i} d_{\mathcal{M}}(\mathrm{rep}_w, q)$. Let B be the set obtained from A_i by replacing every vertex of level i with its children. The set A_{i-1} is obtained from B by throwing out any vertex w for which $d_{\mathcal{M}}(q, \mathrm{rep}_w) > d_{\mathcal{M}}(q, \mathrm{rep}_v) + \frac{2\tau}{\tau-1} \cdot \tau^{i-1}$. It is easily checked that A_{i-1} has the required properties.

The running time is clearly dominated by $\lambda^{O(1)}$ times the sum of the A_i's sizes. For $i > f$, $d_{\mathcal{M}}(q, \mathrm{rep}_v)$ is at most $\frac{2\tau}{\tau-2} \cdot \tau^i$, and therefore $|A_i| \leq \lambda^{O(1)}$. For $i \leq f$, we have only a weak bound of $|A_i| \leq \lambda^{O(f-i)}$. Thus the running time of the algorithm for t steps follows. Notice that any point in A_{l-i} is $(1+\tau^{l-f-i+4})$-ANN. ∎

For a set P with spread Φ, by applying the algorithm of Lemma 4.1 with u the root of T, and $t = \lceil \log_\tau(\Phi/\varepsilon) - f \rceil$, Lemma 4.1 gives a $(1+\varepsilon)$-approximate nearest neighbor scheme with $O(n \log n)$ expected construction time and $O(\log \Phi + \varepsilon^{-O(\dim)})$ query time. (Note that the algorithm does not need to know t (and thus f) in advance—it can estimate the current approximation by comparing $d_{\mathcal{M}}(q, \mathrm{rep}_v)$ to τ^i.) This gives an alternative to the data-structure of Krauthgamer and Lee [KL04b], with a slightly faster construction time. Their construction time is $O(n \log \Phi \log\log \Phi)$ if one uses the insertion operation for their data-structure (note that in the constant doubling dimension setting, $\log n = O(\log \Phi)$). In fact, in this case, the Rel() data-structure is not needed since $\mathrm{Rel}(\mathrm{root}) = \{\mathrm{root}\}$. Therefore the storage for this ANN scheme is $O(n)$, with no dependency on the dimension. A similar construction was obtained independently in [BKL04]. However, its construction time is $O(n^2)$.

4.2. Low quality ring separator tree

Lemma 4.2. *One can construct a data-structure which supports $2n$-ANN queries in $2^{O(\dim)} \log n$ time. The construction time is $2^{O(\dim)} n \log n$, and the data-structure uses $2^{O(\dim)} n$ space.*

Proof: The data structure is a binary search tree S, in which each vertex of the tree v is associated with a point $p_v \in P$ and radius r_v. We are guaranteed that $n/2\lambda^3 \leq |\mathbf{b}(p_v, r_v)| \leq (1 - 1/2\lambda^3)n$ and that $(\mathbf{b}(p_v, (1+1/2n)r_v) \setminus \mathbf{b}(p_v, (1-1/2n)r_v)) \cap P = \emptyset$. The left subtree is recursively constructed on the

set $P \cap \mathbf{b}(p_v, r_v)$, and the right subtree is recursively constructed on $P \setminus \mathbf{b}(p_v, r_v)$. The depth of S is clearly at most $O(\lambda^3 \log n)$.

The construction of S is similar to the construction of the low quality spanner (Section 3.2) and uses Lemma 2.4 as follows. Apply Lemma 2.4 to find $p \in P$ and r such that $|\mathbf{b}(p,r)| \geq n/(2\lambda^3)$, whereas $|\mathbf{b}(p,2r)| \leq n/2$. From the pigeonhole principle, there exists $r' \in [(1+1/2n)r, 2r - r/2n)$ for which $\mathbf{b}(p,(1+1/2n)r') \setminus \mathbf{b}(p,(1-1/2n)r') = \emptyset$. We now make a root v for the ring separator tree, set $p_v = p$ and $r_v = r'$, and recurse on $\mathbf{b}(p_v, r_v)$ as the left subtree and $P \setminus \mathbf{b}(p_v, r_v)$ as the right subtree. The construction time $T(n)$ obeys the recursive formula $T(n) = T(n_1) + T(n_2) + O(n)$, where $n_1 + n_2 = n$, $n/2\lambda^3 \leq n_1 \leq n/2$.

Once we have this data-structure, $2n$-ANN can be found in $O(\lambda^3 \log n)$ time as follows. Let the root of the ring separator tree be u. Given a query point q, check its distance to p_u. If $d_{\mathcal{M}}(q, p_u) \leq r_u$, then recurse on the left subtree. Otherwise, recurse on the right subtree. At the end, return the nearest point to q among p_v, where v is on the path traversed by the algorithm.

The running time of this procedure is clearly dominated by the height of the tree which is $O(\lambda^3 \log n)$.

To see that this is indeed $2n$-ANN, let a be the vertical path in the tree traversed by the algorithm, and let b be the vertical path in the tree connecting the root to the nearest neighbor of q in P. Let v be the lowest common vertex of a and b. Suppose that a continued on the left subtree of v while b continued on the right subtree. In this case the distance from q to the nearest neighbor is at least $r_v/2n$, while $d_{\mathcal{M}}(p_v, q) \leq r_v$. Thus p_v is $2n$-ANN.

If a continued on the right subtree of v while b continued on the left subtree of v, then the distance from the nearest neighbor is at least $r_v/2n + (d_{\mathcal{M}}(p_v, q) - r_v)$, while p_v is at distance $d_{\mathcal{M}}(p_v, q)$. The ratio between these two quantities is clearly at most $2n$. ∎

Remark 4.3. As is pointed out in [IM98, KL04a], it is possible to duplicate points in the ring for the two subtrees. Hence we can actually partition the $\mathbf{b}(p,2r) \setminus \mathbf{b}(p,r)$ into $t \leq n$ subrings and choose to duplicate a "light" ring. When $t = 1$, we obtain the ring separator tree from [KL04a] that supports $O(1)$-ANN queries, but requires $n^{2^{O(\dim)}}$ storage. For general $t \leq n$ we obtain a data-structure that supports $O(t)$-ANN queries, and that by choosing the right ring to duplicate, consumes only $n^{(3\log 2\lambda)^{1/t}}$ storage. To see this, we set $\beta = (3\log 2\lambda)^{1/t}$ and prove by induction on n that it is possible to find a ring such that the number of leaves in the tree is at most n^β. Denote $\eta_i = |\mathbf{b}(p,(1+i/t)r)|/n$. Note that $(2\lambda)^{-3} \leq \eta_0 \leq \eta_1 \leq \cdots \eta_t \leq n/2$, and therefore there exists $i \leq t$ for which $\eta_{i-1} \geq \eta_i^\beta$; otherwise $(2\lambda)^{-3} < \eta_0^{\beta^t} \leq (1/2)^{\beta^t}$ which is a contradiction. Thus by duplicating the ith ring, and by applying the inductive hypothesis on the number of leaves in the subtrees, the resulting tree will have at most $(\eta_i n)^\beta + ((1-\eta_{i-1})n)^\beta \leq (\eta_{i-1} + (1-\eta_{i-1}))n^\beta$ leaves.

Thus, setting $t = O(\log\log\lambda \cdot \log n)$, we obtain a linear space ring separator tree that supports $O(t)$-ANN queries in $O(\log n)$ time.

4.3. ANN algorithm for arbitrary spread

The algorithm for arbitrary spread is now pretty clear. During the preprocessing we construct the augmented net-tree from Section 3. We also construct the low quality ring separator tree. The construction time is $2^{O(\dim)} n \log n$, and the space used is $2^{O(\dim)} n$.

Given a query point $q \in \mathcal{M}$, and the approximation parameter $\varepsilon > 0$, the query algorithm consists of the following three steps:

1. First, find $2n$-ANN p_1 using the low quality ring separator tree of Section 4.2.

2. Next, find a vertex u in the net-tree that is an ancestor for p_1 and that satisfies

$$\ell(\overline{\mathrm{p}}(u)) - 1 \geq \lceil \log_\tau(16 \cdot d_{\mathcal{M}}(p_1, q)) \rceil \geq \ell(u).$$

Hence

$$d_{\mathcal{M}}(\mathrm{rep}_u, q) \leq d_{\mathcal{M}}(\mathrm{rep}_u, p_1) + d_{\mathcal{M}}(p_1, q) \leq 2.5 \cdot \tau^{\ell(u)} + \tfrac{1}{16}\tau^{\ell(\overline{\mathrm{p}}(u))-1}.$$

3. We now split the analysis into two cases as follows:

   (a) If $2.5 \cdot \tau^{\ell(u)} \geq \frac{1}{16}\tau^{\ell(\overline{\mathrm{p}}(u))-1}$, then clearly $d_{\mathcal{M}}(\mathrm{rep}_u, q) \leq 5\tau^{\ell(u)}$, and thus u satisfies the conditions of Lemma 4.1.

   (b) If, on the other hand, $2.5 \cdot \tau^{\ell(u)} < \frac{1}{16}\tau^{\ell(\overline{\mathrm{p}}(u))-1}$, then the packing property of the net-tree implies that

$$\begin{aligned} P \cap \mathbf{b}(q, d_{\mathcal{M}}(q, \mathrm{rep}_u)) &\subset P \cap \mathbf{b}(\mathrm{rep}_u, 2d_{\mathcal{M}}(q, \mathrm{rep}_u)) \\ &\subset P \cap \mathbf{b}\left(\mathrm{rep}_u, \frac{1}{4} \cdot \tau^{\ell(\overline{\mathrm{p}}(u))-1}\right) \subset P_u, \end{aligned}$$

   and therefore $\widehat{p} \in P_u$. Thus, in this case u also satisfies the conditions of Lemma 4.1.

4. Set $l = \ell(u)$. Using the notation of Lemma 4.1, the fact that p_1 is a $2n$-ANN implies that $f \geq l - (1 + \log n)$, thus by setting the number of steps to $t = \lceil \log(n/\varepsilon) \rceil$, and applying the algorithm of Lemma 4.1, we obtain $(1+\varepsilon)$-ANN.

The running time of the query is

$$\lambda^{O(1)} \log n + O(\log n) + \lambda^{O(1)} \log n + \varepsilon^{-O(\dim)} \leq \lambda^{O(1)} \log n + \varepsilon^{-O(\dim)}.$$

We summarize as follows.

Theorem 4.4. *Given a set P of n points of bounded doubling dimension* dim *in a metric space $\mathcal{M}$, one can construct a data-structure for answering ANN queries (where the quality parameter ε is provided together with the query). The query time is $2^{O(\dim)} \log n + \varepsilon^{-O(\dim)}$, the expected preprocessing time is $2^{O(\dim)} n \log n$, and the space used is $2^{O(\dim)} n$.*

Theorem 4.4 compares quite favorably with the result of Krauthgamer and Lee [KL04a], which solves the same problem with the same (tight) query time but uses $O(2^{O(\dim)} n^2 \operatorname{polylog}(n))$ space.

5. Fast construction of WSPD and spanners

Let P be an n-point subset of a metric space $\mathcal{M}$ with doubling dimension dim and a parameter $1/4 > \varepsilon > 0$. Denote by $A \otimes B$ the set $\left\{ \{x, y\} \,\middle|\, x \in A,\, y \in B \right\}$. A WSPD with parameter ε^{-1} of P is a set of pairs $\{\{A_1, B_1\}, \ldots, \{A_s, B_s\}\}$ such that

1. $A_i, B_i \subset P$ for every i.
2. $A_i \cap B_i = \emptyset$ for every i.
3. $\cup_{i=1}^{s} A_i \otimes B_i = P \otimes P$.

4. $d_{\mathcal{M}}(A_i, B_i) \geq \varepsilon^{-1} \cdot \max\{\mathrm{diam}(A_i), \mathrm{diam}(B_i)\}$.

The notion of WSPD was defined by Callahan and Kosaraju [CK95] for Euclidean spaces. Talwar [Tal04] have shown that this notion transfers to constant doubling metrics. In particular, he proves that any n-point metric with doubling dimension dim admits WSPD in which the number of pairs is $n\varepsilon^{-O(\dim)} \log \Phi$. We improve this result.

Lemma 5.1. *For $1 \geq \varepsilon > 0$, one can construct an ε^{-1}-WSPD of size $n\varepsilon^{-O(\dim)}$, and the expected construction time is $2^{O(\dim)} n \log n + n\varepsilon^{-O(\dim)}$.*

Furthermore, the pairs of the WSPD *correspond to (P_u, P_v), where u, v are vertices of a net-tree of P, and for any pair (P_u, P_v) in* WSPD, $\mathrm{diam}(P_u), \mathrm{diam}(P_v) \leq \varepsilon d_P(\mathrm{rep}_u, \mathrm{rep}_v)$.

Proof: We compute the net-tree T using Theorem 3.1. For concreteness of the WSPD, assume also that some weak linear order $\preceq$ is defined on the vertices of T. The WSPD is constructed by calling $\mathsf{genWSPD}(u_0, u_0)$, where u_0 is the root of the net-tree T, and $\mathsf{genWSPD}(u, v)$ is defined recursively as follows:

$\mathsf{genWSPD}(u, v)$
  Assume $\ell(u) > \ell(v)$ or $(\ell(u) = \ell(v)$ and $u \preceq v)$
    (otherwise exchange $u \leftrightarrow v$).
  If $8\frac{2\tau}{\tau-1} \cdot \tau^{\ell(u)} \leq \varepsilon \cdot d_{\mathcal{M}}(\mathrm{rep}_u, \mathrm{rep}_v)$, then
    return $\{\{u, v\}\}$
  else
    Denote by $u_1, \ldots, u_r$ the children of u
    return $\bigcup_{i=1}^{r} \mathsf{genWSPD}(u_i, v)$.

For any node $u \in T$, we have $\mathrm{diam}(P_u) \leq 2\frac{2\tau}{\tau-1} \cdot \tau^{\ell(u)}$ (see Definition 2.1). In particular, for every output pair $\{u, v\}$, it holds that

$$\begin{aligned} \max\{\mathrm{diam}(P_u), \mathrm{diam}(P_v)\} &\leq 2\frac{2\tau}{\tau-1} \cdot \max\left\{\tau^{\ell(u)}, \tau^{\ell(v)}\right\} \leq \frac{\varepsilon}{4} d_P(\mathrm{rep}_u, \mathrm{rep}_v) \\ &\leq \frac{\varepsilon}{4}(d_P(P_u, P_v) + \mathrm{diam}(P_u) + \mathrm{diam}(P_v)), \end{aligned}$$

and so $\max\{\mathrm{diam}(P_u), \mathrm{diam}(P_v)\} \leq \frac{\varepsilon}{4(1-\varepsilon/2)} d_P(P_u, P_v) \leq \varepsilon d_P(P_u, P_v)$, since $\varepsilon \leq 1$. Similarly, for any $x \in P_u$ and $y \in P_v$, we have

$$d_P(\mathrm{rep}_u, \mathrm{rep}_v) \leq d_P(x, y) + \mathrm{diam}(P_u) + \mathrm{diam}(P_v) \leq (1 + \varepsilon) d_P(x, y).$$

One can verify that every pair of points is covered by a pair of subsets $\{P_u, P_v\}$ output by the $\mathsf{genWSPD}$ algorithm.

We are left to argue about the size of the output (the running time is clearly linear in the output size). Let $\{u, v\}$ be an output pair and assume that the call to $\mathsf{genWSPD}(u, v)$ was issued by $\mathsf{genWSPD}(u, \overline{\mathrm{p}}(v))$. We charge this call to $\overline{\mathrm{p}}(v)$, and we will prove that each vertex is charged at most $\varepsilon^{-O(\dim)}$ times.

Fix $v' \in T$. It is charged by pairs of the form $\{u, v\}$ in which $\overline{\mathrm{p}}(v) = v'$, and which were issued inside $\mathsf{genWSPD}(u, v')$. This implies that $\ell(\overline{\mathrm{p}}(u)) \geq \ell(v') \geq \ell(u)$.

Since the pair (u, v') was not generated by genWSPD, we conclude that $d_P(\mathrm{rep}_{v'}, \mathrm{rep}_u) \leq (8\frac{2\tau}{\tau-1} \cdot \tau^{\ell(v')})/\varepsilon$. The set

$$U = \left\{ w \;\middle|\; \ell(\overline{\mathrm{p}}(w)) \geq \ell(v') \geq \ell(w) \text{ and } d_P(\mathrm{rep}_{v'}, \mathrm{rep}_w) \leq 8\frac{2\tau}{\varepsilon(\tau-1)} \cdot \tau^{\ell(v')} \right\}$$

contains u, and U is a subset of $\mathcal{N}_C(\ell(v'))$. By Proposition 2.2, for every $u_1, u_2 \in U$, if $u_1 \neq u_2$, then $d_P(P_{u_1}, P_{u_2}) \geq \tau^{\ell(v')-1}/4$. By the doubling property, we have $|U| \leq \varepsilon^{-O(\dim)}$. We therefore infer that v' can be charged only by pairs in $U \times C_{v'}$, where $C_{v'}$ is the set of children of v'. We conclude that v' might be charged at most $|U| \cdot |C_{v'}| \leq (2/\varepsilon)^{O(\dim)} = \varepsilon^{-O(\dim)}$ times. Thus, the total number of pairs generated by the algorithm is $n\varepsilon^{-O(\dim)}$. ∎

5.1. Spanners

Definition 5.2. A t-spanner of a finite metric space P is a weighted graph G whose vertices are the points of P, and for any $x, y \in P$,

$$d_P(x, y) \leq d_G(x, y) \leq t \cdot d_P(x, y),$$

where d_G is the metric of the shortest path on G.

Theorem 5.3. *Given an n-point metric P with doubling dimension* dim *and parameter $1 \geq \varepsilon > 0$, one can compute a $(1+\varepsilon)$-spanner of P with $n\varepsilon^{-O(\dim)}$ edges, in $2^{O(\dim)} n \log n + n\varepsilon^{-O(\dim)}$ expected time.*

Proof: Let $c \geq 16$ be an arbitrary constant, and set $\delta = \varepsilon/c$. Compute a δ^{-1} WSPD decomposition using the algorithm of the previous section. For every pair $\{u, v\} \in$ WSPD, add an edge between $\{\mathrm{rep}_u, \mathrm{rep}_v\}$ with weight $d_P(\mathrm{rep}_u, \mathrm{rep}_v)$. Let G be the resulting graph; clearly, the resulting shortest path metric d_G dominates the metric d_P.

The upper bound on the stretch is proved by induction on the length of pairs in the WSPD. Fix a pair $x, y \in P$; by our induction hypothesis, we have that for every pair $z, w \in P$ such that $d_P(z, w) < d_P(x, y)$, it holds that $d_G(z, w) \leq (1 + c\delta) d_P(z, w)$.

The pair x, y must appear in some pair $\{u, v\} \in$ WSPD, where $x \in P_u$ and $y \in P_v$. Thus $d_P(\mathrm{rep}_u, \mathrm{rep}_v) \leq (1 + 2\delta) d_P(x, y)$ and $d_P(x, \mathrm{rep}_u), d_{\mathcal{M}}(y, \mathrm{rep}_v) \leq \delta d_{\mathcal{M}}(\mathrm{rep}_u, \mathrm{rep}_v)$ by Lemma 5.1. By the inductive hypothesis

$$\begin{aligned}
d_G(x, y) &\leq d_G(x, \mathrm{rep}_u) + d_G(\mathrm{rep}_u, \mathrm{rep}_v) + d_G(\mathrm{rep}_v, y) \\
&\leq (1 + c\delta) d_P(x, \mathrm{rep}_u) + d_P(\mathrm{rep}_u, \mathrm{rep}_v) + (1 + c\delta) d_P(\mathrm{rep}_v, y) \\
&\leq 2(1 + c\delta) \cdot \delta \cdot d_P(\mathrm{rep}_u, \mathrm{rep}_v) + d_P(\mathrm{rep}_u, \mathrm{rep}_v) \\
&\leq (1 + 2\delta + 2c\delta^2)(1 + 2\delta) d_P(x, y) \\
&\leq (1 + \varepsilon) d_P(x, y),
\end{aligned}$$

since $\delta c \leq \varepsilon \leq 1$ and $16\delta \leq 1$ and $c \geq 11$. ∎

6. Compact representation scheme

A *CRS* of a finite metric space P is a "compact" data-structure that can answer distance queries for pairs of points. We measure the performance of a CRS using four parameters $(\mathsf{P}, \mathsf{S}, \mathsf{Q}, \overline{\kappa})$, where P is

the preprocessing time of the distance matrix, S is the space used by the CRS (in terms of words), Q is the query time, and $\overline{\kappa}$ is the approximation factor.

The distance matrix by itself is a $(\mathsf{P} = O(1), \mathsf{S} = O(n^2), \mathsf{Q} = O(1), \overline{\kappa} = 1)$-CRS. The ε^{-1}-WSPD as well as the $(1+\varepsilon)$-spanner are representations of $(1+O(\varepsilon))$-approximation of the metric that consumes only $\varepsilon^{-O(\dim)}n$ space. However, naively it takes $\Omega(n)$ time to answer approximate distance queries in these data-structures.

In this section, we obtain the following theorem.

Theorem 6.1. *For any n point metric with doubling dimension* dim*, the following exist:*

(a) *A $(\mathsf{P} = 2^{O(\dim)} n \log^2 n + \varepsilon^{-O(\dim)}n, \mathsf{S} = \varepsilon^{-O(\dim)}n, \mathsf{Q} = 2^{O(\dim)}, \overline{\kappa} = 1+\varepsilon)$-CRS.*

(b) *A $(\mathsf{P} = 2^{O(\dim)} \cdot \mathrm{poly}(n) + \varepsilon^{-O(\dim)}n, \mathsf{S} = \varepsilon^{-O(\dim)}n, \mathsf{Q} = O(\dim), \overline{\kappa} = 1+\varepsilon)$-CRS.*

For general n-point metrics, Thorup and Zwick [TZ01] obtained a $(kn^{1+1/k}, kn^{1+1/k}, O(k), 2k-1)$-CRS, where $k \in \mathbb{N}$ is a prescribed parameter. The trade-off between the approximation and the space is essentially tight for general metrics. Closer in spirit to our setting, Gudmunsson et al. [GLNS02a, GLNS02b] considered metrics that are t-approximated by Euclidean distances in $\Re^d$, where both d and t are (possibly large) constants. They showed that such metrics have $(O(n \log n), O(n), O(1), 1+\varepsilon)$-CRS (the O notation here hides constants that depend on ε, d, and t). Our scheme strictly extends[3] their result since metrics that are t-approximated by a set of points in the d-dimensional Euclidean space have doubling dimension at most $O(d \log(2t))$. We further discuss previous work on a special type of CRS, called *distance labeling*, in Section 6.3.

Our scheme is naturally composed of two parts. In Section 6.1 we show that by using the net-tree it is possible to convert an A-approximate CRS into a $(1+\varepsilon)$-approximate CRS in essentially $O(\log A)$ query time (and even $O(\log \log A)$ query time). We then show in Section 6.2 how to obtain an $O(1)$-approximate CRS using Assouad's embedding. In Section 6.3 we observe that Assouad's embedding can be used in distance labeling schema.

6.1. Approximation boosting lemma

Assume we are given a data-structure $\mathcal{A}$, which is a $(\mathsf{P}, \mathsf{S}, \mathsf{Q}, \overline{\kappa})$-CRS of a set $P \subset \mathcal{M}$, where $\overline{\kappa} \leq 3n^2$. In this section, we derive a CRS with improved approximation. Besides storing the data-structure of $\mathcal{A}$, we also need the following data-structures:

1. The net-tree T, augmented so that it supports the following operations:
   (a) $O(\log n)$ time access for ancestors of a given level as defined in Section 3.5.
   (b) Constant time access for an ancestor at a given level l of a given vertex x, when $l + 6\log n$ is at least the level of a given ancestor z of x.
   (c) A constant time access for the lca of two vertices in T [BF00].
2. An ε^{-1}-WSPD W on the net-tree T, with support for fast membership queries. For each pair we also store the distance between its representatives. By using hashing membership, queries can be answered in constant time.
3. The $(3n^2)$-approximation HST H of Section 3.2. The HST H should be augmented with the following features:

[3] Caveat: They use a weaker model of computation.

(a) A constant time access to lca queries, after a linear time preprocessing [BF00].

(b) Each vertex u of H contains pointers to the following set of vertices in T:

$$K_u = \{x \in T : \ d_{\mathcal{M}}(\mathrm{rep}_x, \mathrm{rep}_u) \leq 4\Delta_u \text{ and } \ell(x) < \log \Delta_u \leq \ell(\overline{\mathrm{p}}(x))\}\,.$$

Note that $|K_u| \leq \lambda^{O(1)}$, and computing all these sets can be accomplished in $\lambda^{O(1)} n \log n$ time by finding the level $\lceil \log \Delta_u \rceil$ ancestor z of rep_u in T in $O(\log n)$ time, and then scanning $\mathrm{Rel}(z)$.

All these data-structures can be created in $2^{O(\dim)} n \log n + \varepsilon^{-O(\dim)} n$ time and $\varepsilon^{-O(\dim)} n$ space.

Assuming Query-$\mathcal{A}(x,y)$ returns a value η, such that $d_{\mathcal{M}}(x,y)/\overline{\kappa} \leq \eta \leq d_{\mathcal{M}}(x,y)$, the query algorithm is as follows:

```
Query-B(x, y ∈ P)
    z ← lca_H(x, y).
    u' ← ancestor of x in T among K_z, v' ← ancestor of y in T among K_z.
    η ← Query-A(x, y).
    u_0 ← ancestor of x in level ⌊log(εη)⌋, v_0 ← ancestor of y in level ⌊log(εη)⌋.
    u ← u_0, v ← v_0.
    while {u, v} ∉ W do
        if ℓ(p̄(u)) < ℓ(p̄(v)) or ( ℓ(p̄(u)) = ℓ(p̄(v)) and p̄(v) ⪯ p̄(v) ) then
            u ← p̄(u)
        else
            v ← p̄(v).
    return d_M(rep_u, rep_v).
```

Implementation details. u' is found by scanning all vertices in K_z (there are only $\lambda^{O(1)}$ such vertices) and checking which one of them is an ancestor of x in T (ancestorship can be checked using the lca operation on T). Note that an ancestor of x must be contained in K_z, since $d_{\mathcal{M}}(\mathrm{rep}_z, x) \leq \Delta_z$, and thus the ancestor of the level immediately below $\log \Delta_z$ must be in K_z. A similar thing happens with v'. Both η and Δ_z are $3n^2$-approximations to $d_{\mathcal{M}}(x,y)$, and therefore $\ell(u') - \ell(u_0) \leq 4 \log n + 3$; hence u_0 can be accessed in constant time. The same goes for v_0.

The following lemma is an immediate consequence of the way in which the WSPD algorithm works.

Lemma 6.2. *For a pair $\{s,t\} \in W$ (the ε^{-1}-WSPD), and $\ell(s) \leq \ell(t)$, one of the following conditions must be satisfied:*

1. $\ell(s) \leq \ell(t) < \ell(\overline{\mathrm{p}}(s))$ *and* $\frac{2\tau}{\tau-1} \cdot \tau^{\ell(\overline{\mathrm{p}}(s))} > \varepsilon \cdot d_{\mathcal{M}}(\mathrm{rep}_{\overline{\mathrm{p}}(s)}, \mathrm{rep}_t)$, *and* $\frac{2\tau}{\tau-1} \cdot \tau^{\ell(s)} \leq \varepsilon \cdot d_{\mathcal{M}}(\mathrm{rep}_s, \mathrm{rep}_t)$.
2. $\ell(s) < \ell(t) = \ell(\overline{\mathrm{p}}(s))$, *and* $\overline{\mathrm{p}}(s) \preceq t$, *and* $\frac{2\tau}{\tau-1} \cdot \tau^{\ell(\overline{\mathrm{p}}(s))} > \varepsilon \cdot d_{\mathcal{M}}(\mathrm{rep}_{\overline{\mathrm{p}}(s)}, \mathrm{rep}_t)$. $\frac{2\tau}{\tau-1} \cdot \tau^{\ell(s)} \leq \varepsilon \cdot d_{\mathcal{M}}(\mathrm{rep}_s, \mathrm{rep}_t)$.

Proposition 6.3. *The while loop finds a pair in W after $O(\log \overline{\kappa})$ steps.*

Proof: Denote by $\{u_0, v_0\}$ the pair with which the loop begins. It is straightforward to see that the loop climbs through all ancestor pairs $\{u,v\}$ of $\{u_0,v_0\}$ that satisfy either (i) $\ell(u) \leq \ell(v) < \ell(\overline{\mathrm{p}}(u))$, or (ii) $\ell(u) < \ell(v) = \ell(\overline{\mathrm{p}}(u))$ and $\overline{\mathrm{p}}(u) \preceq v$.

Thus, if an ancestor pair exists in W, it will be found by the loop. As we argue in Lemma 5.1, there exists an ancestor pair $\{\bar{u}, \bar{v}\}$ of $\{x,y\}$ in W. Our choice $\{u_0, v_0\}$ ensures that u_0 is a descendant of $\bar{u}$ at most $O(\log \overline{\kappa})$ levels down T, and the same goes for v_0 and $\bar{v}$. ∎

Combining the above claims, implies the following.

Lemma 6.4. *Let P be an n-point metric. Assume we are given a $(\mathsf{P}, \mathsf{S}, \mathsf{Q}, \overline{\kappa})$-CRS $\mathcal{A}$ of a set P, where $\overline{\kappa} \leq 3n^2$. Then, one can obtain $(\mathsf{P}', \mathsf{S}', \mathsf{Q}', 1+\varepsilon)$-CRS $\mathcal{B}$ of P, where $\mathsf{P}' = \mathsf{P} + 2^{O(\dim)} n \log n + \varepsilon^{O(\dim)} n$, $\mathsf{S}' = \mathsf{S} + \varepsilon^{-O(\dim)} n$, $\mathsf{Q}' = \mathsf{Q} + O(\log \overline{\kappa})$.*

Remark 6.5. The dependence of the query time on $\overline{\kappa}$ can be improved from $O(\log \overline{\kappa})$ to $O(\log \log \overline{\kappa})$ without sacrificing any other parameter. The idea is to replace the "ladder climbing" in the algorithm above (the while loop) with a binary search on the $\log \overline{\kappa}$ levels. To do so we change the WSPD procedure to output *all* pairs it encounters. This clearly does not change asymptotically the size of W. We do a binary search on the $\log \overline{\kappa}$ relevant levels to find the lowest level pairs which still appear in the WSPD, and this gives the relevant pairs. We do not pursue this improvement rigorously, since in the CRS that we develop in the next section, the query time Q dominates $\overline{\kappa}$ anyway, and thus this would lead to no asymptotic savings in the query time.

6.2. Assouad embedding

To quickly obtain a constant approximation of the distance, we will use a theorem due to Assouad [Ass83] (see also [Hei01, GKL03]). The following is a variant of the original statement, tailored for our needs, and its proof is provided for the sake of completeness.

Theorem 6.6. *Any metric space $\mathcal{M}$ with doubling dimension* dim *can be embedded in ℓ_∞^d, where $d \leq \varepsilon^{-O(\dim)}$, such that the metric $(\mathcal{M}, \sqrt{d_{\mathcal{M}}})$ is distorted by a factor of $1+\varepsilon$.*

Proof: Fix $r > 0$. We begin by constructing an embedding $\phi^{(r)} : \mathcal{M} \to \Re^{d_1}$, where $d_1 = \varepsilon^{-O(\dim)}$ with the following properties for every $x, y \in \mathcal{M}$:

1. $\left\| \phi^{(r)}(x) - \phi^{(r)}(y) \right\|_\infty \leq \min\{r, d_{\mathcal{M}}(x,y)\}$.
2. If $d_{\mathcal{M}}(x,y) \in [(1+\varepsilon)r, 2r)$, then $\left\| \phi^{(r)}(x) - \phi^{(r)}(y) \right\|_\infty \geq (1-\varepsilon)r$.

We take an εr-net $\mathcal{N}^{(r)}$ of $\mathcal{M}$ and color it such that every pair $x, y \in \mathcal{N}^{(r)}$ for which $d_{\mathcal{M}}(x,y) \leq 4r$ is colored differently. Clearly, $d_1 = \varepsilon^{-O(\dim)}$ colors suffice. Associate with every color i a coordinate, and for $x \in \mathcal{M}$ define $\phi_i^{(r)}(x) = \max\{0, r - d_{\mathcal{M}}(x, C_i)\}$, where $C_i \subset \mathcal{N}^{(r)}$ is the set of points of color i.

We next check that the two properties above are satisfied. As $\phi_i^{(r)}(x) \in [0, r]$, it is clear that $|\phi_i^{(r)}(x) - \phi_i^{(r)}(y)| \leq r$ for every color i. The 1-Lipschitz property easily follows from the triangle inequality.

Next, assume that $d_{\mathcal{M}}(x,y) \in [(1+\varepsilon)r, 2r]$. Since $d_{\mathcal{M}}(x, \mathcal{N}^{(r)}) \leq \varepsilon r$, there exists a color i for which $d_{\mathcal{M}}(x, C_i) \leq \varepsilon r$. This implies (by the triangle inequality) that $d_{\mathcal{M}}(y, C_i) \geq r$, and hence $|\phi_i^{(r)}(x) - \phi_i^{(r)}(y)| \geq (1-\varepsilon)r$. Thus, the concatenation of all these coordinates, $\phi^{(r)} = \oplus_i \phi_i^{(r)}$, satisfies the condition above.

Let $d_2 = 8\varepsilon^{-1} \log(\varepsilon^{-1})$. The final embedding $\phi : \mathcal{M} \to \Re^{d_2 d_1}$ is done by combining a weighted sum of $\phi^{(r)}$ as follows. Let $M_l(x)$ denote the matrix of size $d_2 \times d_1$, such that it is all zero, except the $(l \pmod{d_2})$th row, which is $M_l(x) = \phi^{((1+\varepsilon)^l)}(x)$. Then

$$\phi(x) = \sum_{l \in \mathbf{Z}} \psi_l(x) \quad \text{for} \quad \psi_l(x) = \frac{M_l(x)}{(1+\varepsilon)^{l/2}}.$$

To see that the embedding is a $1+O(\varepsilon)$-approximation of $\sqrt{d_{\mathcal{M}}}$, fix a pair of points $x, y \in \mathcal{M}$, and let $l_0 \in \mathbb{Z}$ such that $d_{\mathcal{M}}(x,y) \in [(1+\varepsilon)^{l_0+1}, (1+\varepsilon)^{l_0+2})$. Then in the relevant coordinates the ℓ_∞ distance between x and y is

$$\begin{aligned}
\left\| \sum_{k \in \mathbf{Z}} (\psi_{l_0+d_2 k}(x) - \psi_{l_0+d_2 k}(y)) \right\|_\infty &\geq \|\psi_{l_0}(x) - \psi_{l_0}(y)\|_\infty - \sum_{k<0} \|\psi_{l_0+d_2 k}(x) - \psi_{l_0+d_2 k}(y)\|_\infty \\
&\quad - \sum_{k>0} \|\psi_{l_0+d_2 k}(x) - \psi_{l_0+d_2 k}(y)\|_\infty \\
&\geq (1-\varepsilon)(1+\varepsilon)^{l_0/2} - \sum_{k<0} \frac{(1+\varepsilon)^{2+l_0+d_2 k}}{(1+\varepsilon)^{(l_0+d_2 k)/2}} - \sum_{k>0} \frac{(1+\varepsilon)^{2+l_0}}{(1+\varepsilon)^{(l_0+d_2 k)/2}} \\
&\geq (1-\varepsilon) \cdot (1+\varepsilon)^{l_0/2} - \varepsilon \cdot (1+\varepsilon)^{l_0/2} - \varepsilon \cdot (1+\varepsilon)^{l_0/2} \geq (1-O(\varepsilon)) \sqrt{d_{\mathcal{M}}(x,y)}.
\end{aligned}$$

On the other hand, for each $j \in \{0, \ldots, d_2 - 1\}$,

$$\begin{aligned}
&\left\| \sum_{k \in \mathbf{Z}} (\psi_{l_0+j+d_2 k}(x) - \psi_{l_0+j+d_2 k}(y)) \right\|_\infty \\
&\qquad \leq \sum_{k \leq 0} \|\psi_{l_0+j+d_2 k}(x) - \psi_{l_0+j+d_2 k}(y)\|_\infty + \sum_{k>0} \|\psi_{l_0+j+d_2 k}(x) - \psi_{l_0+j+d_2 k}(y)\|_\infty \\
&\qquad \leq \sum_{k \leq 0} \frac{(1+\varepsilon)^{2+l_0+j+d_2 k}}{(1+\varepsilon)^{(l_0+j+d_2 k)/2}} + \sum_{k>0} \frac{(1+\varepsilon)^{2+l_0}}{(1+\varepsilon)^{(l_0+j+d_2 k)/2}} = (1+O(\varepsilon)) \sqrt{d_{\mathcal{M}}(x,y)}.
\end{aligned}$$

Hence $\|\phi(x) - \phi(y)\|_\infty$ is a $1+O(\varepsilon)$-approximation to $\sqrt{d_{\mathcal{M}}(x,y)}$. ∎

The relevance of Assouad's embedding to compact representations is clear: Intuitively, $\phi(x)$ is short, and given $\phi(x)$ and $\phi(y)$, we can compute the square of the ℓ_∞ norm of the difference and obtain a $(1+\varepsilon)$-approximation to $d_{\mathcal{M}}(x,y)$. Note, however, that in order to be able to do this, we need to store $\Theta(\log(\Phi/\varepsilon))$ bits for each real number, which may require many words to be represented in our computation model (see Section 2). We solve this issue in Lemma 6.9 by reducing the problem for metrics with arbitrary spread to a set of similar problems on metrics with only polynomial spread, on which Assouad's embedding can be applied.

Lemma 6.7. *Given an n-point metric M with a polynomially bounded spread Φ and doubling dimension* dim*, an Assouad embedding (with parameter ε) of M can be computed in $\varepsilon^{-O(\dim)} n \log^2 n$ time.*

Proof: We follow closely the proof of Theorem 6.6. For each scale $(1+\varepsilon)^l$, we find in $O(n)$ time an $\varepsilon(1+\varepsilon)^l$-net $\mathcal{N}^{((1+\varepsilon)^l)}$ from the net-tree. We define a graph on this net: two points are connected by an edge if they are at a distance of at most $4(1+\varepsilon)^l$. This can be done in $\varepsilon^{-O(\dim)} n$ time using a variant of Rel() sets (basically, we compute sets like Rel() that contain points at a distance of at most $O(\varepsilon^{-1})$ times the current scale, instead of 13 times the current scale). We then partition $\mathcal{N}^{((1+\varepsilon)^l)}$ to color-classes using the greedy algorithm. Implemented with hashing, it works in expected $O(n)$ steps. Next, for each color-class we construct a $(1+\varepsilon/2)$-ANN data-structure, and thus we can compute a $(1+\varepsilon/2)$-approximation to $d_{\mathcal{M}}(x, C_i)$. Note that in the proof of Theorem 6.6, by enlarging the constants a little bit, a $(1+\varepsilon/2)$-approximation suffices. We repeat this construction for the $\log_{1+\varepsilon} \Phi$ levels in the metric. The rest of the embedding calculation is straightforward.

The running time of the algorithm is therefore $\varepsilon^{-O(\dim)} n \log n \log \Phi$. ∎

Remark 6.8. We believe that for $\varepsilon = 100$, a similar embedding can be constructed directly on the net-tree in $2^{O(\dim)}n$ time. The construction seems, however, much more complicated than the one described in Lemma 6.7. We have therefore decided that the slight gain in preprocessing time (overall, a factor of $\log n$, since the running time for constructing the net-tree is $2^{O(\dim)}n \log n$) is not worth the complications.

Lemma 6.9. *If there exists a $(\mathsf{P}, \mathsf{S}, \mathsf{Q}, \overline{\kappa})$-CRS $\mathcal{A}$ for every n-point metric with doubling dimension* dim *and spread $\leq 3(n/\varepsilon)^{12}$, and if P is concave, then there exists a $(P(4n) + 2^{O(\dim)}n \log n,\ S + O(n),\ Q + O(1),\ (1+\varepsilon)\overline{\kappa})$-CRS $\mathcal{B}$ for every n-point metric with doubling dimension* dim *(without any assumption on the spread).*

Proof: Denote by H the low quality HST of Section 3.2 which is a $3n^2$-approximation to the given metric $\mathcal{M}$.

Set $a_1 = 0$ and $a_2 = \lceil 5(\log(\varepsilon^{-1}) + \log_2 n)\rceil$. Apply the following procedure on H to obtain two HSTs, H_1 and H_2. Scan H from the top down. Retain the root, the leaves, and all internal vertices $u \in H$ with the following property: There exists $b > 0$ such that $\log_2 b \equiv a_i \pmod{\lceil 10(\log(\varepsilon^{-1}) + \log_2 n)\rceil}$ and $\Delta_{\overline{\mathrm{p}}(u)} > b \geq \Delta_u$. The HST H_i is constructed naturally on the retained vertices: A retained vertex u is connected to a parent v in H_i if v is the lowest retained ancestor of u in H.

Next, for each nonleaf vertex $u \in H_i$, $i \in \{1, 2\}$, denote by $C(u)$ the set of children of u. We observe that $R(C(u)) = \{\mathrm{rep}_u |\ u \in C(u)\}$ has a spread of at most $3(n/\varepsilon)^{12}$. To see this, note that $\mathrm{diam}(R(C(u))) \leq \Delta_u$, and, on the other hand, let b the largest real number such that $b < \Delta_u$ and $\log b \equiv a_i \pmod{\lceil 10(\log(\varepsilon^{-1}) \log_2 n)\rceil}$. Obviously $b \geq \Delta_u/(n/\varepsilon)^{10}$ and for every $x, y \in C(u)$, $\Delta_{\mathsf{lca}_H(x,y)} \geq b$, and therefore $d_{\mathcal{M}}(x, y) \geq b/(3n^2)$. Thus, for each internal vertex $u \in H_i$ we can construct a $\overline{\kappa}$-approximate CRS $\mathcal{A}$ to $R(C(u))$. The whole construction time is therefore $2^{O(\dim)}n \log n + \sum_k P(n_k) \leq 2^{O(\dim)}n \log n + P(4n)$.

We equip H, H_1, and H_2 with a data-structure for handling queries for lca and finding an ancestor at a given depth, both in constant time.

A distance query for the pair $x, y \in \mathcal{M}$ is processed as follows. Let $u_i = \mathsf{lca}_{H_i}(x, y)$. let x_i be a child of u_i which is an ancestor of x in H_i, and similarly y_i. Note that u_i, x_i, y_i can be computed in constant time using the lca and depth ancestor queries.

Further observe that $\exists i \in \{1, 2\}$ for which $\max\{\Delta_{x_i}, \Delta_{y_i}\} \leq \Delta_{\mathsf{lca}_H(x,y)}/(n/\varepsilon)^5$, and finding this i is an easy task.

We next query the CRS $\mathcal{A}$ of $R(C(u_i))$ for an approximation of $d_{\mathcal{M}}(\mathrm{rep}_{x_i}, \mathrm{rep}_{y_i})$. From the above we deduce that

$$\max\big\{d_{\mathcal{M}}(x, \mathrm{rep}_{x_i}), d_{\mathcal{M}}(y, \mathrm{rep}_{y_i})\big\} \leq \frac{3\varepsilon^5}{n^3} \cdot \frac{\Delta_{\mathsf{lca}_H(x,y)}}{3n^2} \leq \frac{3\varepsilon^5}{n^3} d_{\mathcal{M}}(x, y),$$

and therefore we have obtained a $\overline{\kappa}(1 + \varepsilon)$-approximation to $d_{\mathcal{M}}(x, y)$. ∎

Corollary 6.10. *Every n-point metric with doubling dimension* dim *has a $(\mathsf{P} = \varepsilon^{-O(\dim)}n \log^2 n,\ \mathsf{S} = \varepsilon^{-O(\dim)}n,\ \mathsf{Q} = \varepsilon^{-O(\dim)},\ \overline{\kappa} = 1 + \varepsilon)$-CRS.*

Proof: For the proof, combine Lemmas 6.9 and 6.7. ∎

Note that in Corollary 6.10 the query time depends on ε, in contrast to the claim in Theorem 6.1(a). This can be remedied using Lemma 6.4.

Proof: Proof of Theorem 6.1(a). Use the CRS of Corollary 6.10 *with constant* $\varepsilon_0 = 0.1$ as the bootstrapping CRS in Lemma 6.4. ∎

Proof: Proof of Theorem 6.1(b). In [GKL03], an alternative proof for the Assouad theorem is given with a much improved bound on the dimension of the host space: They prove that for any metric $(\mathcal{M}, d_{\mathcal{M}})$ with doubling dimension dim, it is possible to embed $(\mathcal{M}, d_{\mathcal{M}}^{1/2})$ in $\ell_\infty^{O(\dim)}$ with distortion $O(\dim^2)$.[4]

This embedding can be done in polynomial time. Using it as a replacement for Lemma 6.7, we therefore obtain the claimed CRS. ∎

Remark 6.11. The distortion of embedding into poly(dim) dimensional normed space cannot be improved below 1.9, since such an embedding gives a 1.9 approximate CRS which uses only $O(n\text{poly}(\dim)\log\phi)$ bits of storage with label length which are polynomially dependent on dim (see section 6.3), but Talwar [Tal04] have shown that such a CRS necessarily uses at least $n2^{\Omega(\dim)}$ bits, which is impossible for $\dim = \Omega(\log\log n)$. In this sense the embedding technique of [GKL03] cannot replace Assouad's original technique.

It is still open whether the construction time in Theorem 6.1(b) can be improved to near linear. The difficulty lies in the algorithmic version of the Lovász local lemma. As discussed in Remark 6.5, distortions as high as $2^{2^{O(\dim)}}$ are tolerable in this context.

6.2.1. Lower bound

We next argue that beating the $\Omega(\dim)$ query time using schema similar to the one presented above is unlikely.

For given reals $d_1, D, d_2 > 1$, we say that a (d_1, D, d_2)-*Assouad-type-scheme* (ATS) exists if there is a monotone increasing bijection $f : [0,\infty) \to [0,\infty)$, such that for all finite metric spaces $(P, d_{\mathcal{M}})$, with doubling dimension at most d_1, there exists $\phi : P \to \mathbb{R}^{d_2}_{\|\cdot\|_X}$, such that for $x, y \in P$, we have

$$\frac{d_{\mathcal{M}}(x,y)}{D} \leq f(\|\phi(x) - \phi(y)\|_X) \leq d_{\mathcal{M}}(x,y).$$

For example, the embedding of [GKL03] cited above is a $(d_1, O(d_1^2), O(d_1))$-ATS for any $d_1 > 1$, and it uses $f(x) = x^2$.

Proposition 6.12. *If $d_2 \leq d_1/5$, then for any $D > 1$, no (d_1, D, d_2)-ATS exists.*

Proof: The argument distinguishes between two essential cases: "Concave" function f cannot be used in any ATS since it causes a violation of the triangle inequality. For "convex" functions f we slightly generalize an argument from [BDG+05] that uses topological considerations (Borsuk–Ulam theorem) to conclude the impossibility.

Indeed, fix a (d_1, D, d_2)-ATS with a function f, where $d_2 \leq d_1/5$. Denote $g : [0,\infty) \to [0,\infty)$, where $g = f^{-1}$.

Suppose first that $\sup_{0<a\leq b<\infty} \frac{g(b)/b}{g(a)/a} = \infty$ ("concave f"). Fix a and b such that $0 < a < b < \infty$ and $\frac{g(b)/b}{g(a)/a} \geq 100D$. Let $n = \lceil 2Db/a \rceil$, and let P be the line metric on $\{0, \ldots, n\}$ such that $d_{\mathcal{M}}(i,j) = a|i-j|$. By the assumption, there exists $\phi : P \to \mathbb{R}^{d_1}_{\|\cdot\|_X}$ such that $\|\phi(i) - \phi(i+1)\|_X \leq g(d_{\mathcal{M}}(i, i+1)) = g(a)$, while on the other hand,

$$g(b) \leq g\left(\frac{\lceil 2Db/a \rceil\, a}{D}\right) = g\left(\frac{d_{\mathcal{M}}(0,n)}{D}\right) \leq \|\phi(0) - \phi(n)\|_X,$$

[4] If one wants to optimize the distortion using their technique, then it is possible to obtain $O(\dim)$ distortion when embedding into $\ell_p^{O(\dim\log\dim)}$.

since g is monotone increasing, as f is monotone increasing. Then by the triangle inequality,

$$g(b) \le \|\phi(0) - \phi(n)\|_X \le \sum_{i=1}^{n} \|\phi(i-1) - \phi(i)\|_X \le n\, g(a) \le 4D\, \frac{b\, g(a)}{a},$$

which implies that $\frac{g(b)/b}{g(a)/a} \le 4D$, which is a contradiction.

Next, assume that there exists $C > 1$ such that $\sup_{0<a\le b<\infty} \frac{g(b)\cdot a}{g(a)\cdot b} \le C$ ("convex f"). Then, for any $a \le b$ we have $\frac{g(b)a}{Cb} \le g(a)$. In particular, we have $\frac{g(d_{\mathcal{M}}(x,y))(d_{\mathcal{M}}(x,y)/D)}{C d_{\mathcal{M}}(x,y)} \le g(d_{\mathcal{M}}(x,y)/D)$. Namely,

$$\frac{g(d_{\mathcal{M}}(x,y))}{C \cdot D} \le g\left(\frac{d_{\mathcal{M}}(x,y)}{D}\right) \le \|\phi(x) - \phi(y)\|_X \le g(d_{\mathcal{M}}(x,y)).$$

Since $\|\cdot\|_X$ is d_2-dimensional, by John's theorem (see [Bar02, Chap. V]) it can be approximated by $\|\cdot\|_2$ up to a $\sqrt{d_2}$ factor. We thus have a $C' > 1$ such that for any d_1-dimensional finite metric $(P, d_{\mathcal{M}})$, there exists $\phi' : (P, d_{\mathcal{M}}) \to \Re^{d_2}_{\|\cdot\|_2}$ satisfying

$$\frac{g(d_{\mathcal{M}}(x,y))}{C'} \le \|\phi'(x) - \phi'(y)\|_2 \le g(d_{\mathcal{M}}(x,y)). \tag{1}$$

We next estimate how much $g \circ d_{\mathcal{M}}$ distorts $d_{\mathcal{M}}$ as a function of the spread of P. Assume that $\min_{x\ne y\in P} d_{\mathcal{M}}(x,y) = a_1$ and $\max_{x\ne y\in P} d_{\mathcal{M}}(x,y) = b_1$, that is, $\Phi(P) = b_1/a_1$. Then

$$\max_{a_1 \le t} \frac{g(t)}{t} = \frac{g(a_1)}{a_1} \cdot \max_{a_1 \le t} \frac{g(t)a_1}{t g(a_1)} \le C \frac{g(a_1)}{a_1},$$
$$\max_{s \le b_1} \frac{s}{g(s)} = \frac{b_1}{g(b_1)} \cdot \max_{s \le b_1} \frac{g(b_1)s}{b_1 g(s)} \le C \frac{b_1}{g(b_1)}.$$

Thus, considering the "distortion" of g, we have

$$\frac{\max_{x\ne y\in P} \frac{g(d_{\mathcal{M}}(x,y))}{d_{\mathcal{M}}(x,y)} \cdot \max_{x\ne y\in P} \frac{d_{\mathcal{M}}(x,y)}{g(d_{\mathcal{M}}(x,y))}}{\Phi(P)} \le \frac{C\frac{g(a_1)}{a_1} \cdot C\frac{b_1}{g(b_1)}}{b_1/a_1} = C^2 \frac{g(a_1)}{g(b_1)}.$$

As $g(0) = 0$ and $\lim_{x\to\infty} g(x) = \infty$, we conclude that this ratio tends to 0 as the spread $\Phi(P)$ tends to ∞. Combining it with (1), we conclude that for $\widehat{\phi} : (P, d_{\mathcal{M}}) \to \Re^{d_2}_{\|\cdot\|_2}$, defined as $\widehat{\phi}(x) = \phi'(x)$, we have $\mathrm{dist}(\widehat{\phi}) = o(\Phi(P))$. We will next show that this is impossible when P is a sufficiently dense net of $\mathbb{S}^{d_2}$.

Let $0 < \eta \le 0.1$. We take $P = P_\eta$ to be an η-net of $\mathbb{S}^{d_2}_{\|\cdot\|_2}$. The finite metric P_η has doubling dimension at most d_1. From the above we can embed $\phi' : P_\eta \to \Re^{d_2}_{\|\cdot\|_2}$ with distortion $o(\Phi(P)) = o(\eta^{-1})$. By scaling we may assume that this embedding is 1-Lipschitz. By Kirszbraun's theorem (see [BL00, Chap. 1]), the embedding ϕ' can be extended to the whole sphere $\widehat{\phi'} : \mathbb{S}^{d_2}_{\|\cdot\|_2} \to \mathbb{R}^{d_2}_{\|\cdot\|_2}$ without increasing the Lipschitz constant. The Borsuk–Ulam theorem (cf. [Mat03]) states that there exists $x \in \mathbb{S}^k$ such that $\widehat{\phi'}(x) = \widehat{\phi'}(-x)$. Note that $\exists y, z \in P_\eta$ such that $\|x - y\|_2 \le \eta$, and $\|(-x) - z\|_2 \le \eta$. Since $\widehat{\phi'}$ is 1-Lipschitz, we have

$$\|\phi'(y) - \phi'(z)\|_2 = \left\|\widehat{\phi'}(y) - \widehat{\phi'}(z)\right\|_2 \le \left\|\widehat{\phi'}(y) - \widehat{\phi'}(x)\right\|_2 + \left\|\widehat{\phi'}(-x) - \widehat{\phi'}(z)\right\|_2 \le 2\eta.$$

On the other hand, $\|y - z\|_2 \ge 1 - 2\eta$, which means that the Lipschitz constant of ϕ'^{-1}, and thus the distortion of ϕ', is at least $\Omega(\eta^{-1})$. This is a contradiction when η is a sufficiently small positive number, since we argued above that the distortion must be $o(\Phi(P)) = o(\eta^{-1})$. ∎

6.3. Distance labeling

An approximate distance labeling scheme (ADLS) seeks to compute for each point in the metric a short label such that given the labels of a pair of points, it is possible to compute efficiently an approximation of the pairwise distance. Thus, ADLS is a stricter notion of compact representation.[5] This notion was studied, for example, in [GPPR04, GKK+01, TZ01].

In the constant doubling dimension setting, Gupta et al. [GKL03] have shown a $(1+\varepsilon)$-embedding of the metric in $\ell_\infty^{O(\log n)}$. This implies a $(1+\varepsilon)$-ADLS with $O(\log n \log \Phi)$ bits for each label (the O notation here hides constants that depend on ε and dim). Talwar [Tal04] has shown an improved $(1+\varepsilon)$-ADLS with only $\varepsilon^{-O(\dim)} \log \Phi$ bits per label. Slivkins [Sli05b] has shown a $(1+\varepsilon)$-ADLS with $\varepsilon^{-O(\dim)} \log^2 n \log\log \Phi$ bits per label. Their techniques seem to be very different from each other.

Here we improve Slivkins' result and unify it with Talwar's result under the same technique.

Proposition 6.13. *Given a finite metric space, one can build a $(1+\varepsilon)$-ADLS with*

$$\min\left\{\varepsilon^{-O(\dim)} \log \Phi, \varepsilon^{-O(\dim)} \log n(\log n + \log\log \Phi)\right\}$$

bits per label.

Furthermore, there exist one-dimensional finite metric spaces of size n, and spread $\Phi \geq 2^{2n}$ for which any 1.9-ADLS requires labels of size $\Omega(\log n \log\log \Phi)$ bits per label.

Proof (sketch). First, labels of length $\varepsilon^{-O(\dim)} \log \Phi$ follow directly from Theorem 6.6: We have $\varepsilon^{-O(\dim)}$ coordinates, and, as discussed after the proof of Theorem 6.6, we need only $O(\log(\Phi/\varepsilon))$ bits of accuracy for each coordinate.

We next show a $(1+\varepsilon)$-ADLS using $\varepsilon^{-O(\dim)} \log n(\log n + \log\log \Phi)$ bits per label. We do so by presenting a "distributed implementation" of the data-structure used to prove Corollary 6.10. That data-structure consists of two trees (HSTs) H_1, H_2 on the same set of leaves: the points of the metric. Given two points x^1, x^2, we compute $u_i = \mathsf{lca}_{H_i}(x^1, x^2)$, and x_i^j is the ancestor of x^j in H_i which is the child of u_i. We then apply an Assouad embedding $A(x_i^j)$ that uses $O(\log n + \log\log \Phi + \log(\varepsilon^{-1}))$ bits. We define an identifier $I(v)$ of vertex $v \in H_i$ to be $A(v)$ concatenated with the Δ_v (encoded with $O(\log\log \Phi)$ bits). Hence, given two points x^1, x^2, using the identifiers $I(x_1^1)$, $I(x_1^2)$, $I(x_2^1)$, $I(x_2^2)$, $I(u_1)$, $I(u_2)$, we can compute a $(1+\varepsilon)$-approximation of $d_{\mathcal{M}}(x_1, x_2)$. We now use (the proof of) a result of Peleg [Pel04]: Given an n-vertex rooted tree with identifiers $I(v)$ of maximum length s on the vertices, it is possible to efficiently compute labels $L(v)$ of length $O(\log n(\log n + s))$ to the vertices, such that given $L(x)$ and $L(y)$ one can efficiently decode $I(u)$, where $u = \mathsf{lca}(x, y)$.

Unfortunately, we need a little bit more, namely, access to the children of u which are the ancestors of x and y. In order to obtain it we tinker with the construction of Peleg: In his Definition 3.2 from [Pel04], we extend the tuple $Q_i(v)$ to be

$$Q_i(v) = \Big\langle \langle i-1, I(\gamma_{i-1}(v))\rangle, \langle i, I(\gamma_i(v))\rangle, \langle i+1, I(\gamma_{i+1}(v))\rangle, \underline{\langle i, I(\mathrm{hs}(\gamma_i(v)))\rangle} \Big\rangle,$$

where $\mathrm{hs}(u)$ is the *heavy sibling* of u (the underlined part is our extension). By studying Peleg's construction, it is easy to verify that this extension suffices.

The above construction is asymptotically optimal in terms of n and Φ when $\Phi \geq 2^{2n}$, as we now prove. In [GPPR04] there is a family of n-vertex weighted rooted binary trees, such that any exact

[5] When comparing the storage of ADLSs to that of the CRSs from the previous sections, note that here we count *bits*, whereas in the rest of the paper we count *words* of length $O(\log n + \log\log \Phi + \log \varepsilon^{-1})$.

distance labeling scheme of the *leaves* requires labels of length $\Omega(\log n \log M)$ bits, where the edge weight is in the range $\{0, \ldots, M-1\}$. A further property of that family of trees is that the depth $h = M \log_2 n$ (i.e., the distance from the root) of all the leaves is the same. We next transform each tree T in that family into an HST H by giving every vertex v a label $2^{-\mathrm{depth}_T(v)}$. For any two leaves x and y, let $d_T(x,y) = 2(h + \log_2 d_H(x,y))$. Furthermore, even a 1.9 approximation of $d_H(x,y)$ allows us to recover the exact value of $d_H(x,y)$, since this value is an integral power of 2. Let us summarize: Given a 1.9 approximation of the distance in H allows us to obtain the exact distance in T. Therefore by setting $M = (\log_2 \Phi)/n$, it proves a lower bound of $\Omega(\log n \log\log \Phi)$ on the average label's length for a 1.9-ADLS for this family of HSTs. Since these HSTs are binary, their doubling dimension is 1. ∎

After a preliminary version of this paper appeared, Slivkins [Sli05a] managed to produce an ADLS with labels of length $\varepsilon^{-O(\dim)} \log n \log\log \Phi$, which improves upon our construction in the range $n^{\log\log n} \ll \Phi \ll 2^n$.

7. Doubling measure

A measure μ on a metric space $\mathcal{M}$ is called η-doubling if, for any $x \in \mathcal{M}$ and $r \geq 0$, $\mu(\mathbf{b}(x,2r)) \leq \eta \cdot \mu(\mathbf{b}(x,r))$. *Doubling measure* is already a useful notion in the analysis of metric spaces (see [Hei01]) and has recently been used in some algorithmic applications [Sli05b]. Vol'berg and Konyagin [VK87] proved that any compact λ-doubling metric space has a $\lambda^{O(1)}$-doubling measure. Wu's proof of this theorem [Wu98] can be implemented in linear time on the net-tree.

We assume that the net-tree T is already given. Denote by $\deg(v)$ the number of children of $v \in T$. Let $\gamma = \max_{v \in T} \deg(v)$ be the maximum degree in T. As we have seen before, $\gamma \leq 2^{O(\dim)}$. The probability measure μ is computed by calling $\mathsf{Partition}(\mathrm{root}, 1)$, where $\mathsf{Partition}$ is defined recursively as follows.

```
Partition(u ∈ T, p_u ∈ [0,1]).
    if u is a leaf then
        Set μ({rep_u}) ← p_u.
    else
        for each child v of u with rep_v ≠ rep_u do
            Set p_v ← p_u/γ.
            Call Partition(v, p_v).
        Let v_0 be the unique child of u such that rep_{v_0} = rep_u.
        Set p_{v_0} ← p_u(1 − (deg(u) − 1)/γ).
        Call Partition(v_0, p_{v_0}).
```

Proposition 7.1. *For any $u \in T$, we have $p_u = \mu(P_u)$.*

Proof: The proof is by straightforward induction on the height of T. ∎

Proposition 7.2. *Fix $l \in \mathbb{N}$, and two vertices u and v in T, such that $\max\{\ell(u), \ell(v)\} < l \leq \min\{\ell(\overline{\mathrm{p}}(u)), \ell(\overline{\mathrm{p}}(v))\}$ and $d_{\mathcal{M}}(\mathrm{rep}_u, \mathrm{rep}_v) \leq 40\tau^l$. Then $p_u \leq \gamma^{O(1)} p_v$.*

Proof: Denote $w = \mathsf{lca}_T(u,v)$, and denote by $w = u_0, u_1, \ldots, u_a = u$ the path in T from w to u, and by $w = v_0, v_1, \ldots, v_b = v$ the path in T from w to v.

We claim that for any $i \geq 1$, if $\ell(u_i) > l + 3$, then $\mathrm{rep}_{u_i} \neq \mathrm{rep}_{u_{i+1}}$. Indeed, otherwise

$$\begin{aligned} d_{\mathcal{M}}(\mathrm{rep}_{u_i}, \mathrm{rep}_v) &\leq d_{\mathcal{M}}(\mathrm{rep}_{u_{i+1}}, \mathrm{rep}_u) + d_{\mathcal{M}}(\mathrm{rep}_u, \mathrm{rep}_v) \\ &\leq \frac{2\tau}{\tau-1} \cdot \tau^{\ell(u_{i+1})} + 40\tau^l \leq \frac{2}{\tau-1}\tau^{\ell(u_i)} + 40\tau^{-4} \cdot \tau^{\ell(u_i)} \leq \frac{\tau^{\ell(u_i)}}{4}, \end{aligned}$$

but this is a contradiction to the packing property of Definition 2.1, since $v \notin P_{u_i}$ (note that for this argument to work, τ needs to be a large enough constant, say 11).

Next, we claim that for any $i \geq 1$ for which $\ell(u_i) > l+3$, $\ell(u_{i-1}) = \ell(u_i)+1$. Otherwise, $\ell(u_{i-1})-1 \geq \ell(u_i)+1$, implying

$$\begin{aligned} d_{\mathcal{M}}(\mathrm{rep}_{u_i}, \mathrm{rep}_v) &\leq d_{\mathcal{M}}(\mathrm{rep}_{u_{i+1}}, \mathrm{rep}_u) + d_{\mathcal{M}}(\mathrm{rep}_u, \mathrm{rep}_v) \leq \frac{2\tau}{\tau-1} \cdot \tau^{\ell(u_i)} + 40\tau^{-4} \cdot \tau^{\ell(u_i)} \\ &= \left(\frac{2\tau}{\tau(\tau-1)} + \frac{40}{\tau^5}\right) \cdot \tau^{\ell(u_i)+1} \leq \frac{\tau^{\ell(u_{i-1})-1}}{4} = \frac{\tau^{\ell(\overline{\mathrm{p}}(u_i))-1}}{4}, \end{aligned}$$

contradicting the packing property of Definition 2.1, since $v \notin P_{u_i}$.

Thus, the path between u and w is full, containing vertices on all levels, except maybe the last three levels. Furthermore, the representatives are different in each level. We therefore conclude that $p_u \leq p_w/\gamma^{\ell(w)-l-4}$. On the other hand, $p_v \geq p_w\gamma^{\ell(w)-l+1}$. Therefore $p_u \leq \gamma^5 p_v$. ∎

Theorem 7.3. *For any n-point metric space having doubling dimension* dim*, it is possible to construct a $2^{O(\dim)}$-doubling measure in $2^{O(\dim)} n \log n$ time.*

Proof: The running time of Partition is clearly linear and is dominated by the time to construct the net-tree.

We are left to prove that μ is a $\lambda^{O(1)}$-doubling measure. Let $x \in P$ and $r > 0$. Denote $N = \left\{ u \in T \;\middle|\; \ell(u) \leq \log_\tau(r/8) < \ell(\overline{\mathrm{p}}(u)) \right\}$. As we have seen in Proposition 2.2, the representatives of the vertices of N forms a net in the right scale. In particular, there exists $\widehat{x} \in N$ such that $d_{\mathcal{M}}(x, \mathrm{rep}_{\widehat{x}}) \leq 3r/8$ and $P_{\widehat{x}} \subset \mathbf{b}(\mathrm{rep}_{\widehat{x}}, 3r/8) \subset \mathbf{b}(x,r)$. Hence $p_{\widehat{x}} \leq \mu(\mathbf{b}(x,r))$. On the other hand, any two different representatives of vertices from N are at least $r/40$ separated, and therefore, for $X = N \cap \left\{ u \in T \;\middle|\; \mathrm{rep}_u \in \mathbf{b}(x,3r) \right\}$, we have $|X| \leq \lambda^{O(1)}$. Note that $\mathbf{b}(x,2r) \subset \cup_{u \in X} P_u$, and therefore

$$\mu(\mathbf{b}(x,2r)) \leq \sum_{u \in X} p_u \leq |X| \max_{u \in X} p_u.$$

By Proposition 7.2, $\max_{u \in X} p_u \leq \lambda^{O(1)} p_{\widehat{x}}$. We conclude that $\mu(\mathbf{b}(x,2r)) \leq \lambda^{O(1)} \mu(\mathbf{b}(x,r))$. ∎

We note in passing that algorithm Partition can be programmed in our computational model since every point gets at least a $2^{-O(n \log n)}$ measure, which can be easily represented in a floating-point word of length $O(\log n)$. Moreover, the algorithm has a "built in" mechanism to handle rounding error: instead of dividing by γ, we can divide by, say, 2γ, and now rounding errors are automatically offset in the measure given to v_0.

8. Lipschitz constant of mappings

Definition 8.1. A function $f : (P, \nu) \to (\mathcal{M}, \rho)$ is K-Lipschitz if for any $x, y \in P$, we have $\rho(f(x), f(y)) \leq K \cdot \nu(x, y)$.

A point $x \in P$ is K-Lipschitz if, for any $y \in P$, we have $\rho(f(x), f(y)) \leq K \cdot \nu(x, y)$.

Thus, given a set of points $P \subseteq \Re^d$, and a mapping $f : P \to \Re^{d'}$, it is natural to ask how quickly we can compute the Lipschitz constant for f on the set P, and more specifically, to compute it for every point of P.

8.1. The low-dimensional Euclidean case

Here, we consider a mapping $f : P \to (M, \rho)$, where $P \subseteq \Re$ is of size n, and (M, ρ) is an arbitrary metric space given as a matrix.

Proposition 8.2. *Computing the Lipschitz constant for f on P can be done in $O(n \log n)$ time.*

Proof: Indeed, let a, b, c be three numbers in P such that $a < b < c$. Observe that

$$\frac{\rho(f(c), f(a))}{c - a} \leq \frac{\rho(f(c), f(b)) + \rho(f(b), f(a))}{c - b + b - a} \leq \max\left(\frac{\rho(f(c), f(b))}{c - b} , \frac{\rho(f(b), f(a))}{b - a} \right),$$

since for any p, q, r, s positive numbers such that $p/q \leq r/s$, we have $p/q \leq (p+r)/(q+s) \leq r/s$. Thus, the Lipschitz constant is realized by a consecutive pair of points in P. We can therefore sort P and compute the slope for every consecutive pair. Clearly, the maximum is the Lipschitz constant of f. ∎

Proposition 8.3. *Let P be a set of n numbers on the real line, and let $f : P \to \Re$ be a given mapping. One can compute the Lipschitz constant of f on every point of P in $O(n \log^2 n)$ time.*

Proof: Consider the set $Q = \{(p, f(p)) | \ p \in P\}$. Let p be a point in P, and let L_p be the set of points of Q strictly to the left of p (according to the x-order), and let R_p be the set of points to its right. Denote by $\mathcal{CH}(A)$ the convex hull of $A \subset \Re^2$. If we know the tangents to $\mathcal{CH}(L_p)$ and $\mathcal{CH}(R_p)$ that pass through p, then we can compute the Lipschitz constant of p in constant time (i.e., it is the slope of the tangent with the largest slope).

Here, one can use the data-structure of Overmars and van Leeuwen [OvL81], which supports the maintenance of the convex hull under insertions, deletions, and tangent queries in $O(\log^2 n)$ per operation. Indeed, sort the points of P from left to right. Let $p_1, \ldots, p_n$ be the sorted points. Clearly, given $\mathcal{CH}(L_{p_i})$ and $\mathcal{CH}(R_{p_i})$ stored in the dynamic convex hull data-structure, we can compute $\mathcal{CH}(L_{p_{i+1}})$ and $\mathcal{CH}(R_{p_{i+1}})$ by deleting p_{i+1} from $\mathcal{CH}(R_{p_i})$ and inserting p_i into $\mathcal{CH}(L_{p_i})$. Thus, we can compute all the relevant convex hulls in $O(n \log^2 n)$ time. Furthermore, when we have $\mathcal{CH}(L_{p_i})$ and $\mathcal{CH}(R_{p_i})$, we perform tangent queries to compute the Lipschitz constant of p_i. Thus, the overall running time is $O(n \log^2 n)$. ∎

Theorem 8.4. *Given a set P of n points in the plane, and a mapping $f : P \to \Re$, one can compute the Lipschitz constant of f in $O(n \log^2 n)$ expected time.*

Proof: Assume that we know that f is K-Lipschitz on a set $Q \subseteq P$, and we would like to verify that it is K-Lipschitz on $\{q\} \cup Q$, where $q \in P \setminus Q$. This can be visualized as follows: From every point $p \in P$, there is an associated point in $\Re^3$, which is $\widehat{p} = (p_x, p_y, f(p))$. Being K-Lipschitz, as far as p is concerned, implies that q must lie below the upper cone of slope K emanating from $\widehat{p}$ and above the

lower cone of slope K emanating from $\widehat{p}$. Thus, if we collect all those upper cones, then q must lie below their lower envelope. However, since the upper cones all have the same slope, their lower envelope is no more than a (scaled) version of an additive weighted Voronoi diagram in the plane. Such a diagram can be computed in $O(n \log n)$ time for n points, and a point-location query in it can be performed in $O(\log n)$ time.

In fact, using the standard Bentley–Saxe technique [BS80], one can build a data-structure, where one can insert such upper cones in $O(\log^2 n)$ amortized time, given a query point q in the plane, and decide in $O(\log^2 n)$ time which of the cones inserted lies on the lower envelope vertically above q. Similar data-structures can be built for the upper envelope of the lower cones.

Thus, if we conjecture that the Lipschitz constant is K, then we can verify it for P in $O(n \log^2 n)$ time by inserting the points of P into the upper and lower envelope data-structure described above. However, let us assume that K is too small. Then, after inserting a subset Q of points into the data-structure, we will try to verify that the Lipschitz constant for a point $p \in P$ is K and fail. Then, it must be that the Lipschitz constant of f on $Q \cup \{p\}$ is realized by p. Thus, we can compute the Lipschitz constant of p in $Q \cup \{p\}$ in $O(|Q|)$ time, update our guess K, and rebuild the upper and lower data-structures for $Q \cup \{p\}$.

Of course, in the worst case, this would required $O(n^2 \log^2 n)$ running time (i.e., we would fail on every point). However, it is well known that if we randomly permute the points, and handle the points according to this ordering, then the value of the Lipschitz constant on every prefix would change $O(\log n)$ times in expectation. Thus, this would lead to $O(n \log^3 n)$ expected running time. Moreover, a slightly more careful analysis shows that the expected running time is $O(n \log^2 n)$. See [CS89] for details of such analysis. ∎

8.2. Constant doubling dimension to arbitrary metric

Theorem 8.5. *We are given a metric (P, ν) of n points having doubling dimension d, and a mapping $f : P \to (\mathcal{M}, \rho)$, where $\mathcal{M}$ is an arbitrary metric space. Then one can compute a $(1+\varepsilon)$-approximation of the Lipschitz constant of f in $n\varepsilon^{-O(d)} \log^2 n$ expected time.*

Proof: The algorithm is as follows:

1. Compute ε^{-1}-WSPD of P according to Section 5.
2. Set $K \leftarrow 0$.
3. For every pair $(A, B) \in \varepsilon^{-1}$-WSPD do:
   (a) Obtain *some* pair of points $a \in A$ and $b \in B$.
   (b) Compute $K \leftarrow \max\{K, \frac{\rho(f(a),f(b))}{\nu(a,b)}\}$.

Obviously the value K computed by the algorithm above is not larger than the Lipschitz constant of f. We next show that it is not much smaller. Let $x, y \in P$ be a pair in which f obtains its Lipschitz constant, i.e., $\frac{\rho(f(x),f(y))}{\nu(x,y)} = \max_{a \neq b} \frac{\rho(f(a),f(b))}{\nu(a,b)}$. Let $\{A, B\} \in$ WSPD be a pair such that $x \in A$, $y \in B$. Our algorithm chooses some pair $a \in A$, $b \in B$. Using the triangle inequality we have

$$\begin{aligned}\frac{\rho(f(a), f(b))}{\nu(a, b)} &\geq \frac{\rho(f(x), f(y)) - \operatorname{diam}(f(A)) - \operatorname{diam}(f(B))}{\nu(x, y) + \operatorname{diam}(A) + \operatorname{diam}(B)} \\ &\geq \frac{\rho(f(x), f(y)) - \operatorname{diam}(f(A)) - \operatorname{diam}(f(B))}{(1 + 2\varepsilon)\nu(x, y)}.\end{aligned}$$

If $\max\{\mathrm{diam}(f(A)), \mathrm{diam}(f(B))\} \leq \varepsilon \cdot \rho(f(x), f(y))$ then we conclude that

$$\frac{\rho(f(a), f(b))}{\nu(a,b)} \geq \frac{(1-2\varepsilon)\rho(f(x), f(y))}{(1+2\varepsilon)\nu(x,y)}$$

and we are done. Otherwise, assume that $\mathrm{diam}(f(A)) > \varepsilon \cdot \rho(f(x), f(y))$. Then there exists $f(a_1), f(a_2) \in f(A)$ for which $\rho(f(a_1), f(a_2)) > \varepsilon \cdot \rho(f(x), f(y))$, whereas

$$\nu(a_1, a_2) \leq \mathrm{diam}(A) \leq \varepsilon \cdot \nu(A, B) \leq \varepsilon \cdot \nu(x, y).$$

Thus,

$$\frac{\rho(f(a_1), f(a_2))}{\nu(a_1, a_2)} > \frac{\varepsilon \cdot \rho(f(x), f(y))}{\varepsilon \cdot \nu(x, y)},$$

which is a contradiction to the maximality of the pair $\{x, y\}$. ∎

9. Fast approximation of the doubling dimension

Theorem 9.1. *Given a metric space $\mathcal{M}$ with n points, one can approximate the doubling dimension* dim *of $\mathcal{M}$, up to a constant factor, in $2^{O(\dim)} n \log n$ expected time.*

Notice that this theorem, apart from its intrinsic interest, also removes the need to specify dim together with the input for the other algorithms in this paper.

The algorithm suggested in Theorem 9.1 naturally uses the net-tree.

Proposition 9.2. *Given a net-tree T of a metric $\mathcal{M}$, denote by λ_T the maximum out degree in T. Then $\log \lambda_T$ is a constant approximation to* $\dim(\mathcal{M})$.

Proof: Let $v \in T$ be the vertex with the maximum number of children λ_T. By Definition 2.1, any covering of $\mathbf{b}(\mathrm{rep}_v, \frac{2\tau}{\tau-1}\tau^{\ell(v)})$ by balls of radius $\frac{\tau-5}{4\tau(\tau-1)}$ requires at least λ_T such balls. This means that $\dim(\mathcal{M}) = \Omega(\log \lambda_T)$.

The upper bound $\dim(\mathcal{M}) = O(\log \lambda_T)$ follows easily from the arguments of Section 7. There, we actually prove the existence of a $\lambda_T^{O(1)}$-doubling measure in $\mathcal{M}$, and it is easy to prove that the existence of an α doubling measure in $\mathcal{M}$ implies that $\dim(\mathcal{M}) \leq \alpha$. ∎

Proof of Theorem 9.1. By Proposition 9.2 it is enough to show an implementation of the algorithm for constructing the net-tree that is oblivious to the doubling dimension of the metric. Checking the algorithm in Section 3, we observe that the algorithms in sections 3.1, 3.3, and 3.4 are indeed oblivious to the doubling dimension. We are therefore left with describing a doubling-dimension-oblivious algorithm for constructing HST that $O(n^2)$-approximates the given metric. More specifically, the only part that needs to be changed is the use of Lemma 2.4 in Lemma 3.5. To this end, instead of knowing λ, we "guess" the doubling constant to be 2^i, increasing i until we "succeed." More accurately, in the ith iteration, we apply the following sampling step 2^{3i} times: Pick randomly a point p from P, and compute the ball $\mathbf{b}(p, r)$ of smallest radius around p containing at least $n/(2 \cdot 2^{3i})$ points. Next, consider the ball of radius $\mathbf{b}(p, 2r)$. If it contains $\leq n/2$ points, the algorithm succeeded, and it stops. The algorithm is guaranteed to stop when $i \geq \lceil \log n \rceil$). Denote by $\delta = \delta(X)$ the random value, which is the value of 2^i when the algorithm stopped, when applied to a point set $X \subset \mathcal{M}$.

The resulting spanner is a $3n$-approximation *regardless* of the random bits, and thus the correctness of the net-tree algorithm is guaranteed. We need only argue about the expected running time for constructing the HST. The running time of the HST constructed is dominated by the spanner construction and the number of edges in it (see Lemma 3.5). Denote by λ the doubling constant of the metric $\mathcal{M}$.

Proposition 9.3. *For any $X \subseteq \mathcal{M}$,*

1. $\mathbf{E}[\delta(X)^{-3}] \geq \lambda^{-3}/16$.
2. $\mathbf{E}[\delta(X)^{3}] = O(\lambda^3)$.

Proof: Consider the algorithm above for computing $\delta(X)$. Once i reaches the value $k = \lceil \log_2 \lambda \rceil$, the probability of success on each point sampled is at least 2^{-3k} (by the argument in Lemma 2.4). Hence the probability of success in the ith round, $i \geq k$, conditioned on a failure in all previous rounds is at least $1 - (1-2^{-3k})^{2^{3i}}$, which means that

$$\mathbf{E}\big[\delta(X)^{-3}\big] \geq 1 - (1-2^{-3k})^{2^{3k}} 2^{-3k} \geq (1-1/e)\lambda^{-3}/8.$$

It also means that $\Pr[\delta \geq 2^{k+i}] \leq (1-2^{-3k})^{2^{3(k+i-1)}} \leq \exp\big(-\binom{i}{2}\big)$, and therefore

$$\mathbf{E}\big[\delta^3\big] = \sum_{t=1}^{\infty} \Pr[\delta^3 \geq t] \leq 2\lambda^3 + \sum_{t=2^{3k}}^{\infty} \exp\left(-\binom{\frac{\log t - 3k}{3}}{2}\right) \leq 2\lambda^3 + O(1).$$

■

We prove only an upper bound on the running time. Bounding the number of edges is similar. Denote by $f(X)$ the running time of the algorithm when applied to $X \subseteq \mathcal{M}$, and let $g(X) = \mathbf{E}[f(X)]$ and $g(n) = \sup_{X \subseteq \mathcal{M},\, |X|=n} g(X)$.

The spanner construction algorithm of Lemma 3.5 satisfies

$$g(X) \leq \mathbf{E}\left[\max_{\delta(X)^{-3} \leq \alpha \leq 1/2} \big(g(\alpha|X|) + g((1-\alpha)|X|) + c'\delta(X)^3 n\big)\right]. \tag{2}$$

We now prove by induction that $g(n) \leq c\lambda^3 n \ln n$ for some $c > 0$. Fix Y to be a subset of $\mathcal{M}$ of size n such that $g(Y) = g(n)$. We have

$$\begin{aligned}
g(n) &\leq \mathbf{E}\left[\max_{\delta(Y)^{-3} \leq \alpha \leq 1/2} \big(\mathbf{E}[g(\alpha|Y|)] + \mathbf{E}[g((1-\alpha)|Y|)] + c'\delta^3 n\big)\right] \\
&\leq \mathbf{E}\left[\max_{\delta(Y)^{-3} \leq \alpha \leq 1/2} \big(c\lambda^3 \alpha|Y|\ln(\alpha|Y|) + c\lambda^3(1-\alpha)|Y|\ln((1-\alpha)|Y|) + c'\delta^3 n\big)\right] \\
&\leq \mathbf{E}\big[c\lambda^3\delta^{-3} n \ln\big(\delta^{-3} n\big) + c\lambda^3(1-\delta^{-3}) n \ln\big(\big(1-\delta^{-3}\big)\, n\big) + c'\delta^3 n\big] \\
&\leq c\lambda^3 n \cdot \mathbf{E}\big[\delta^{-3}\ln\big(\delta^{-3} n\big) + (1-\delta^{-3})\ln\big(\big(1-\delta^{-3}\big)\, n\big)\big] + (c''\lambda^3 + d')n \\
&\leq c\lambda^3 n \cdot \mathbf{E}\big[\ln\big(\big(1-\delta^{-3}\big)\, n\big)\big] + (c''\lambda^3 + d')n \\
&\leq c\lambda^3 n \ln n + c\lambda^3 n \cdot \mathbf{E}\big[\ln(1-\delta^{-3})\big] + (c''\lambda^3 + d')n \\
&\leq c\lambda^3 n \ln n - c\lambda^3 n \cdot \mathbf{E}\big[\delta^{-3}\big] + (c''\lambda^3 + d')n \\
&\leq c\lambda^3 n \ln n - c\lambda^3 n \cdot \big(\lambda^{-3}/16\big) + (c''\lambda^3 + d')n \\
&\leq c\lambda^3 n \ln n,
\end{aligned}$$

since $\ln(1-\delta^{-3}) \leq -\delta^{-3}$, by Proposition 9.3, and for $c > 0$ large enough. ■

10. Concluding remarks

In this paper, we show how to efficiently construct hierarchical nets for finite spaces with low doubling dimension and use this construction in several applications. We believe that this result will have further applications.

Among other things, our fast construction of WSPD implies a near linear time construction of an approximate minimum spanning tree of the space. Our fast construction of net-tree implies that one can do 2-approximate k-center clustering in $O(n \log n)$ expected time.

Further, transfer of problems and techniques from low-dimensional Euclidean space to low-dimensional metrics seems to be interesting. A plausible example of such problems is the construction of $(1 + \varepsilon)$-spanners with some additional properties (such as low total weight or small hop-diameter). Results of this flavor exist in low-dimensional Euclidean spaces.

It is easy to verify that, for a general metric, no HST can be constructed without inspecting all $\binom{n}{2}$ edges. Indeed, consider the uniform metric over n points, and change in an adversarial fashion a single edge to have length 0.

10.1. All nearest neighbors

The *all nearest neighbor problem* is to compute for a set P of n points the (exact) nearest neighbor for each point of $p \in P$ in the set $P \setminus \{p\}$. It is known that in low-dimensional Euclidean space this can be done in $O(n \log n)$ time [Cla83, Vai89, CK95]. One can ask if a similar result can be attained for finite metric spaces with low doubling dimensions. Below we show that this is impossible.

Consider the points $p_1, \ldots, p_n$, where the distance between p_i and p_j, for $i < j$, is either 2^j or $2^j + \varepsilon$, for $\varepsilon < 0.1$. It is easy to verify that this metric has doubling constant at most three. We now show that for any deterministic algorithm for computing all nearest neighbors, there is a metric in the family of the metrics described above for which the algorithm performs $\binom{n}{2}$ distance queries.

This claim is proved using an adversarial argument: When the adversary is queried about the distance between p_i and p_j, for $i < j$, if not all the distances between $p_1, \ldots, p_{j-1}$ and p_j were specified, the adversary will always return the distance to be $2^j + \varepsilon$. The distances 2^j would be returned only for the last pair among the $j - 1$ pairs in this set. In particular, for the algorithm to know what is the closest point to p_j, it must perform $j - 1$ queries. Thus, overall, an algorithm doing all nearest neighbors for $p_1, \ldots, p_n$ will have to perform $\binom{n}{2}$ queries.

A similar asymptotic lower bound can be proved for randomized algorithms using Yao's principle (here the adversary selects for each j one index $i_j < j$ at random for which $d(p_{i_j}, p_j) = 2^j$, and for the rest of $i \neq i_j$, $i < j$, $d(p_i, j) = 2^j + \varepsilon$).

At this point, it is natural to ask whether one can achieve running time of $O(n \log(n\Phi(P)))$ for the all nearest neighbor problem. This, however, is straightforward. Indeed, compute a 4-WSPD of P. Clearly, if q is a nearest neighbor for p, then there is a pair in the WSPD such that p is the only point on one side, and the other side contains q. Thus, we scan all such unbalanced pairs (one point on one side, and many points on the other side) and compute the nearest neighbor for each point. Thus, this computes all nearest neighbors. As for the running time analysis, consider all such pairs in distance range l to $2l$, and observe that by a packing argument, for any node u in the net-tree, the number of such WSPD pairs with u in them is $2^{O(\dim)}$. In fact, along a path in the net-tree, only a constant number of nodes might participate in such pairs. Thus, every point is being scanned $2^{O(\dim)}$ times, implying that scanning all such pairs takes $2^{O(\dim)}n$ time. There are $\lceil \lg(\Phi(P)) \rceil$ resolutions, so the overall running time is $2^{O(\dim)}n \log(n\Phi(P))$.

Acknowledgments

We thank James Lee, Ken Clarkson, and Alex Slivkins for helpful correspondence regarding the content of this paper. James Lee pointed us to some recent related papers and suggested the construction of a doubling measure as an application of the net-tree. The observations about the all nearest neighbors problem rose from discussions with Ken Clarkson. Alex Slivkins pointed out to us an error in a preliminary version of this paper. Finally, the authors would like to thank the anonymous referees for their useful comments.

References

[AMN+98] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. Assoc. Comput. Mach.*, 45(6), 1998.

[Ass83] P. Assouad. Plongements lipschitziens dans $\mathbf{R}^n$. *Bull. Soc. Math. France*, 111(4):429–448, 1983.

[Bar02] A. Barvinok. *A course in convexity*, volume 54 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2002.

[BDG+05] M. Bădoiu, K. Dhamdhere, A. Gupta, Y. Rabinovich, H. Räcke, and R. Ravi. Approximation algorithms for embedding into low-dimensional spaces. In *Proc. 16th ACM-SIAM Sympos. Discrete Algorithms*, 2005.

[BF00] M. A. Bender and M. Farach-Colton. The lca problem revisited. In *Proc. 4th Latin Amer. Symp. on Theor. Info.*, pages 88–94. Springer-Verlag, 2000.

[BFC04] M. A. Bender and M. Farach-Colton. The level ancestor problem simplified. *Theo. Comp. Sci.*, 321(1):5–12, 2004.

[BKL04] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor, 2004.

[BL00] Y. Benyamini and J. Lindenstrauss. *Geometric nonlinear functional analysis*. AMS, 2000.

[BS80] J. L. Bentley and J. B. Saxe. Decomposable searching problems i: Static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.

[Cal95] P. B. Callahan. The well separated pair decomposition and its application. Technical report, Johns Hopkins U., 1995.

[CGMZ05] H. T-H Chan, A. Gupta, B. M. Maggs, and S. Zhou. On hierarchical routing in bounded growth metrics. In *Proc. 16th ACM-SIAM Sympos. Discrete Algorithms*, 2005.

[CK95] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. Assoc. Comput. Mach.*, 42:67–90, 1995.

[Cla83] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 226–232, 1983.

[Cla99] K. L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete Comput. Geom.*, 22(1):63–93, 1999.

[Cla02] K. L. Clarkson. Nearest neighbor searching in metric spaces: Experimental results for sb(s). Available from `http://cm.bell-labs.com/who/clarkson/Msb/white_paper.pdf`, 2002.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[FG88] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 434–444, 1988.

[GKK+01] C. Gavoille, M. Katz, N. A. Katz, C. Paul, and D. Peleg. Approximate distance labeling schemes. In *Proc. 9th Annu. European Sympos. Algorithms*, volume 2161 of *Lect. Notes in Comp. Sci.*, pages 476–487. Springer, 2001.

[GKL03] A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Proc. 44th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 534–543, 2003.

[GLNS02a] J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. Smid. Approximate distance oracles for geometric graphs. In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*, pages 828–837. Society for Industrial and Applied Mathematics, 2002.

[GLNS02b] J. Gudmundsson, C. Levcopoulos, G. Narasimhan, and M. H. M. Smid. Approximate distance oracles revisited. In *Proc. 13th Annu. Internat. Sympos. Algorithms Comput.*, pages 357–368. Springer-Verlag, 2002.

[Gon85] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.

[GPPR04] C. Gavoille, D. Peleg, S. Perennes, and R. Raz. Distance labeling in graphs. *J. Algorithms*, 53(1):85–112, 2004.

[Har01] S. Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 94–103, 2001.

[Har04] S. Har-Peled. Clustering motion. *Discrete Comput. Geom.*, 31(4):545–565, 2004.

[Hei01] J. Heinonen. *Lectures on analysis on metric spaces*. Universitext. Springer-Verlag, New York, 2001.

[HKMR04] K. Hildrum, J. Kubiatowicz, S. Ma, and S. Rao. A note on the nearest neighbor in growth-restricted metrics. In *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms*, pages 560–561. Society for Industrial and Applied Mathematics, 2004.

[HM04] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-median clustering and their applications. In *Proc. 36th Annu. ACM Sympos. Theory Comput.*, pages 291–300, 2004.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 604–613, 1998.

[Kei88] J. M. Keil. Approximating the complete Euclidean graph. In *Proc. 1st scand. Worksh. Algorithm Theory*, number 318 in LNCS, pages 208–213. Springer, 1988.

[KL04a] R. Krauthgamer and J. R. Lee. The black-box complexity of nearest neighbor search. In *Proc. 31st Internat. Colloq. Automata Lang. Prog.*, pages 858–869, 2004.

[KL04b] R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 798–807. Society for Industrial and Applied Mathematics, 2004.

[Knu97] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley, third edition edition, 1997.

[KR02] D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proc. 34th Annu. ACM Sympos. Theory Comput.*, pages 741–750. ACM Press, 2002.

[Laa02] T. J. Laakso. Plane with A_∞-weighted metric not bi-Lipschitz embeddable to $\mathbb{R}^N$. *Bull. London Math. Soc.*, 34(6):667–676, 2002.

[LS98] J. Luukkainen and E. Saksman. Every complete doubling metric space carries a doubling measure. *Proc. Amer. Math. Soc.*, 126(2):531–534, 1998.

[Mat03] J. Matoušek. *Using the Borsuk-Ulam theorem*. Universitext. Springer-Verlag, Berlin, 2003. Lectures on topological methods in combinatorics and geometry, Written in cooperation with Anders Björner and Günter M. Ziegler.

[OvL81] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[Pel04] D. Peleg. Informative labeling schemes for graphs. *Theoretical Computer Science*, 2004. To appear. Preliminary version appeared in MFCS '00.

[PS89] D. Peleg and A. Schäffer. Graph spanners. *J. Graph Theory*, 13:99–116, 1989.

[Sem96] S. Semmes. On the nonexistence of bi-Lipschitz parameterizations and geometric problems about A_∞-weights. *Rev. Mat. Iberoamericana*, 12(2):337–410, 1996.

[Sli05a] A. Slivkins. Distance estimation and object location via rings of neighbors. Technical Report TR2005-1977, Cornell CIS, 2005.

[Sli05b] A. Slivkins. Distributed approaches to triangulation and embedding. In *Proc. 16th ACM-SIAM Sympos. Discrete Algorithms*, 2005.

[Tal04] K. Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proc. 36th Annu. ACM Sympos. Theory Comput.*, pages 281–290. ACM Press, 2004.

[TZ01] M. Thorup and U. Zwick. Approximate distance oracles. In *Proc. 33rd Annu. ACM Sympos. Theory Comput.*, pages 183–192. ACM Press, 2001.

[Vai89] P. M. Vaidya. An $o(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.

[VK87] A. L. Vol′berg and S. V. Konyagin. On measures with the doubling condition. *Izv. Akad. Nauk SSSR Ser. Mat.*, 51(3):666–675, 1987.

[Wu98] J.-M. Wu. Hausdorff dimension and doubling measures on metric spaces. *Proc. Amer. Math. Soc.*, 126(5):1453–1459, 1998.

A. Proof of Lemma 3.9

Notice that Lemma 3.7 implies that $(\overline{r}_i)_{i\geq 1}$ is a monotone nonincreasing sequence and that $r_i \geq \overline{r}_i \geq r_i/(1+n^{-2}) \geq \frac{4}{5}r_i$.

Proof of Lemma 3.9. We prove by induction on k all five assertions together. The base case is obvious. Assume by the induction hypothesis that $T^{(k-1)}$ satisfies all the properties above. We prove it for $T^{(k)}$.

Property (i). Every point inserted during the lth phase (i.e., a point p_i for which $\lceil \log_\tau \overline{r}_i \rceil = l$) must have its current parent (in $T^{(k)}$) at level l. Thus, if $\ell(\widehat{u}) > l$, this means that c_{p_k} was inserted before the current phase, which means that it is indeed the closest point to p_k among $\{p_1, \ldots, p_h\}$. Otherwise, if $\ell(\widehat{u}) = l$, then

$$\begin{aligned} d_{\mathcal{M}}(\widehat{u}, q) &\leq d_{\mathcal{M}}(\widehat{u}, c_{p_k}) + d_{\mathcal{M}}(c_{p_k}, p_k) + d_{\mathcal{M}}(p_k, q) \\ &\leq 2 \cdot \tau^l + (1+n^{-2})\tau^l + (1+n^{-2})\tau^l \leq 13 \cdot \tau^l. \end{aligned}$$

Since q appears before the level l began, either $\ell(\overline{\mathrm{p}}(q)) > l$ and then $q \in \overline{\mathrm{Rel}}(\widehat{u})$, or $\ell(\overline{\mathrm{p}}(q)) = l$; but then it must be that $\mathrm{rep}_{\overline{\mathrm{p}}(q)} = q$, so $\overline{\mathrm{p}}(q) \in \overline{\mathrm{Rel}}(\widehat{u})$. Either case q is a representative of a vertex in $\overline{\mathrm{Rel}}(\widehat{u})$ which is the same as $\mathrm{Rel}(\widehat{u})$ (in $T^{(k-1)}$).

Property (ii). We shall prove it for both p_k and the new internal vertex (in case (a) of the construction). Consider first case (a) of the construction:

$$d_{\mathcal{M}}(\mathrm{rep}_u, \mathrm{rep}_v) = d_{\mathcal{M}}(\mathrm{rep}_u, q) \leq \tau^{\ell(u)},$$

where the last inequality follows from the induction hypothesis. Also,

$$d_{\mathcal{M}}(\mathrm{rep}_v, p_k) \leq 2 \cdot \tau^{\ell(v)},$$

and we are done with the first case of the construction.

Case (b) of the construction follows from the definition of q and since, as argued above, for $u = \overline{\mathrm{p}}(q)$, $\mathrm{rep}_u = q$.

Property (iii). Fix some $t \in \Re$, and let x and y be two vertices for which $\max\{\ell(x), \ell(y)\} < t \leq \min\{\ell(\overline{\mathrm{p}}(x)), \ell(\overline{\mathrm{p}}(y))\}$. If both x and y are not p_k, then the claim follows from the inductive hypothesis (even for the newly formed internal vertex, since it inherits its parent and representative from a previously established vertex). Otherwise, assume $x = p_k$. As p_k is the latest addition of a leaf to T, $d_{\mathcal{M}}(p_k, \mathrm{rep}_y) \geq \overline{r}_{k-1} \geq \tau^{l-1}$. Note that $\ell(\overline{\mathrm{p}}(p_k)) = l$, so $t \leq l$, and we conclude that $d_{\mathcal{M}}(\mathrm{rep}_x, \mathrm{rep}_y) \geq \tau^{t-1}$.

Property (iv). We next prove that $T^{(k)}$ is a net-tree. The only nonstraightforward claims are the packing and covering properties. The covering property follows from Property (ii) of this lemma: Let

$u = u_1$ be a vertex, $v = u_m$ a descendant, and $\langle u_1, \ldots, u_m \rangle$ the path between them in T; then

$$\begin{aligned} d_{\mathcal{M}}(\mathrm{rep}_u, \mathrm{rep}_v) &\leq \sum_{i=1}^{m-1} d_{\mathcal{M}}(\mathrm{rep}_{u_i}, \mathrm{rep}_{u_{i+1}}) \leq 2 \sum_{i=1}^{m-1} \tau^{\ell(u_i)} \\ &\leq 2 \sum_{i=1}^{m-1} \tau^{\ell(u_1)-(i-1)} \leq \frac{2\tau}{\tau - 1} \cdot \tau^{\ell(u)}. \end{aligned}$$

The packing property is more delicate. Let w be an arbitrary vertex in $T^{(k)}$, and let $x \notin P_w$ be a point. We want to prove that $d_{\mathcal{M}}(x, \mathrm{rep}_w) \geq \frac{\tau-5}{2(\tau-1)} \tau^{\ell(\overline{\mathrm{p}}(w))-1}$. Let $\widehat{x} \in T^{(k)}$ be an ancestor of x such that $\ell(\widehat{x}) \leq \ell(\overline{\mathrm{p}}(w)) - 1 < \ell(\overline{\mathrm{p}}(\widehat{x}))$. Applying Property (iii) with $t = \ell(\overline{\mathrm{p}}(w))$, we get that $d_{\mathcal{M}}(\mathrm{rep}_{\widehat{x}}, \mathrm{rep}_w) \geq \tau^{\ell(\overline{\mathrm{p}}(w))-1}$.

If $x = \widehat{x}$, we are done. Otherwise, if $\ell(\widehat{x}) < \ell(\overline{\mathrm{p}}(w)) - 1$, then, by Property (ii), we have

$$\begin{aligned} d_{\mathcal{M}}(\mathrm{rep}_w, x) &\geq d_{\mathcal{M}}(\mathrm{rep}_w, \mathrm{rep}_{\widehat{x}}) - d_{\mathcal{M}}(\mathrm{rep}_{\widehat{x}}, x) \\ &\geq \tau^{\ell(\overline{\mathrm{p}}(w))-1} - \frac{2\tau}{\tau - 1} \cdot \tau^{\ell(\overline{\mathrm{p}}(w))-2} = \frac{\tau - 3}{\tau - 1} \cdot \tau^{\ell(\overline{\mathrm{p}}(w))-1}, \end{aligned}$$

and we are done.

Otherwise, let $\bar{x} \in T^{(k)}$ be an ancestor of x which is the child of $\widehat{x}$ ($\overline{\mathrm{p}}(\bar{x}) = \widehat{x}$). If $\mathrm{rep}_{\bar{x}} = \mathrm{rep}_{\widehat{x}}$, then the preceding argument (where $\ell(\widehat{x}) < \ell(\overline{\mathrm{p}}(w)) - 1$) also applies here, and we are done.

Otherwise, we get the following situation: $\ell(\overline{\mathrm{p}}(\bar{x})) = \ell(\overline{\mathrm{p}}(w)) - 1$ and $\mathrm{rep}_{\bar{x}} \neq \mathrm{rep}_{\overline{\mathrm{p}}(\bar{x})}$. But this can happen only if $\bar{x}$ was inserted during level $\ell(\overline{\mathrm{p}}(w)) - 1$. Recall that the algorithm connects $\mathrm{rep}_{\bar{x}}$ as a child of a vertex in level $\ell(\overline{\mathrm{p}}(w)) - 1$ whose representative is the closest point among those appearing during the levels greater than $\ell(\overline{\mathrm{p}}(w)) - 1$. Note that both $\mathrm{rep}_{\overline{\mathrm{p}}(\bar{x})}$ and rep_w are inserted in a level greater than $\ell(\overline{\mathrm{p}}(w)) - 1$. We conclude that $d_{\mathcal{M}}(\mathrm{rep}_{\bar{x}}, \mathrm{rep}_{\overline{\mathrm{p}}(\bar{x})}) \leq d_{\mathcal{M}}(\mathrm{rep}_{\bar{x}}, \mathrm{rep}_w)$. Therefore

$$\begin{aligned} d_{\mathcal{M}}(\mathrm{rep}_{\bar{x}}, \mathrm{rep}_w) \geq \max\bigl\{ & d_{\mathcal{M}}(\mathrm{rep}_{\bar{x}}, \mathrm{rep}_{\overline{\mathrm{p}}(\bar{x})}), d_{\mathcal{M}}(\mathrm{rep}_w, \mathrm{rep}_{\overline{\mathrm{p}}(\bar{x})}) - d_{\mathcal{M}}(\mathrm{rep}_{\bar{x}}, \mathrm{rep}_{\overline{\mathrm{p}}(\bar{x})}) \bigr\} \\ &\geq \frac{d_{\mathcal{M}}(\mathrm{rep}_w, \mathrm{rep}_{\overline{\mathrm{p}}(\bar{x})})}{2} \geq 0.5 \cdot \tau^{\ell(\overline{\mathrm{p}}(w))-1}. \end{aligned}$$

Hence, by the covering property,

$$\begin{aligned} d_{\mathcal{M}}(x, \mathrm{rep}_w) &\geq d_{\mathcal{M}}(\mathrm{rep}_{\bar{x}}, \mathrm{rep}_w) - d_{\mathcal{M}}(\mathrm{rep}_{\bar{x}}, x) \geq 0.5 \cdot \tau^{\ell(\overline{\mathrm{p}}(w))-1} - \frac{2\tau}{\tau - 1} \cdot \tau^{\ell(\overline{\mathrm{p}}(w))-2} \\ &= \frac{\tau - 5}{2(\tau - 1)} \cdot \tau^{\ell(\overline{\mathrm{p}}(w))-1}, \end{aligned}$$

and we are done.

Property (v). Assume that a new vertex x is attached as a child to a vertex y. We shall prove that our traversing algorithm visits all vertices w for which either $w \in \overline{\mathrm{Rel}}(x)$ or $x \in \overline{\mathrm{Rel}}(w)$. Suppose first that $x \in \overline{\mathrm{Rel}}(w)$. Thus, $\ell(w) < \ell(y)$. Let z be an ancestor of w for which $\ell(z) \leq \ell(y) < \ell(\overline{\mathrm{p}}(z))$. Let $\langle z = z_1, \ldots, z_m = w \rangle$ be the path between them in T. Then, for any $1 \leq i \leq m - 1$, it holds that

$$\begin{aligned} d_{\mathcal{M}}(\mathrm{rep}_x, \mathrm{rep}_{z_i}) &\leq d_{\mathcal{M}}(\mathrm{rep}_x, \mathrm{rep}_w) + d_{\mathcal{M}}(\mathrm{rep}_{z_i}, \mathrm{rep}_w) \\ &\leq 13 \cdot \tau^{\ell(z_m)} + \frac{2\tau}{\tau - 1} \cdot \tau^{\ell(z_i)} \leq 13 \cdot \tau^{\ell(z_i)}. \end{aligned}$$

Thus $x \in \overline{\mathrm{Rel}}(z_i)$ for any $2 \leq i \leq m$. Thus, if $z = z_1 \in \mathrm{Rel}(y)$, we are assured that $w = z_m$ will be visited. Indeed, $z \in \overline{\mathrm{Rel}}(y)$, since

$$\begin{aligned} d_{\mathcal{M}}(\mathrm{rep}_y, \mathrm{rep}_z) &\leq d_{\mathcal{M}}(\mathrm{rep}_y, \mathrm{rep}_x) + d_{\mathcal{M}}(\mathrm{rep}_x, \mathrm{rep}_w) + d_{\mathcal{M}}(\mathrm{rep}_w, \mathrm{rep}_z) \\ &\leq 2 \cdot \tau^{\ell(y)} + 13 \cdot \tau^{\ell(w)} + \frac{2\tau}{\tau - 1} \cdot \tau^{\ell(z)} \\ &\leq 2 \cdot \tau^{\ell(y)} + 13 \cdot \tau^{\ell(y)-1} + \frac{2\tau}{\tau - 1} \cdot \tau^{\ell(y)} \\ &\leq 13 \cdot \tau^{\ell(y)}. \end{aligned}$$

This means that $z \in \mathrm{Rel}(y)$ by the inductive hypothesis.

Next, we consider the case when $w \in \overline{\mathrm{Rel}}(x)$. In this case, $\ell(w) \leq \ell(x) < \ell(\overline{\mathrm{p}}(w))$ and

$$d_{\mathcal{M}}(\mathrm{rep}_w, \mathrm{rep}_y) \leq d_{\mathcal{M}}(\mathrm{rep}_w, \mathrm{rep}_x) + d_{\mathcal{M}}(\mathrm{rep}_x, \mathrm{rep}_y) \leq 13 \cdot \tau^{\ell(x)} + 2 \cdot \tau^{\ell(y)} \leq 13 \cdot \tau^{\ell(y)}.$$

Hence, if $\ell(\overline{\mathrm{p}}(w)) > \ell(y)$, then $w \in \overline{\mathrm{Rel}}(y)$ which implies that $w \in \mathrm{Rel}(y)$ by the inductive hypothesis, and we are done.

If $\ell(\overline{\mathrm{p}}(w)) = \ell(y)$, then

$$\begin{aligned} d_{\mathcal{M}}(\mathrm{rep}_{\overline{\mathrm{p}}(w)}, \mathrm{rep}_y) &\leq d_{\mathcal{M}}(\mathrm{rep}_{\overline{\mathrm{p}}(w)}, \mathrm{rep}_w) + d_{\mathcal{M}}(\mathrm{rep}_w, \mathrm{rep}_x) + d_{\mathcal{M}}(\mathrm{rep}_x, \mathrm{rep}_y) \\ &\leq 2 \cdot \tau^{\ell(y)} + 13 \cdot \tau^{\ell(y)-1} + 2 \cdot \tau^{\ell(y)} \leq 13 \cdot \tau^{\ell(y)}. \end{aligned}$$

Thus, in this case $\overline{\mathrm{p}}(w) \in \overline{\mathrm{Rel}}(y)$, and using the inductive hypothesis, we are done.

We are left with the case $\ell(\overline{\mathrm{p}}(w)) < \ell(y)$. In this case

$$\begin{aligned} d_{\mathcal{M}}(\mathrm{rep}_{\overline{\mathrm{p}}(w)}, \mathrm{rep}_x) &\leq d_{\mathcal{M}}(\mathrm{rep}_{\overline{\mathrm{p}}(w)}, \mathrm{rep}_w) + d_{\mathcal{M}}(\mathrm{rep}_w, \mathrm{rep}_x) \\ &\leq 2 \cdot \tau^{\ell(\overline{\mathrm{p}}(w))} + 13 \cdot \tau^{\ell(\overline{\mathrm{p}}(w))-1} \leq 13 \cdot \tau^{\ell(\overline{\mathrm{p}}(w))}. \end{aligned}$$

Thus, we have that $x \in \mathrm{Rel}(\overline{\mathrm{p}}(w))$. As was proved above, this means that $\overline{\mathrm{p}}(w)$ will be visited, and since x is added to $\mathrm{Rel}(\overline{\mathrm{p}}(w))$, the algorithm also visits the children of $\overline{\mathrm{p}}(w)$, and in particular, w. ■