Viewing the Rings of a Tree: Minimum Distortion Embeddings into Trees[*]

Amir Nayyeri[†] Benjamin Raichel[‡]

Abstract

We describe a $(1+\varepsilon)$ approximation algorithm for finding the minimum distortion embedding of an n-point metric space, (X, d_X), into a tree with vertex set X. The running time of our algorithm is $n^2 \cdot (\Delta/\varepsilon)^{(O(\delta_{opt}/\varepsilon))^{2\lambda+1}}$ parameterized with respect to the spread of X, denoted by Δ, the minimum possible distortion for embedding X into any tree, denoted by δ_{opt}, and the doubling dimension of X, denoted by λ. Hence we obtain a PTAS, provided δ_{opt} is a constant and X is a finite doubling metric space with polynomially bounded spread, for example, a point set with polynomially bounded spread in constant dimensional Euclidean space. Our algorithm implies a constant factor approximation with the same running time when Steiner vertices are allowed.

Moreover, we describe a similar $(1+\varepsilon)$ approximation algorithm for finding a tree spanner of (X, d_X) that minimizes the maximum stretch. The running time of our algorithm stays the same, except that δ_{opt} must be interpreted as the minimum stretch of any spanning tree of X. Finally, we generalize our tree spanner algorithm to a $(1+\varepsilon)$ approximation algorithm for computing a minimum stretch tree spanner of a weighted graph, where the running time is parameterized with respect to the maximum degree, in addition to the other parameters above. In particular, we obtain a PTAS for computing minimum stretch tree spanners of weighted graphs, with polynomially bounded spread, constant doubling dimension, and constant maximum degree, when a tree spanner with constant stretch exists.

1 Introduction

Given a general metric space (X, d_X), consider the problem of finding a host metric space from within some class of "simple" metric spaces that (X, d_X) can be embedded into while preserving pairwise distances as much as possible. This is a central problem in the algorithmic study of metric spaces, as naturally finding such a simpler metric can unlock a set of efficient algorithmic tools which may be less effective on more complex spaces.

To quantify the extent to which an embedding preserves distances, we consider the (multiplicative) distortion, which is a widely used and studied measure, having many nice properties such as scale invariance. Formally, given metric spaces (X, d_X) and (Y, d_Y), an *embedding* of X into Y is an injective map $f : X \to Y$, with *expansion* e_f and *contraction* c_f defined as

$$e_f = \max_{\substack{x,x' \in X \\ x \neq x'}} \frac{d_Y(f(x), f(x'))}{d_X(x, x')}, c_f = \max_{\substack{x,x' \in X \\ x \neq x'}} \frac{d_X(x, x')}{d_Y(f(x), f(x'))}$$

The *distortion* of f is then defined as $\delta_f = e_f \cdot c_f$. Low distortion embeddings have been extensively studied and have been used in a variety of computer science applications (see [IM04, Ind01, Mat13]).

Among alternatives, one of the most widely studied classes of simpler host metric spaces is the class of weighted trees, whose structure is well understood and readily allows one to apply tools such as dynamic programming. Furthermore, such embeddings have found natural applications, for example, in estimating phylogenetic trees [KW99]. Closely related to minimum distortion embeddings into trees is the problem of finding tree spanners with minimum stretch. Given a graph G, a tree spanner with stretch δ is a spanning tree of G preserving distances up to a multiplicative factor of δ, i.e. a δ distortion embedding into a spanning tree of G. As minimal distance preserving structures, tree spanners have for example found applications in distributed systems [DH98, PR01].

In this paper, we provide parameterized approximation algorithms for minimum distortion embeddings into trees, and minimum stretch tree spanners.

[*] The full version is also available online http://web.engr.oregonstate.edu/~nayyeria/pubs/host-tree.pdf.

[†] School of Electrical Engineering and Computer Science; Oregon State University; nayyeria@eecs.oregonstate.edu; http://web.engr.oregonstate.edu/~nayyeria/. Work on this paper was partially supported by NSF CRII Award 1566624.

[‡] Department of Computer Science; University of Texas at Dallas; Richardson, TX 75080, USA; benjamin.raichel@utdallas.edu; http://utdallas.edu/~bar150630/. Work on this paper was partially supported by NSF CRII Award 1566137 and CAREER Award 1750780.

In other words, we seek to answer the fundamental question, how well can a given metric space (or graph) be represented by a tree?

Significance. Finding an approximate minimum distortion embedding into a tree is a provably hard problem, thus many previous works have focused on the simpler case when the input is the shortest path metric of an unweighted graph (as discussed in detail below). Here we consider the far more general weighted case, i.e. the input is any finite metric. In order to make such a large jump we must parameterize our running times on certain quantitative measures of the source metric, in particular, the doubling dimension and spread.[1] It is important to note that our running time depends only polynomially on the spread, and thus is designed to handle reasonably large ranges of distances. (Note for unweighted graphs the spread is trivially polynomially bounded.) Our running time is also parameterized on the optimal distortion, δ_{opt}. This is natural because when δ_{opt} is large not only is the problem hard to approximate, but also a minimum distortion embedding becomes less informative. Note that more generally removing any of these parameterizations quickly either leads to an open problem or a known hard case. Moreover, whenever these parameters are bounded we get a PTAS for finding the minimum distortion embedding into a tree (or a PTAS for the minimum stretch tree spanner). Thus as a natural example, given a point set in low dimensional Euclidean space, with up to polynomial spread, we can get a $(1+\varepsilon)\delta_{opt}$ embedding in polynomial time if δ_{opt} is below some constant threshold, and otherwise report that the input metric cannot be well represented by a tree.

1.1 Previous Work

Embedding into trees. Nearly a half century ago, Buneman studied the problem of reconstructing trees from distance measures [Bun71]. He showed that an embedding with distortion one can be found in $O(n^4)$ time if it exists. Later, Agarwala et al. [ABF$^+$99] showed that in the absence of a perfect embedding, finding a minimum distortion embedding is not only NP-complete, but actually APX-hard.[2] Moreover, in certain cases much stronger hardness results are known. For example, finding the minimum distortion embedding into the real line, that is a tree of maximum degree two, is hard to approximate within a polynomial factor even when embedding from weighted tree metrics with polynomial spread [BCIS05] (note the problem is much easier for additive distortion, as there is a 2-approximation [HIL98]). Thus it is natural to consider restrictions on the source metric space. In particular, Bădoiu et al. [BIS07] showed the minimum distortion embedding for *unweighted* graphs into trees can be approximated within a constant factor in polynomial time. Their result leads to the state of the art 6-approximation after a couple of improvements [BDH$^+$08, CDN$^+$10]. In contrast to unweighted graphs, far less is known about embedding general metrics into trees. In fact, the only nontrivial approximation, found by Bădoiu et al. [BIS07], gives an embedding with distortion $(\delta_{opt} \log n)^{\sqrt{\log \Delta}}$, where δ_{opt} is the minimum distortion.[3]

Tree spanners. The history of tree spanner algorithms is somewhat similar. Cai and Corneil initiated the study of tree spanners [CC95], and showed that the 1-tree spanner of a weighted graph, if it exists, coincides with the minimum spanning tree, and therefore can be computed efficiently. Nevertheless, computing t-tree-spanners is NP-complete for $t > 1$. For *unweighted* graphs, in the same paper it was shown that the situation is slightly better: there are polynomial time algorithms to find 1-tree-spanners or 2-tree spanners, if they exist, while finding t-tree-spanners is NP-complete for $t > 4$. For unweighted graphs, Emek and Peleg [EP08] and Dragan and Köhler [DK14] show $O(\log n)$-approximation algorithms for finding minimum stretch tree spanners. More recently, Fomin et al. [FGvL11] showed that for constants t and w, t-tree-spanners of treewidth w for bounded degree graphs can be found in polynomial time if they exist [FGvL11] (also see [Pap15]). To the best of the authors' knowledge no approximation algorithm is known for general metric spaces for $t > 1$.

Geometric tree spanners are an interesting special case, where the input is a weighted graph representing the distances between points from a metric space. Not much is known even for this special case. (Note the significance of requiring that the spanner is a tree, as there are many results when other sparse graphs are allowed.) Eppstein [Epp00] asked whether one can compute the minimum stretch geometric tree spanner or the minimum stretch hamiltonian path for a planar point set, either exactly or approximately, in

[1] The spread is the ratio of the largest to smallest distance in the metric, sometimes referred to as the aspect ratio.

[2] Note [ABF$^+$99] states the additive distortion case is APX-hard, however, Chepoi et al. [CDN$^+$10] noted that the proof also implies the same for the multiplicative distortion for a smaller constant.

[3] There is a different line of research for embedding a graph into a *given* tree (or graph), see for example [KRS04, FFL$^+$13, NR17]. We emphasize that the goal of this paper is different as here we look for the best possible tree to embed into. Also, note we focus on multiplicative distortion. See [HIL98, ABF$^+$99] for results on *additive* distortion.

polynomial time. Cheon et al. [CHL07] partially answers this question by showing NP-hardness for the decision problem. Eppstein and Wortman [EW05] give a nearly linear time algorithm to find the minimum stretch star for a planar point set. As for approximation algorithms, prior to our work, no nontrivial approximation was known even for the case when the input is a planar point set. Our results imply a PTAS for computing the minimum stretch spanning tree and the minimum stretch hamiltonian path of a planar point set provided polynomially bounded spread and constant stretch. (Here we seek tree spanners minimizing the maximum multiplicative stretch, although different variants have been studied before. We refer the reader to [LW08] for a list of different tree spanner problems and a survey of corresponding results.)

1.2 Our results

In this paper, we consider the problems of embedding a general metric space into a tree, and finding the minimum stretch tree spanner. We give approximation algorithms whose running times are parameterized with respect to: δ_{opt}, the minimum distortion (or stretch); Δ, the spread of X; and λ, the doubling dimension of X. Our main result is an algorithm to embed a general metric space (X, d_X) into a tree with vertex set X.[4]

Theorem 1.1. *Let X be an n-point metric space, with doubling dimension λ and spread Δ. Also, let δ_{opt} be the minimum distortion of any embedding of X into any tree with vertex set X. For any $0 < \varepsilon < 1$, there is an $n^2 \cdot (\Delta/\varepsilon)^{(O(\delta_{opt}/\varepsilon))^{2\lambda+1}}$ time algorithm to compute a $(1+\varepsilon)\delta_{opt}$ distortion embedding of X into a tree with vertex set X.*

To obtain the above result we first show how to compute a $(1+\varepsilon)$-approximation to the minimum distortion embedding into a tree on vertex set X with bounded degree (which may be of independent interest). Then it is argued our bounded doubling dimension assumption implies that a tree with arbitrary degree can be embedded into a tree with bounded degree with distortion at most $1+\varepsilon$.

The result of Gupta [Gup01], which shows that Steiner vertices can help only up to a factor of eight in the distortion (see Lemma 2.3), implies that the output of the algorithm of Theorem 1.1 is also a constant factor approximation for embedding into a tree when Steiner vertices are allowed.

Corollary 1.1. *Let X be an n-point metric space, with doubling dimension λ and spread Δ. Let δ_{opt} be the minimum distortion of any embedding of X into any tree. For any $0 < \varepsilon < 1$, in $n^2 \cdot (\Delta/\varepsilon)^{(O(\delta_{opt}/\varepsilon))^{2\lambda+1}}$ time one can compute an $(8 + \varepsilon)\delta_{opt}$ distortion embedding of X into a tree.*

Our approach can also be adapted to compute tree spanners for general *weighted graphs*, however, the running time depends on the maximum allowable degree for the tree spanner.

Theorem 1.2. *Let $G = (X, E, w)$ be a weighted graph, and let (X, d_X) be its shortest path metric space. Let λ and Δ denote the doubling dimension and spread of (X, d_X), respectively, and let $\mathsf{deg} > 0$ be some integer. Let δ_{opt} be the minimum possible stretch of any spanning tree of G of maximum degree at most deg. For any $0 < \varepsilon < 1$, there is an $n^2 \cdot (\Delta/\varepsilon)^{\log(\mathsf{deg})(O(\delta_{opt}/\varepsilon))^{2\lambda+1}}$ time algorithm to compute a $(1 + \varepsilon)\delta_{opt}$-tree-spanner with maximum degree at most deg.*

To prove the above theorem we argue our approach can be modified to allow an additional weight constraint function $h : X \times X \to 2^{\mathbb{R}^+}$, which specifies the set of permitted weights for every pair of vertices $x, y \in X$, if we choose to include (x, y) in the tree. Thus the above theorems are special cases, where Theorem 1.1 allows the weight to be any value in $\mathbb{R}^+$, and Theorem 1.2 sets $h(x, y) = \{w_G(x, y)\}$ if $(x, y) \in E$, and $h(x, y) = \emptyset$ otherwise.

The case of geometric tree spanners mentioned above, is the special case when we restrict the permitted weight for each pair to be the metric distance, i.e. when G in the above theorem is the complete graph. Similar to Theorem 1.1, in this case one can argue our assumptions imply a tree with arbitrary degree embeds into a tree with bounded degree with distortion $\leq 1 + \varepsilon$. Thus we get the following corollary, which alternatively can be argued as a direct corollary of Theorem 1.1. (See the full version for details.)

Corollary 1.2. *Let X be a metric space with doubling dimension λ and spread Δ. Let δ_{opt} be the minimum possible stretch of any spanning tree of X. For any $0 < \varepsilon < 1$, there is an $n^2 \cdot (\Delta/\varepsilon)^{(O(\delta_{opt}/\varepsilon))^{2\lambda+1}}$ time algorithm to compute a $(1+\varepsilon)\delta_{opt}$-tree-spanner.*

Note that Theorem 1.1 gives a PTAS for the minimum distortion embedding of a finite metric space (X, d_X) into a tree on vertex set X, provided that δ_{opt} and λ are constants, and that Δ is polynomially

[4] For all our results we actually prove a stronger running time bound. Namely the $(O(\delta_{opt}/\varepsilon))^{2\lambda+1}$ term in the exponent can instead be written as $\log(1/\varepsilon)(1/\varepsilon)(O(\delta^2_{opt}/\varepsilon))^{\lambda}$. In the theorem statements, however, we prefer a less cluttered form, as it allows one to more clearly see the rough dependence on each parameter.

bounded. Under the same set of conditions, Corollary 1.1 gives a constant factor approximation algorithm for embedding X into any tree. Again, under the same conditions, Corollary 1.2 gives a PTAS for computing the minimum stretch geometric tree spanner, and Theorem 1.2 gives a PTAS for computing the minimum stretch bounded degree tree spanner of a weighted graph.

Outline. After covering basic background in Section 2, we give an overview of our approach in Section 3. In Section 4 we present our main result for approximating minimum distortion embeddings of metric spaces into bounded degree trees. We then show how to remove the bounded degree assumption in Section 5, by proving that with $(1+\varepsilon)$ distortion, any tree can be embedded into a tree whose degree is bounded by a function only depending on ε and the doubling dimension. Finally, in Section 6, we show our algorithm can be adapted to find tree spanners by formulating the problem in a more general setting.

2 Preliminaries

Graphs and metrics. We use $G = (V, E, w)$ to denote an undirected graph with vertex set V, edge set E, and positive edge weight function $w : E \to \mathbb{R}^+$. The ***shortest path metric*** (V, d_G) of a graph G is defined by the distance function $d_G : V \times V \to \mathbb{R}^{\geq 0}$, where $d_G(u, v)$ is the length of the shortest u-to-v path in G. For each $u \in V$, we define $\mathbf{adj}_{\boldsymbol{G}}\boldsymbol{(u)}$ to be a list of all incident edges to u in G.

Throughout the paper we use Δ to denote the ***spread*** of the given finite metric space (X, d_X), that is $\Delta = (\max_{x\neq y\in X} d_X(x, y))/(\min_{x\neq y\in X} d_X(x, y))$. Given a metric space (X, d_X), a point $x \in X$, and a radius $r \in \mathbb{R}^+ \cup \{0\}$, the ***ball*** $B(x, r)$ is the subset of all points of X whose distance to x is at most r. The ***doubling dimension*** of a metric space (X, d_X) is the smallest $\lambda \in \mathbb{R}^+$ such that for any $r \in \mathbb{R}^+$, each ball of radius r can be covered by at most 2^λ balls of radius $r/2$. A metric space is called ***doubling*** if λ is bounded by a constant (independent of the size of the metric). The following lemma of Gupta et al. [GKL03] is helpful in the analysis in this paper.

Lemma 2.1. ([GKL03], Proposition 1.1) *Let (X, d_X) be a metric with doubling dimension λ, and let $X' \subseteq X$. If all pairwise distances in X' are at least ℓ, then any ball of radius R in X contains at most $\left(\frac{2R}{\ell}\right)^\lambda$ points of X'.*[5]

Embeddings and distortion. An embedding of a metric space (X, d_X) to a metric space (Y, d_Y) is an injective map $f : X \to Y$. The ***contraction*** c_f and the ***expansion*** e_f of f are defined as

$$c_f = \max_{\substack{x,y\in X\\ x\neq y}} \frac{d_X(x, y)}{d_Y(f(x), f(y))},\quad e_f = \max_{\substack{x,y\in X\\ x\neq y}} \frac{d_Y(f(x), f(y))}{d_X(x, y)}$$

An embedding is called ***non-contracting*** if its contraction is at most one. The ***distortion*** of f is defined as $\delta = c_f \cdot e_f$. Often in this paper we consider the identity map as an embedding from a metric space (X, d_X) to the shortest path metric (X, d_T) of a tree $T = (X, E_T, w_T)$. To simplify notation, in these cases, we drop f and compare x-to-y distance in X, denoted by $d_X(x, y)$, with the x-to-y distance in T, denoted by $d_T(x, y)$. Also to simplify, we refer to the identity map (X, d_X) to (X, d_T) as the ***embedding defined by*** T. We use $\delta_{opt}(X)$ to refer to the smallest possible distortion for embedding X into any tree. When it is clear from the context we use the same notation, $\delta_{opt}(X)$, to refer to the smallest possible distortion for embedding X into any tree *with vertex set* X. We use $\delta_{opt}(X, \mathsf{deg})$ to refer to the smallest possible distortion for embedding X into any tree of maximum degree at most deg. Since distortion is scale invariant a non-contracting embedding of expansion $\delta_{opt}(X)$ always exists, and throughout the text we assume we are looking for a such an embedding.

We found the following lemma helpful when working with embeddings between shortest path metrics of graphs.

Lemma 2.2. ([KRS04], Proposition 2.3) *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two positively weighted undirected graphs, and let d_G and d_H be their shortest path metrics, respectively. Let $f : V_G \to V_H$ be a bijection. Then the expansion of f is achieved by an adjacent pair $u, v \in V_G$, and the contraction of f (or the expansion of f^{-1}) is achieved by an adjacent pair $x, y \in V_H$.*

In this paper, we consider embedding into trees both when Steiner vertices are allowed and when they are not allowed. The following Lemma of Gupta, ensures that the optimal tree metrics for these two problems differ up to a factor of at most eight.

Lemma 2.3. ([Gup01], Theorem 1.1) *Given a tree $T' = (V', E', w')$ with shortest path metric $d_{T'}$, and a set of required vertices $V \subseteq V'$, there exists a tree $T = (V, E, w)$ with shortest path metric d_T such that for all $x, y \in V$, $1 \leq \frac{d_T(x,y)}{d_{T'}(x,y)} \leq 8$. Moreover, T can be computed in polynomial time.*

Tree spanners. Let $G = (V, E_G, w_G)$ be a graph, and let $T = (V, E_T, w_T)$ be a spanning tree

[5] Note that λ in their paper is the doubling constant, whereas in this paper it denotes the doubling dimension.

of G, where w_T is the restriction of w_G to E_T. Let $e = (u, v) \in E_G$. The ***stretch*** (or dilation) of the edge e is defined as $str_T(e) = d_T(u, v)/d_G(u, v)$. Note that $d_T(u, v) \geq d_G(u, v)$. The ***stretch*** (or dilation) of T is then defined as the maximum stretch of the edges in E_G, $str_T = \max_{e \in E_G} str_T(e)$. By Lemma 2.2, the identity map is a map of distortion str_T from (V, d_G) to (V, d_T). If $str_T = t$, we say that T is a ***t*-tree-spanner** of G. Hence, finding the minimum stretch spanning tree is equivalent to finding the spanning tree into which the identity map has the lowest distortion.

Consider the case when the input is instead a metric space (X, d_X). Let $G_X = (X, E_X, d_X)$ be the complete graph over X, where for each $x, x' \in X$, the weight of edge (x, x') is $d_X(x, x')$. Then one can analogously define the ***geometric stretch*** of a pair in X and a ***geometric t-tree-spanner*** of X, by using the graph G_X in the above definitions.

3 Overview

Here we sketch our algorithm and its analysis for embedding an n-point metric space (X, d_X) into a tree with vertex set X, which by Lemma 2.3 will also serve as a sketch for the case when Steiner vertices are allowed. Our algorithms for the tree spanner cases follow a similar high level approach as sketched here, but require enforcing a set of additional constraints (on edge weights). The details of these constraints and their enforcement can be found in Section 6.

Throughout, for any $x \in X$ the term *point* is used when referring to x in the metric space (X, d_X), and the term *vertex* when referring to x in the tree. Here we assume we are given a value δ such that $\delta \geq \delta_{opt}$, where δ_{opt} is the minimum distortion of any embedding of (X, d_X) into a tree (with vertex set X). Ultimately our actual algorithm performs an exponential search to approximately find δ_{opt}, where the procedure sketched below can be seen to fail and hence return "false" if $\delta < \delta_{opt}$. Moreover, assume the spread of (X, d_X), denoted by Δ, is polynomially bounded, that δ is a constant, and X is doubling. Under these conditions, we describe a polynomial time algorithm to compute an embedding of X into a tree with $O(\delta)$ distortion. Our actual algorithm achieves $(1+\varepsilon)\delta$ distortion, though for simplicity here we are satisfied with this weaker guarantee.

As distortion is scale invariant, as remarked in the previous section, we can restrict our attention to non-contracting embeddings where the expansion is at most δ. Moreover, scale invariance also implies that we can assume the smallest distance in (X, d_X) is 1, and hence the largest distance is Δ. As the expansion is at most δ, this implies we can restrict our attention to trees with edge weights in the interval $[1, \delta\Delta]$. Finally, to make things simpler we assume all edge weights are integers (which is valid since we are only seeking a constant factor approximation).

We start by describing a more comprehendible version of the algorithm containing many of the key ideas, though with an exponential running time. Then we modify the algorithm step by step to obtain a polynomial running time.

Stitching local views. For each point $x \in X$, our algorithm tries to enumerate all possible local "views" of what a distortion at most δ embedding could look like when standing at the vertex x (i.e. the image of point x). Then, our algorithm tries to stitch together these views (each containing only partial information of the tree) into a tree T on X with $O(\delta)$ distortion.

As a first attempt, we define a local *view* at a vertex x to contain precise information about the location of all other vertices relative to the vertex x. Specifically, a view V_x at a vertex x includes the following information (from the tree of an at most δ distortion embedding):

(1) The degree of x.

(2) For each $y \in X$: (a) the branch of x leading to y, and (b) the distance of x to y.

Figure 1-left shows a possible view at a vertex x and a possible view at vertex y on a tree with vertex set $\{a, b, \ldots, h, i, x, y\}$. Note any embedding of X into a tree T *implies* a view V_x at each vertex x. In this case, we also say V_x *extends* to T. Note that a view can extend to more than one tree, as the distance/branch information at one vertex is not sufficient to uniquely reconstruct a tree. Figure 1-right shows a tree that is an extension of both of the views (at x and y) on the left.

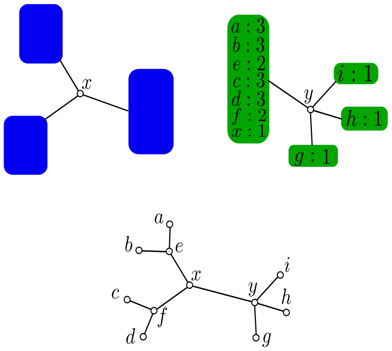

Figure 1: Top: views at x and y, Bottom: a tree that is an extension of both views. To keep the figure readable, unweighted distance are used, though in general weights are allowed.

We now formalize the notion of *stitching*. For a given view V_x at a vertex x, for every $z \in X$ define (i)

$b_x(z)$ to be the branch of x that leads to z according to V_x, and (ii) $d_x(z)$ to be the x, z-distance according to V_x. For a given view V_y at another vertex y, similarly define $b_y(z)$ and $d_y(z)$. Let $b = b_x(y)$ denote the branch label of y in V_x, and let $b' = b_y(x)$ denote the branch label of x in V_y. Intuitively, we say that V_x and V_y are *stitchable* if when we identify the labels b and b', all pieces of information in V_x and V_y look consistent. Specifically,

(1) $d_x(y) = d_y(x)$. Call this value ℓ (i.e. the length of the edge (x, y)).

(2) For any $z \in X$,

  (a) $b_x(z) = b$ if and only if $b_y(z) \neq b'$, and

  (b) if $b_x(z) = b$ then $d_x(z) = d_y(z) + \ell$, otherwise $d_x(z) = d_y(z) - \ell$.

For example, the views in Figure 1 are stitchable, and the stitched result is shown in Figure 2-left.

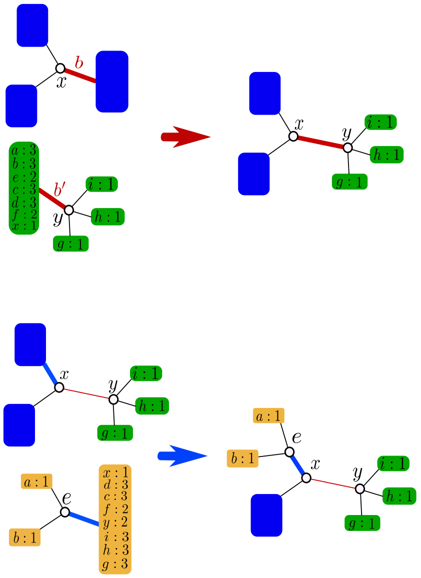

Figure 2: Top: stitching together a view at x and a view at y (to build the edge (x, y)), Bottom: stitching a view at e to the view at x (to build the subtree of x, y, and e).

The stitching operation tells us how to build one edge of our desired tree. Next, by stitching another view to this "edge" one obtains a larger subtree (see Figure 2-right). By continually stitching together more and more views, our ultimate goal is to obtain a full tree T on vertex set X. So suppose we successfully stitched together views into such a tree. What can be said about the resulting tree this stitching produces? First, it is not hard to see that requiring consistency of the branch information implies the resulting tree defines a valid embedding (i.e. a point cannot be mapped to two different vertices). Second, observe that a view centered at some vertex x records the distance from x to the image of any other $z \in X$ under this embedding, and so requiring consistency of distances can be shown to imply that the view at z must also record the same distance to x. This implies that if for each $x \in X$, if locally at the view of x no distance from x was expanded by more than δ, then globally the resulting tree defines an embedding with expansion at most δ, and hence distortion at most δ by our non-contracting assumption. Thus we restrict our attention to *plausible* views, where a view at a vertex x is plausible if for all $z \in X$, the x-z distance in the view has a value between $d_X(x, z)$ and $\delta d_X(x, z)$.

There are many potential views at a vertex x which are plausible. Though as described below, by a number of careful summarization steps we can make the view descriptions compact enough such that we can enumerate all possible plausible views. However, deciding which views to stitch together from these lists is still a daunting task. Fortunately, dynamic programming can be used to give an algorithm whose running time is polynomial in the number of views. Interestingly, while this dynamic programming ultimately works because our goal is to stitch together a tree (a structure amenable to dynamic programming), the dynamic programming we now describe is not actually done over a tree.

Assembling the tree. Provided the set of all plausible views at every vertex, we now describe a dynamic program which builds a tree T on vertex set X by stitching together appropriate views. To facilitate our dynamic program, we fix an arbitrary point $r \in X$, and root all trees with vertex set X at r. Fixing r allows us to uniquely define the set of descendants for each view V_x (at a vertex x). Namely, $y \in X$ is a descendant of x in V_x if the branch of x leading to y (according to V_x) is different from the branch of x leading to r (according to V_x). In other words, y is a descendant in V_x if for all extensions of V_x to a tree T with root r, y is a descendant of x in T. We denote the set of descendants of x according to V_x by $des(x, V_x)$. We emphasize it is possible a vertex y is a descendant of x according to a view V_x and *not* a descendant of x according to another view V'_x.

Now we are ready to define our subproblems. For any plausible view V_x at any $x \in X$, we say $Subtree(x, V_x)$ is true if and only if there is a *set* of views $\mathcal{V}$, one per vertex of $des(x, V_x)$, such that

- $\mathcal{V} \cup \{V_x\}$ can be stitched together to build a tree with vertex set $des(x, V_x) \cup \{x\}$ and root x.

The definition of $Subtree(\cdot, \cdot)$ implies the following recursive algorithm to check if $Subtree(x, V_x)$ is true. Let $b_0, b_1, b_2, \ldots, b_t$ be all the branches of x according to V_x, and let b_0 be the branch that leads to r. $Subtree(x, V_x)$ is true if and only if there are $y_1, \ldots, y_t \in X$ and views $V_{y_1}, \ldots, V_{y_t}$ such that for every $i \in \{1, \ldots, t\}$ we have:

(1) b_i is the branch of x that leads to y_i (according to V_x),

(2) V_x can be stitched to V_{y_i}, and

(3) $Subtree(y_i, V_{y_i})$ is true.

Using this recursive relation we can build our dynamic programming table to check if there exists a plausible view V_r at the root r such that $Subtree(r, V_r)$ is true. If so, by tracing back through the dynamic programming table, we can stitch together a set of plausible views to build a tree T with distortion at most δ. (Note it is now easy to see that if we had allowed $\delta < \delta_{opt}$, in this case the dynamic program would fail, and hence we would know to return "false".)

Observe that the running time of our dynamic program is clearly polynomial in terms of $n = |X|$ and the maximum number of plausible views at any vertex. Unfortunately however, with the current definition of a view, the total number of plausible views at a vertex can be exponentially large. A trivial bound on the number of such views at a vertex $x \in X$ is,

$$n \cdot (n)^n \cdot (\delta\Delta)^n,$$

as there are at most n choices for the degree of x, and for any other y in $X \setminus \{x\}$ there are at most n choices for its branch, and $\delta\Delta$ choices for its distance. Recall the $\delta\Delta$ bound on the number of distances follows since the maximum distance in (X, d_X) was Δ, and we are looking for an embedding of distortion at most δ (and we assumed integral distances).

In Section 4 we prove that a doubling tree metric embeds into a bounded degree tree metric with constant distortion. Therefore, if our goal is to achieve a constant factor approximation, then we can assume that the degree of our target tree is bounded by a constant. This reduces our bound on the number of plausible views at any vertex to

$$O(1) \cdot (O(1))^n \cdot (\delta\Delta)^n = (O(\Delta))^n,$$

as we assumed δ is a constant. At this point the number of views, and therefore the running time of our dynamic program, is still exponential in n. Note however that up until the point we assumed the tree degree was constant, our algorithm had actually been exact. Thus we can now take advantage of the extra slack of moving to an approximation, to drastically improve the running time.

Hierarchical nets. To reduce the above running time we have no choice but to make the views more concise. To that end, for each point $x \in X$, we choose a subset $I_x \subseteq X$, and include branch/distance information only for the points of I_x (instead of all of X) in *any* view at x. We say a vertex y is *visible* from x if $y \in I_x$. We now argue that if one selects the visible vertices for each view carefully, then only a logarithmic number is sufficient to guarantee that the resulting assembled tree of plausible views has $O(\delta)$ distortion.[6]

Now lets figure out how to construct I_x. This subset of X will still somehow need to approximately capture the distance information from all $y \notin I_x$ to x. Suppose that for any $y \notin I_x$, we guarantee that there is some $z \in I_x$ such that $d_X(y, z) \leq d_X(x, y)/c$ for some constant $c > 1$. Then in this case up to a constant factor $d_X(x, z) \approx d_X(x, y)$, and so potentially the information recorded for z can be used as a proxy for that of y. (For example, if $d_X(y, z) \leq d_X(x, y)/2$, then by applying the triangle inequality (twice) we have $d_X(x, z) \in [(1/2)d_X(x, y), (3/2)d_X(x, y)]$.) Ultimately, however, we need to approximate the distance from y to x in the tree, not in the input metric space. Assuming that the x-z and y-z distances are not contracted and expand by at most δ when going from (X, d_X) to the tree, then by simply changing our requirement to $d_X(y, z) \leq d_X(x, y)/(c\delta)$, we assure only a constant factor distance error in the tree. So how do we ensure that the x-z and y-z distances are not contracted and expand by at most δ? Well, since z is visible to x, the plausibility of our current view at x ensures this for the x-z distance. For the y-z distance we then should ensure z is also visible from the view at y. In short, for any $y \notin I_x$, we need to guarantee there is a $z \in I_x$ that is (i) sufficiently close to y, and (ii) is visible from y. We now define I_x sets which achieve this goal while being concise.

To construct the I_x we use r-nets, a standard geometric tool, where an r-net is any subset of X such that (i) pairs of net points are at least r apart and (ii) every point in X has distance at most r to its nearest net point. The above discussion then implies that I_x should be constructed such that for any $y \in X$ it contains y's nearest net point from an r-net of X where (up to a constant) $r = d_X(x, y)/\delta$. Now we want I_x to be small, so we cannot afford to build a custom radius net for every possible distance

[6] For our dynamic program to work, it is crucial these views determine the branch information for all vertices. However, we now focus only on preserving distances, as we can prove this implies we can determine branches exactly.

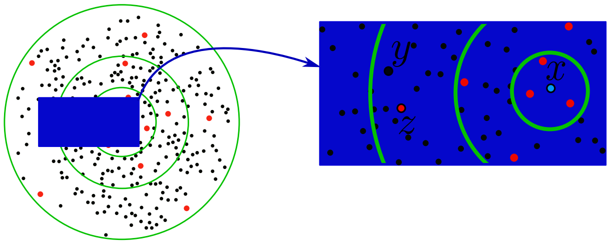

Figure 3: Left: hierarchical nets around x (net points are red, x is blue), right: z is used to estimate the distance to y in x's view.

to x. Thus instead we bin distance by factors of δ. Specifically, we construct a set of nested nets: $X = X_{\geq 0} \supseteq X_{\geq 1} \supseteq \ldots \supseteq X_{\geq \log_\delta \Delta}$, where $X_{\geq s}$ is a δ^s-net. (Note $\delta^{\log_\delta \Delta} = \Delta$ is the largest radius we need to consider as Δ is the largest distance in (X, d_X). Also, such a nested set of nets can be easily computed with the standard greedy k-center algorithm.) So consider any $y \in X$, where $d_X(y, x)$ lies somewhere in the interval $[\delta^{s+1}, \delta^{s+2}]$, for some integer s. Then I_x should be constructed so that it includes y's nearest net point in $X_{\geq s}$ (as $d_X(y, x)$ may be as small as δ^{s+1}). All points whose distance to x lie in this range are contained in the ball $B(x, \delta^{s+2})$, and hence their nearest $X_{\geq s}$ net points are contained in $B(x, 2\delta^{s+2})$. Thus in general I_x is constructed by including all net points from $X_{\geq s}$ contained in $B(x, 2\delta^{s+2})$, for all values of s. Intuitively, I_x is thus a net of points whose density exponentially decreases with respect to the distance from x (see Figure 3).

Construct the I_x sets as described above for all $x \in X$. Now fix some I_x, and for any $y \notin I_x$, consider its nearest neighbor in all different scale nets. Specifically, let z_s be the nearest neighbor of y in $X_{\geq s}$. By construction all these nearest neighbors are visible from y (i.e. are in I_y). Let t be the smallest index such that z_t is also visible from x. (Note t is well defined as the points in $X_{\geq \log_\delta \Delta}$ are visible to everyone.) It can be shown that for this choice of z_t (in particular, because z_{t-1} is not visible from x), that z_t is sufficiently close to y (relative to the distance to x), and thus z_t is the point we sought above (visible to both x and y and) which guarantees our desired properties.

The only question that remains, is how big is I_x? Since X is doubling and δ is a constant, there are $O(1)$ points from $X_{\geq s}$ inside each ball $B(x, 2\delta^{s+2})$. (This follows from Lemma 2.1 and the packing property of nets, i.e. property (i) above.) Therefore, the total size of I_x is bounded by $O(\log_\delta \Delta)$, the number of concentric balls. (Note that $\log_\delta \Delta = O(\log \Delta)$ if $\delta > 2$, which we can assume as a constant factor approximation suffices for the overview.) With these conciser views, the number of plausible views per vertex goes down to

$$(O(\Delta))^{O(\log \Delta)} = \Delta^{O(\log \Delta)},$$

which readily implies an algorithm whose running is polynomial in n and quasi-polynomial in Δ.

Anchors and mile markers. We managed to reduce the number of visible vertices in each view to $O(\log \Delta)$, thus obtaining an algorithm with quasi-polynomial dependence on the spread. Getting a polynomial dependence on the spread by reducing the number of visible vertices in each view to a constant seems impossible. Thus alternatively, we now seek to improve this dependence by storing less information about the distances to each visible vertex.

At first blush, the solution may seem obvious. Just record distances approximately rather than exactly, since our solution is already approximate because instead of mapping each point we only mapped its closest net point at an appropriate scale. Specifically, if a view V_x is mapping a scale s net point y, then record the distance from x to y in the image up to a factor of roughly δ^s. This approach however has a fatal flaw, as deciding whether views can stitch together becomes ambiguous, especially over relatively short edges. Suppose the view at x claims the distance to y in the tree is in between $10\delta^s$ and $11\delta^s$. As we walk from x to y in the tree, at some point our estimate of the distance to y in the current view will have to be decreased (otherwise we never reach y). The issue is that in our dynamic program as we stitch views, since we don't actually know the tree structure, there is no way to know when this update should happen. Specifically, our dynamic program must try both long and short edges on this path. If it tries an edge that is longer than δ^s, well then it knows the estimate must be decreased at the next view. However, if the next edge is much smaller than δ^s then knowing whether to update or not means knowing where in the range $[10\delta^s, 11\delta^s]$ the distance to y lies, i.e. we are back to needing to know the distance exactly. In other words, if we walk down a long path with short edges, the views across each edge look consistent, but by the time we reach y something will have gone wrong.

To resolve this issue, rather than recording the exact distance to the image of each net point, instead we fix an arbitrary vertex $\mathsf{a} \in X$, called the *anchor*, and for any given view V_x centered at a vertex x we only record the distance from x to a exactly. Note that to check if two views across a given edge in the tree are consistent with respect to the anchor, we just verify that their claimed distances from the view centers to the anchor differ by exactly the length of the edge. (Note whether the distance should go up

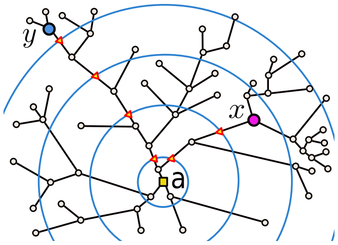

Figure 4: Anchor (yellow square), beacon rings (blue circles), and x-to-y mile marks (red triangles).

or down, depends on whether we are walking towards or away from the anchor, and hence we also record the branch of the anchor.)

Now consider a tree T, and for a given integer $s \leq \log_\delta \Delta$, imagine placing a set of concentric rings around the anchor a, with radii $i\delta^s$ for all integers $i \geq 0$ (see Figure 4). Call these *beacon rings* of scale s and consider the locations where these rings cross T. For any vertices x, y we define their scale s approximate distance to be the number scale s beacon rings on the unique x, y-path in T. As a simple analogy, when driving from point A to point B on the highway, if one records the number of mile markers that get passed, then one will know the distance from A to B, at the resolution of a mile. Of course, our algorithm does not know T a priori, but when stitching together two views V_x and V_y, the number of rings that cross the resulting edge (x, y) can be computed from the exact distance of x and y to the anchor (without knowing T). Thus, we can achieve an approximate version of our stitching definition.

In a view V_x at x, we therefore register the exact distance to the anchor point, and for each visible scale s net point we register its distance from x with only δ^s accuracy (by recording the number of beacon ring crossings of scale s on its path from x). Since any visible scale s net point has distance $O(\delta^{s+2})$ from x, as δ is a constant, there are $O(1)$ choices for its distance estimate from x. As there are $O(\log \Delta)$ visible points from x, there are $(O(1))^{O(\log \Delta)}$ choices for the branch/distance information of all visible points from x. Moreover, there are $O(\Delta)$ choices for the branch/distance of the anchor point, and $O(1)$ choices for the degree of x. Overall, the number of plausible views at vertex s is bounded by,

$$(O(1))^{O(\log \Delta)} \cdot O(\Delta) \cdot O(1) = O(\mathrm{poly}(\Delta)).$$

Recall that ultimately we list the set of all possible plausible views at each of the n vertices in X, and then run a dynamic programming algorithm whose running time is polynomial in the total number of views. Thus, overall our running time is $O(\mathrm{poly}(n\Delta))$.

Techniques of this paper. The idea of enumerating selected pieces of information about the embedding and combining these pieces using dynamic programming over an amenable structure such as a tree or line, has a long tradition in the embedding community. See for example [KRS04, BDG+05, FFL+13, NR15, NR17], which includes previous works by the authors. However, which pieces of information to consider and how to apply the dynamic programming is problem specific, and is what distinguishes these result from one another. Thus it is important to note that while for consistency we adopt the terminology of "views" previously used by the authors in [NR15, NR17], the information contained in these views differs substantially. Moreover, the main idea of defining approximate distance relative to anchor points is new, and has the potential for future applications (as well as potentially improving/simplifying previous results). Additionally, in these listed previous works the target structure (a tree or line) was known and fixed (though which points map to which vertices was not), and so the dynamic programming was more natural. Interestingly in our case, as the tree structure is not fixed in advance, our dynamic programming is not done over the tree, though still manages to compute it in the end.

4 Bijective embedding into trees

In this section, we consider the problem of embedding a metric space (X, d_X), with doubling dimension λ and spread Δ into a weighted tree $T = (X, E_T, w_T)$ *with vertex set* X (i.e. defining a bijection), and maximum degree deg. We use $\delta_{opt}(X, \mathsf{deg})$ to denote the minimum achievable distortion of such an embedding. We show a $(1+\varepsilon)$-approximation algorithm for finding this optimal embedding (Theorem 4.1). Theorem 1.1 is then immediately implied from Theorem 4.1 and Corollary 5.1.

4.1 Setup During this section assume that the smallest distance in X is one, so the largest distance is Δ. This can be ensured by scaling X. Let $\delta \geq \delta_{opt}(X, \mathsf{deg})$, and let a be an arbitrary fixed point of X, called the ***anchor***. Let $\{x_0\} = X_{\geq S+1} \subseteq X_{\geq S} \subseteq \ldots \subseteq X_{\geq 0} = X$, where $X_{\geq s}$ is a maximal subset of points with mutual distances at least δ^s, $S = \lceil \log_\delta(\Delta) \rceil$, and $x_0 \in X$ is an arbitrary point. For technical reasons we extend these sets to be defined for negative values of s, where $X_{\geq s} = X$ for any negative value s. A point $x \in X$ is called a ***scale s point*** if $X_{\geq s}$ is the sparsest net in the sequence that contains x. The ***scale s nearest neighbor*** of a point $x \in X$ is denoted by $nn_s(x)$, and is defined to be the closest point of $X_{\geq s}$ to x. Note that by the maximality of $X_{\geq s}$, we have that $d_X(x, nn_s(x)) < \delta^s$. Therefore, for example, $nn_s(x) = x$ for all $x \in X$ and for any $s \leq 0$.

We build a tree into which X can be embedded with nearly optimal distortion in this section. Part of the process is to find the edge weights of this tree. The following lemma limits the range of the search space for the weight values, while ensuring a bounded approximation factor.

Lemma 4.1. *Let $G = (V, E, w)$ be a graph with minimum edge weight one, and let d_G be the shortest path metric of G. For any $0 < \sigma \leq 1$, G can be embedded to a graph $G' = (V, E, w')$ whose edge weights are multiples of σ with distortion $\leq 1 + \sigma$.*

Proof. Let $G' = (V, E, w')$ be the graph obtained from G by setting the weight of each edge e to $w'(e) = \lceil w(e)/\sigma \rceil \cdot \sigma$. We bound the distortion of the identity map from (V, d_G) to $(V, d_{G'})$. Since $w'(e) \geq w(e)$ the map is non-contracting. Furthermore, for each $e = (x, y) \in E$, we have

$$w'(e) \leq w(e) + \sigma \leq w(e)\left(1 + \frac{\sigma}{w(e)}\right) \leq w(e)\,(1 + \sigma)\,,$$

as $w(e)$ is at least one. Hence, by Lemma 2.2, the distortion is at most $1 + \sigma$.

4.2 Views The key building blocks used in our algorithm are *views*, which are collections of relevant information about what the embedding looks like around the images of points in X. This information is limited in scope so that it can be guessed by our algorithm. Specifically the view at a vertex x ('vertex' signifying it is the image of the 'point' x), specifies the degree of the image of x, the location of the anchor vertex relative to the image of x, and approximate relative locations of the images of all scale s points that are at distance $O(\delta^{s+2})$ from x in the preimage. To describe the location of the anchor vertex we specify the branch (edge) adjacent to vertex x which leads to the anchor as well as the exact distance to the anchor. Similarly, for each vertex y which is the image of one of these scale s points, we specify the branch, but rather than specifying the distance to y exactly we just record the number of beacon ring crossings (as described in the overview) of the x-to-y path in the image.

Formally,$V_x = (\deg_x, (A_x^b, A_x^d), \{(b_x^s, r_x^s)\}_{L \leq s \leq S})$ is a ***view*** at $x \in X$ with parameters $(\mathsf{deg}, \sigma, c, \delta)$ where each component is defined as follows.

(1) An integer, $\deg_x \in \{1, 2, \ldots, \mathsf{deg}\}$.

(2) (a) $A_x^b \in \{1, 2, \ldots, \deg_x\} \cup \{null\}$.

  (b) $A_x^d \in \{0, \sigma, 2\sigma, \ldots, \lfloor \delta\Delta/\sigma \rfloor \cdot \sigma\}$.

(3) For each $L \leq s \leq S$

  (a) $b_x^s : X_{\geq s} \cap B(x, c \cdot \delta^{s+2}) \to \{1, \ldots, deg_x\} \cup \{null\}$.

  (b) $r_x^s : X_{\geq s} \cap B(x, c \cdot \delta^{s+2}) \to \{0, 1, \ldots, \lfloor c\delta^3 \rfloor\}$.

In the definition above we set $L = \lfloor -(\log_\delta c + 3) \rfloor$, as the minimum distance in X is one, and so $B(x, c \cdot \delta^s)$ is empty for $s \leq L$. Throughout the text c is considered to be a sufficiently large value, which will be specified later, and intuitively acts as a dial controlling the approximation quality of the embedding. Whenever, it is clear from the context, we drop the specification of the parameters to simplify the explanation. We say that a point y is ***visible*** under V_x at scale s, if it is in the domain of b_x^s and r_x^s. We say that a point y is visible under V_x if there exists an $L \leq s \leq S$ such that y is is visible under V_x at scale s.

The first step of our algorithm is to list the set of possible views at every point of X. The following lemma bounds the number of such views.

Lemma 4.2. *There are at most*

$$\frac{1}{\sigma} \cdot (c\Delta)^{O(\log_\delta(c \cdot \mathsf{deg}))(2c\delta^2)^\lambda}$$

different views at any $x \in X$. Moreover, a list of these views can be constructed in time linear in the list size.

Proof. We enumerate all possibilities for a view $V_x = (\deg_x, (A_x^b, A_x^d), \{(b_x^s, r_x^s)\}_{L \leq s \leq S})$ at x. For the first parameter $\deg_x$ in the tuple, there are at most deg possibilities. For (A_x^b, A_x^d) there are at most $(\mathsf{deg} + 1)(\delta\Delta/\sigma + 1)$ possibilities.

So what remains is to bound the possibilities for the b_x^s and r_x^s functions. For each point $y \in X_{\geq s} \cap B(x, c \cdot \delta^{s+2})$ and for each scale $L \leq s \leq S$, there are at most $\mathsf{deg}+1$ possibilities for $b_x^s(y)$, and at most $c\delta^3 + 1$ possibilities for $r_x^s(y)$. By Lemma 2.1, there are at most $(2c\delta^{s+2}/\delta^s)^\lambda = (2c\delta^2)^\lambda$ points in $X_{\geq s} \cap B(x, c \cdot \delta^{s+2})$. Therefore, for any $L \leq s \leq S$, there are at most $((\mathsf{deg}+1) \cdot (c\delta^3 + 1))^{(2c\delta^2)^\lambda}$ number of choices for b_x^s and r_x^s.

There are $S - L + 1$ scales overall, so the total number of views at x is at most:

$$\begin{aligned}
&\mathsf{deg} \cdot (\mathsf{deg} + 1) \cdot (\delta\Delta/\sigma + 1) \cdot \\
&\quad \left(((\mathsf{deg} + 1)(c\delta^3 + 1))^{(2c\delta^2)^\lambda}\right)^{\lceil \log_\delta(\Delta) \rceil - \lfloor -(\log_\delta c + 3) \rfloor + 1} \\
&= \frac{\Delta}{\sigma}\left((\mathsf{deg} \cdot c\delta)^{O(\log_\delta(c\Delta))(2c\delta^2)^\lambda}\right) \\
&= \frac{\Delta}{\sigma}\left(\delta^{O(\log_\delta(c \cdot \mathsf{deg}) \log_\delta(c\Delta))(2c\delta^2)^\lambda}\right) \\
&= \frac{1}{\sigma}(c\Delta)^{O(\log_\delta(c \cdot \mathsf{deg}))(2c\delta^2)^\lambda}.
\end{aligned}$$

Feasibility/Plausibility. From a list of views generated by Lemma 4.2, we would like to only keep the views that can be completed into ***feasible*** trees on X, that is trees that define non-contracting embeddings of expansion at most δ. To formally describe our desired properties for such views, we define restriction of embeddings, and extensions of views as follows.

Let $T = (X, E_T, w_T)$ be a tree, and let $x \in X$. We define the ***restricted*** view of T around x to be the view $V_x = (\deg_x, (A^b_x, A^d_x), \{(b^s_x, r^s_x)\}_{L \le s \le S})$ specified as follows.

(1) $\deg_x = \deg_T(x)$, where $\deg_T(x)$ denotes the degree of x in T.

(2) $A^d_x = \lfloor d_T(\mathsf{a}, x)/\sigma \rfloor \cdot \sigma$.

(3) Fix a global ordering on the edges of T, and let $\ell : \mathrm{adj}(x) \to \{1, \ldots, \deg_x\}$ be the bijection that (for every $1 \le i \le \deg_x$) assigns the ith element of $\mathrm{adj}(x)$ to i. We have:

  (a) If $x = \mathsf{a}$ then $A^b_x = null$, otherwise $A^b_x = \ell(e)$, where e is the first edge of the x-to-a path in T.

  (b) For each $L \le s \le S$ and $y \in X_{\ge s} \cap B(x, c\delta^{s+2})$,

    (i) If $x = y$ then $b^s_x(y) = null$, otherwise $b^s_x(y) = \ell(e)$, where e is the first edge of the x-to-y path in T.

    (ii) $r^s_x(y) = \lfloor d_T(x, \mathsf{a})/\delta^s \rfloor + \lfloor d_T(y, \mathsf{a})/\delta^s \rfloor$ - $2\lfloor d_T(u, \mathsf{a})/\delta^s \rfloor$, where u is the vertex in T that is closest to a on the path from x to y, i.e. u is the lowest common ancestor of x and y if the root is a. (Roughly speaking, up to the δ^s factors this is the distance between x and y as $d_T(x, u) + d_T(y, u) = d_T(x, y)$)

Note that the restricted view of T around x is uniquely defined for fixed values of a, deg, δ, c, and σ. If V_x is the restricted view of T around x, we say that T is an ***extension*** of V_x. Note that a view can possibly be extended to several different trees. A view is called ***feasible*** if it can be extended to a feasible tree. Such an extension is called a ***feasible extension*** of the view.

Ideally, we would like to be able to disregard all non-feasible views from the lists computed by Lemma 4.2. However, it seems impossible to determine feasibility by merely examining a view in isolation from other views. Fortunately, the following weaker condition on views, which can be tested quickly, suffices for our algorithm. We say that a view V_x is ***plausible*** if for any $L \le s \le S$ and any $y \in \mathrm{Dom}(r^s_x)$, we have

$$d_X(x, y) - 2\delta^s \le r^s_x(y) \cdot \delta^s \le \delta d_X(x, y) + 2\delta^s.$$

Note radii of successive beacon rings differ by δ^s, and hence the need for the additive factor of $2\delta^s$, as this is longest a shortest path can be without crossing a beacon ring. Moreover, observe that at a sufficiently small scale the additive error in the above definition will become a multiplicative one. Intuitively a view is plausible if non-feasibility of the view cannot be concluded by examining it in isolation from other views. The following lemma ensures that the plausibility of a view can be checked efficiently.

Lemma 4.3. *There is an $O((2c\delta^2)^\lambda \cdot \log_\delta(c\Delta))$ time algorithm to check the plausibility of a view.*

Proof. Let $V_x = (\deg_x, (A^b_x, A^d_x), \{(b^s_x, r^s_x)\}_{L \le s \le S})$ be a view at $x \in X$. For each $L \le s \le S$, we have $|\mathrm{Dom}(r^s_x)| \le (2c\delta^2)^\lambda$ (by Lemma 2.1). For each element in $\mathrm{Dom}(r^s_x)$ the plausibility condition can be checked in constant time. Therefore, the total running time for checking plausibility is $O((2c\delta^2)^\lambda \cdot (S - L)) = O((2c\delta^2)^\lambda \cdot \log_\delta(c\Delta))$.

The partition function. Although a view provides information only about the images of a relatively small subset of X, more can be deduced from it. Specifically, a view V_x at x uniquely determines the connected components of $T \backslash \{x\}$ for *every* feasible extension T of V_x (if any exists). Note that a priori it is not even clear that these connected components must be the same in different feasible extensions of V_x.

Lemma 4.4. *Let*

$$V_x = (\deg_x, (A^b_x, A^d_x), \{(b^s_x, r^s_x)\}_{L \le s \le S})$$

be any view at $x \in X$. There is an algorithm to compute a partition P of $X \backslash \{x\}$ in $O(n \log_\delta(c\Delta))$ time with the following property.

- *For every feasible extension T of V_x, and any $y, z \in X \backslash \{x\}$, y and z belong the the same connected component of $T \backslash \{x\}$ if and only if y and z belong to the same set of P.*

Proof. Let $y \in X$, and let s be the smallest scale such that b^s_x and r^s_x act on $nn_s(y)$, where s is well defined as b^S_x acts on all $X_{\ge S}$. We show that y and $nn_s(y)$ must belong to the same connected component of $T \backslash \{x\}$ in any feasible extension T of V_x. Note that since b^s_x acts on $nn_s(y)$, the connected component containing $nn_s(y)$ is specified by V_x, and so the lemma statement will then follow.

By the definition of $nn_s(y)$ and the feasibility of T, we have:

$$d_X(y, nn_s(y)) \leq \delta^s \Rightarrow d_T(y, nn_s(y)) \leq \delta^{s+1}.$$

Suppose, to derive a contradiction, that y and $nn_s(y)$ belong to different connected components of $T \backslash \{x\}$. That is, the path from y to $nn_s(y)$ in T contains x. Consequently,

$$d_T(x, y) \leq d_T(y, nn_s(y)) \leq \delta^{s+1}.$$

As T is feasible, it defines a non-contracting embedding. It follows that $d_X(x, y) \leq \delta^{s+1}$. So, by the triangle inequality, we have:

$$\begin{aligned} d_X(x, nn_{s-1}(y)) &\leq d_X(x, y) + d_X(y, nn_{s-1}(y)) \\ &\leq \delta^{s+1} + \delta^{s-1} \leq 2\delta^{s+1}. \end{aligned}$$

Therefore, b_x^{s-1} must act on $nn_{s-1}(y)$ (assuming $c \geq 2$), which is a contradiction with the assumption that s is the smallest scale for which b_x^s acts on $nn_s(y)$.

4.3 Consistency of views

Ultimately we wish to stitch together plausible views at different vertices to yield a feasible tree. To this end, in order to stitch together views they must have consistent descriptions of that tree. As a first step, we define when two plausible views can be stitched together over an edge.

Let $x, y \in X$. Let

$$V_x = (\deg_x, (A_x^b, A_x^d), \{(b_x^s, r_x^s)\}_{L \leq s \leq S})$$

and

$$V_y = (\deg_y, (A_y^b, A_y^d), \{(b_y^s, r_y^s)\}_{L \leq s \leq S})$$

be plausible views at x and y, respectively. Let $i \in \{1, 2, \ldots, \deg_x\}$, and $j \in \{1, 2, \ldots, \deg_y\}$. We say that V_x and V_y can be ***stitched*** together over (i, j) if the following consistency conditions hold.

(1) Either $A_x^b = i$ or $A_y^b = j$, but not both.

(2) For each $L \leq s \leq S$, and each $z \in \mathrm{Dom}(b_x^s) \cap \mathrm{Dom}(b_y^s)$ one of the following conditions hold

  (a) Either $b_x^s(z) = i$ or $b_y^s(z) = j$, but not both.

  (b1) If $b_x^s(z) = i$ and $A_x^b \neq i$ then $r_x^s(z) - r_y^s(z) = \lfloor A_y^d/\delta^s \rfloor - \lfloor A_x^d/\delta^s \rfloor$.

  (b2) If $b_x^s(z) = i$ and $A_x^b = i$ then $r_x^s(z) - r_y^s(z) = \lfloor A_x^d/\delta^s \rfloor - \lfloor A_y^d/\delta^s \rfloor$.

  (b3) If $b_x^s(z) \neq i$ and $A_x^b = i$ then $r_y^s(z) - r_x^s(z) = \lfloor A_x^d/\delta^s \rfloor - \lfloor A_y^d/\delta^s \rfloor$.

  (b4) If $b_x^s(z) \neq i$ and $A_x^b \neq i$ then $r_y^s(z) - r_x^s(z) = \lfloor A_y^d/\delta^s \rfloor - \lfloor A_x^d/\delta^s \rfloor$.

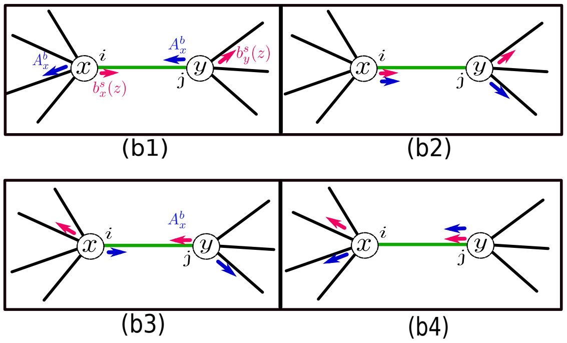

Figure 5: Consistency conditions (2)-(b1) to (2)-(b4).

The following lemma gives an algorithm to check consistency conditions for given V_x, V_y, i, and j.

Lemma 4.5. *Let*

$$V_x = (\deg_x, (A_x^b, A_x^d), \{(b_x^s, r_x^s)\}_{L \leq s \leq S})$$

and

$$V_y = (\deg_y, (A_y^b, A_y^d), \{(b_y^s, r_y^s)\}_{L \leq s \leq S})$$

be plausible views at x and y, respectively. Let $i \in \{1, 2, \ldots, \deg_x\}$, and $j \in \{1, 2, \ldots, \deg_y\}$. There is an $O((2c\delta^2)^{2\lambda} \cdot \log_\delta(c\Delta))$ time algorithm to check if V_x and V_y can be stitched together over (i, j).

Proof. Condition (1) can be checked in $O(1)$ time. For each $L \leq s \leq S$, we have $|\mathrm{Dom}(b_x^s)| \leq (2c\delta^2)^\lambda$ and $|\mathrm{Dom}(b_y^s)| \leq (2c\delta^2)^\lambda$. Therefore, their intersection can be computed in $O((2c\delta^2)^{2\lambda})$ time. For each element in the intersection, conditions (a) and (b1) through (b4) can be checked in constant time. Therefore, the total running time for checking condition (2) is $O((2c\delta^2)^{2\lambda} \cdot (S - L)) = O((2c\delta^2)^{2\lambda} \cdot \log_\delta(c\Delta))$.

Consistent set of views. The images of two points $x, y \in X$ can be adjacent in an optimal tree only if there are plausible views at each of them that can be stitched together. By stitching together pairs of plausible views one at a time, our algorithm builds a tree T over X, and an accompanying collection of views for each vertex in X that can be stitched together over the edges of T. We call such a collection a consistent set over T, and formally define it as follows.

Let $\mathcal{V}$ be a set of plausible views, one at each vertex of X, and let $T = (X, E_T, w_T)$ be a tree. We say that $\mathcal{V}$ is a ***consistent set*** of views over T if there are bijections $\ell_x : \mathrm{adj}(x) \rightarrow \{1, 2, \ldots, \deg_x\}$ for all $x \in X$ with the following properties.

(1) For each $e = (x, y) \in E_T$ and the corresponding views $V_x, V_y \in \mathcal{V}$, we have that V_x and V_y

are consistent over $(\ell_x(e), \ell_y(e))$, and $w_T(e) = |A^d_x - A^d_y|$.

(2) For each x and its corresponding view $V_x = (\deg_x, (A^b_x, A^d_x), \{(b^s_x, r^s_x)\}_{L \le s \le S})$, the following three conditions are equivalent (i) $A^b_x = null$, (ii) $A^d_x = 0$, and (iii) $x = \mathsf{a}$.

4.4 A consistent set is all we need Ultimately our algorithm will attempt to grow a nearly optimal tree from the anchor a using dynamic programming. The space of possible trees over X is far too large to be explored. Thus additionally we guess a view at each vertex as we go, as this will severely limit the possibilities for the subtrees. To this end, in this section we show that limiting to the space of trees with such views is valid. That is, if any set of views with parameters $(\mathsf{deg}, \sigma, c, \delta)$ are consistent over a tree T then it implies the distortion of the embedding defined by T is close to δ (how close depends on c and σ, and is specified below). Thus any set of consistent views will suffice. Moreover, we first show, by considering the restriction of any feasible embedding, that at least one consistent set must exist.

Lemma 4.6. *Let (X, d_X) be a metric space, and let $\delta_{opt} = \delta_{opt}(X, \mathsf{deg})$ be the optimal distortion for embedding X into a tree of max degree at most deg. For any $0 < \sigma \le 1$ and any $c \ge 1$, there exists a set of plausible views $\mathcal{V}$ with parameters $(\mathsf{deg}, \sigma, c, (1+\sigma)\delta_{opt})$, and a tree $T = (X, E_T, w_T)$ of maximum degree deg such that $\mathcal{V}$ is consistent over T.*

Proof. Let $T' = (X, E_{T'}, w_{T'})$ be a tree, into which X can be embedded with distortion δ_{opt}. By Lemma 4.1, there is a tree $T = (X, E_T, w_T)$ whose edge weights are multiples of σ, into which T' can be embedded with distortion $1+\sigma$. Therefore, X can be embedded into T with distortion $(1+\sigma) \cdot \delta_{opt}$. Suppose after relabeling the vertices of T that the identity map from (X, d_X) to (X, d_T) has distortion $(1+\sigma) \cdot \delta_{opt}$. For each $x \in X$, let V_x be the restricted view of this identity map at x in T with parameters deg, σ, c, and $\delta = (1+\sigma) \cdot \delta_{opt}$. Let $\mathcal{V} = \bigcup_{x \in X} V_x$. We show that $\mathcal{V}$ is a set of consistent views over T.

First, for any x, we show that V_x is plausible. By the definition of restriction we have:

$$r^s_x(y) = \lfloor d_T(x, \mathsf{a})/\delta^s \rfloor + \lfloor d_T(y, \mathsf{a})/\delta^s \rfloor - 2\lfloor d_T(u, \mathsf{a})/\delta^s \rfloor,$$

where u is the closest vertex to a on the x-to-y path. Removing the floors we obtain:

$$\frac{d_T(x,\mathsf{a}) + d_T(y,\mathsf{a}) - 2d_T(u,\mathsf{a})}{\delta^s} - 2$$
$$\le r^s_x(y) \le \frac{d_T(x,\mathsf{a}) + d_T(y,\mathsf{a}) - 2d_T(u,\mathsf{a})}{\delta^s} + 2$$

The definition of u implies $d_T(x,y) = d_T(x,\mathsf{a}) + d_T(y,\mathsf{a}) - 2d_T(u,\mathsf{a})$, therefore, we obtain:

$$d_T(x,y) - 2\delta^s \le r^s_x(y) \cdot \delta^s \le d_T(x,y) + 2\delta^s.$$

Since the embedding into T is feasible, i.e. non-contracting with expansion at most δ, we conclude

$$d_X(x,y) - 2\delta^s \le r^s_x(y) \cdot \delta^s \le \delta d_X(x,y) + 2\delta^s.$$

Next, we show the mutual consistency between these sets of restricted views. Let $x, y \in X$, and let $V_x = (\deg_x, (A^b_x, A^d_x), \{(b^s_x, r^s_x)\}_{L \le s \le S})$ and $V_y = (\deg_y, (A^b_y, A^d_y), \{(b^s_y, r^s_y)\}_{L \le s \le S})$ be restricted views of the identity map around x and y in T, respectively. Also, let ℓ_x and ℓ_y be the labeling functions induced by the restrictions to x and y. Suppose, $e = (x,y) \in E_T$. We show that V_x and V_y can be stitched together over $\ell_x(e)$ and $\ell_y(e)$ with weight $w_T(e)$. Condition (1) and (2-a) of consistency are implied by the restriction definition, items (3-a) and (3-b-i). It remains to show that conditions (2-b1) through (2-b4) hold. For any $L \le s \le S$ and any $z \in \mathrm{Dom}(b^s_x) \cap \mathrm{Dom}(b^s_y)$, we have

$$r^s_x(z) = \lfloor d_T(x,\mathsf{a})/\delta^s \rfloor + \lfloor d_T(z,\mathsf{a})/\delta^s \rfloor - 2\lfloor d_T(u_{x,z},\mathsf{a})/\delta^s \rfloor$$

and

$$r^s_y(z) = \lfloor d_T(y,\mathsf{a})/\delta^s \rfloor + \lfloor d_T(z,\mathsf{a})/\delta^s \rfloor - 2\lfloor d_T(u_{y,z},\mathsf{a})/\delta^s \rfloor$$

where $u_{x,z}$ is the closest vertex of the x-to-z path to a, and $u_{y,z}$ is the closest vertex of the y-to-z path to a. We consider conditions (2-b1) through (2-b4) (looking at Figure 5 while reading the following cases may help the reader). Let $i = \ell_x(e)$ for the following case analysis.

Case (2-b1) or (2-b3): We have $b^s_x(z) = i$ and $A^b_x \ne i$, or, $b^s_x(z) \ne i$ and $A^b_x = i$. In both cases, we have $u_{x,z} = x$ and $u_{y,z} = y$. Therefore,

$$\begin{aligned} r^s_x(z) - r^s_y(z) &= \lfloor d_T(y,\mathsf{a})/\delta^s \rfloor - \lfloor d_T(x,\mathsf{a})/\delta^s \rfloor \\ &= \lfloor A^d_y/\delta^s \rfloor - \lfloor A^d_x/\delta^s \rfloor. \end{aligned}$$

Case (2-b2) or (2-b4): we have $b^s_x(z) = i$ and $A^b_x = i$, or, $b^s_x(z) \ne i$ and $A^b_x \ne i$. In both cases, it is implied that $u_{x,z} = u_{y,z}$. Therefore,

$$\begin{aligned} r^s_x(z) - r^s_y(z) &= \lfloor d_T(x,\mathsf{a})/\delta^s \rfloor - \lfloor d_T(y,\mathsf{a})/\delta^s \rfloor \\ &= \lfloor A^d_x/\delta^s \rfloor - \lfloor A^d_y/\delta^s \rfloor. \end{aligned}$$

In both cases, the last equality holds because weights of T are integer multiples of σ.

Next, we show that a consistent set over a tree guarantees near optimal distortion. To that end, we need a few definitions and helper lemmas.

Let $T = (X, E_T, w_T)$ be a tree, and let $\gamma = (v_1, \ldots, v_k)$ be a path in T. We say that γ is ***approaching*** if $d_T(v_1, \mathsf{a}) \geq d_T(v_2, \mathsf{a}) \geq \ldots \geq d_T(v_k, \mathsf{a})$. We say that γ is ***departing*** if $d_T(v_1, \mathsf{a}) \leq d_T(v_2, \mathsf{a}) \leq \ldots \leq d_T(v_k, \mathsf{a})$. Finally, we say that γ is ***monotone*** if it is approaching or departing. Note that any (simple) path γ can be decomposed into $\gamma^- \circ \gamma^+$, such that γ^- is approaching and γ^+ is departing. We show that very accurate information can be deduced from the views of two vertices x and y if the path between them in T is approaching or departing.

Lemma 4.7. *Let $\mathcal{V}$ be a consistent set of views over $T = (X, E_T, w_T)$. Let $x, y \in X$, and let V_x and V_y be the views at x and y, respectively. Finally, let γ be the unique x-to-y path in T. If γ is approaching then $d_T(x, y) = A^d_x - A^d_y$, and if γ is departing then $d_T(x, y) = A^d_y - A^d_x$.*

Proof. Let $\gamma = (x = v_1, \ldots, v_k = y)$. Suppose γ is approaching, the other case is similar. We use induction on k to prove the statement. If $k = 1$ then $x = y$, and the statement trivially holds. If $k > 1$, let $t = v_{k-1}$, and let $V_t \in \mathcal{V}$ be the view at t. By the induction hypothesis, $d_T(x, t) = A^d_x - A^d_t$. Since γ is approaching, A^b_t points to the edge (t, y), and thus since V_t and V_y are consistent over the edge (t, y) in T, we have that $d_T(t, y) = w_T(t, y) = A^d_t - A^d_y$. Overall,

$$\begin{aligned} d_T(x, y) &= d_T(x, t) + w_T(t, y) \\ &= (A^d_x - A^d_t) + (A^d_t - A^d_y) = A^d_x - A^d_y \end{aligned}$$

Lemma 4.8. *Let $\mathcal{V}$ be a consistent set of views over $T = (X, E_T, w_T)$. Let $x, y, z \in X$, and let V_x and V_y be the views at x and y, respectively. Finally, let γ be the unique x-to-y path in T. Suppose γ is monotone and z is in the same connected component with either x or y in $T \backslash (\gamma \backslash \{x, y\})$. If a and z belong to the same connected component of $T \backslash (\gamma \backslash \{x, y\})$ then $r^s_x(z) - r^s_y(z) = \lfloor A^d_x/\delta^s \rfloor - \lfloor A^d_y/\delta^s \rfloor$. Otherwise, $r^s_x(z) - r^s_y(z) = \lfloor A^d_y/\delta^s \rfloor - \lfloor A^d_x/\delta^s \rfloor$.*

Proof. Let $\gamma = (x = v_1, \ldots, v_k = y)$. We use induction on k to prove the statement. If $k = 1$ then $x = y$, and the statement trivially holds. If $k > 1$, let $t = v_{k-1}$, and let $V_t \in \mathcal{V}$ be the view at t. Note that in the lemma statement we assume that z is in the connected component of either x or y in $T \backslash (\gamma \backslash \{x, y\})$, and so z is in the connected component of x or t in $T \backslash (\gamma[x, t] \backslash \{x, t\})$ First, consider the case that a and z belong to the same connected component of $T \backslash (\gamma \backslash \{x, y\})$. By the induction hypothesis,

$$r^s_x(z) - r^s_t(z) = \lfloor A^d_x/\delta^s \rfloor - \lfloor A^d_t/\delta^s \rfloor. \tag{4.1}$$

Since V_t and V_y are consistent over the edge (t, y) in T, and A^b_t and $b^s_t(z)$ are the same, one of conditions (2-b2) or (2-b4) holds. In either case,

$$r^s_t(z) - r^s_y(z) = \lfloor A^d_t/\delta^s \rfloor - \lfloor A^d_y/\delta^s \rfloor.$$

Substituting in Equation (4.1) we obtain the lemma statement.

Next, consider the case that a and z belong to different connected components of $T \backslash (\gamma \backslash \{x, y\})$. By the induction hypothesis,

$$r^s_x(z) - r^s_t(z) = \lfloor A^d_t/\delta^s \rfloor - \lfloor A^d_x/\delta^s \rfloor. \tag{4.2}$$

Since V_t and V_y are consistent over the edge (t, y) in T, and A^b_t and $b^s_t(z)$ are different, one of conditions (2-b1) or (2-b3) holds. In either case,

$$r^s_t(z) - r^s_y(z) = \lfloor A^d_y/\delta^s \rfloor - \lfloor A^d_t/\delta^s \rfloor.$$

Substituting in Equation (4.2) we obtain the lemma statement.

Next, we show that the distance estimators in the views provide relatively accurate estimations for the distance of visible vertices in T.

Lemma 4.9. *Let $\mathcal{V}$ be a consistent set of views over $T = (X, E_T, w_T)$. Let $x, z \in X$, $V_x \in \mathcal{V}$ be the view at x, and $L \leq s \leq S$. If z is visible in V_x at scale s then*

$$d_T(x, z) - 4\delta^s \leq r^s_x(z) \cdot \delta^s \leq d_T(x, z) + 4\delta^s$$

Proof. Let $\gamma = (x = v_1, \ldots, v_k = z)$ be the unique x-to-z path in T. As noted above, γ can be decomposed into two subpaths $\gamma^- = (v_1, \ldots, v_j = y)$, and $\gamma^+ = (y = v_j, v_{j+1}, \ldots, v_k)$ such that γ^- is approaching, and γ^+ is departing. Thus y is the closest point of γ to the anchor point in T. By Lemma 4.7, we have

$$d_T(x, y) = A^d_x - A^d_y, \quad \& \quad d_T(y, z) = A^d_z - A^d_y.$$

Therefore,

$$d_T(x, z) = A^d_z + A^d_x - 2A^d_y. \tag{4.3}$$

On the other hand, by Lemma 4.8 we know

$$\begin{aligned} r^s_x(z) - r^s_y(z) &= \lfloor A^d_x/\delta^s \rfloor - \lfloor A^d_y/\delta^s \rfloor, \\ \& \quad r^s_y(z) - r^s_z(z) &= \lfloor A^d_z/\delta^s \rfloor - \lfloor A^d_y/\delta^s \rfloor. \end{aligned}$$

Consequently,

$$(4.4)\quad r^s_x(z) - r^s_z(z) = \lfloor A^d_x/\delta^s \rfloor + \lfloor A^d_z/\delta^s \rfloor - 2\lfloor A^d_y/\delta^s \rfloor.$$

To obtain the desired statement, we combine Equations (4.3) and (4.4), while noting that the definition of plausible views implies $r^s_z(z) \in \{-2, -1, 0, 1, 2\}$ (as $d_X(z,z) = 0$). First we show the upper bound for $r^s_x(z) \cdot \delta^s$.

$$\begin{aligned} r^s_x(z)\delta^s &= \lfloor A^d_x/\delta^s\rfloor\delta^s + \lfloor A^d_z/\delta^s\rfloor\delta^s - 2\lfloor A^d_y/\delta^s\rfloor\delta^s + r^s_z(z)\delta^s \\ &\le (A^d_x/\delta^s)\delta^s + (A^d_z/\delta^s)\delta^s - 2(A^d_y/\delta^s - 1)\delta^s + 2\delta^s \\ &= A^d_x + A^d_z - 2A^d_y + 2\delta^s + 2\delta^s \\ &\le d_T(x,z) + 4\delta^s. \end{aligned}$$

Next, we show the lower bound for $r^s_x(z) \cdot \delta^s$.

$$\begin{aligned} r^s_x(z)\delta^s &= \lfloor A^d_x/\delta^s\rfloor\delta^s + \lfloor A^d_z/\delta^s\rfloor\delta^s - 2\lfloor A^d_y/\delta^s\rfloor\delta^s + r^s_z(z)\delta^s \\ &\ge (A^d_x/\delta^s - 1)\delta^s + (A^d_z/\delta^s - 1)\delta^s - 2(A^d_y/\delta^s)\delta^s - 2\delta^s \\ &= A^d_x - \delta^s + A^d_z - \delta^s - 2A^d_y - 2\delta^s \\ &\ge d_T(x,z) - 4\delta^s \end{aligned}$$

Now, we are ready to bound the distortion of distances on T. First, we show that this distortion is bounded for a pair of vertices if one is visible under the view at the other one.

LEMMA 4.10. *Let $\mathcal{V}$ be consistent set of views over $T = (X, E_T, w_T)$, let $x, z \in X$, and let $V_x \in \mathcal{V}$ be the view at x. If z is visible in V_x then*

$$\left(1 - \frac{6}{c}\right) \cdot d_X(x,z) \le d_T(x,z) \le \left(1 + \frac{6}{c}\right) \cdot \delta \cdot d_X(x,z).$$

Proof. Note that if $x = z$ the lemma statement trivially holds, thus, assume otherwise. Let $L \le s \le S$ be the smallest scale such that z is visible at scale s in V_x. Since z is visible, and V_x is plausible, we have

$$d_X(x,z) - 2\delta^s \le r^s_x(z) \cdot \delta^s \le \delta d_X(x,z) + 2\delta^s.$$

Also, by Lemma 4.9,

$$d_T(x,z) - 4\delta^s \le r^s_x(z) \cdot \delta^s \le d_T(x,z) + 4\delta^s.$$

Consequently, we have,

$$(4.5)\quad d_X(x,z) - 6\delta^s \le d_T(x,z) \le \delta d_X(x,z) + 6\delta^s.$$

On the other hand, since z is not visible at scale $s-1$, we have

$$c\delta^{s+1} < d_X(x,z) \Rightarrow \delta^s \le \frac{d_X(x,z)}{c}.$$

Substituting in Equation (4.5) we obtain

$$\begin{aligned} \left(1 - \frac{6}{c}\right) \cdot d_X(x,z) \le d_X(x,z) - 6 \cdot \delta^s \le d_T(x,z) \\ \le \delta d_X(x,z) + 6 \cdot \delta^s \le \left(1 + \frac{6}{c}\right) \cdot \delta \cdot d_X(x,z). \end{aligned}$$

Finally, we bound the distortion for any pair of vertices, even if they are not visible under each other's views.

LEMMA 4.11. *Let $\mathcal{V}$ be a consistent set of views over $T = (X, E_T, w_T)$, and let $x, z \in X$. We have,*

$$\left(1 - \frac{14}{c-1}\right) d_X(x,z) \le d_T(x,z) \le \left(1 + \frac{20}{c-1}\right) \delta d_X(x,z).$$

Proof. Note that if $x = z$ the lemma statement trivially holds, thus, assume otherwise. Let s be the smallest scale such that $nn_s(z)$ is visible at V_x, where s is well defined as b^S_x acts on all $X_{\ge S}$. First, we show that $d_X(z, nn_s(z))$ is small compared to $d_X(x,z)$. By the definition of the scale nearest neighbors we have,

$$(4.6)\quad d_X(z, nn_s(z)) \le \delta^s \ \ \& \ \ d_X(z, nn_{s-1}(z)) \le \delta^{s-1}.$$

Since $nn_{s-1}(z)$ is not visible at V_x we have,

$$d_X(x, nn_{s-1}(z)) \ge c \cdot \delta^{s+1},$$

and therefore,

$$\begin{aligned} d_X(x,z) &\ge d_X(x, nn_{s-1}(z)) - d_X(z, nn_{s-1}(z)) \\ &\ge c\delta^{s+1} - \delta^{s-1} \ge (c-1)\delta^{s+1}. \end{aligned}$$

Combining with (4.6) we obtain,

$$\begin{aligned} &\frac{d_X(z, nn_s(z))}{d_X(x,z)} \le \frac{\delta^s}{(c-1)\delta^{s+1}} \\ (4.7)\qquad &\Rightarrow d_X(z, nn_s(z)) \le \frac{d_X(x,z)}{(c-1)\delta}. \end{aligned}$$

Now we use Inequality (4.7) and Lemma 4.10 to show bounds for the $d_T(x,z)$. By our assumption $nn_s(z)$ is visible in V_x. Furthermore, as $d_X(z, nn_s(z)) \le \delta^s$, by the definition of nets, $nn_s(z)$ is visible in V_z. First, we show the upper bound.

$$\begin{aligned} d_T(x,z) &\le d_T(x, nn_s(z)) + d_T(nn_s(z), z) \\ &\le \left(1 + \frac{6}{c}\right) \cdot \delta \cdot (d_X(x, nn_s(z)) + d_X(nn_s(z), z)) \\ &\le \left(1 + \frac{6}{c}\right) \cdot \delta \cdot (d_X(x,z) + 2d_X(nn_s(z), z)) \\ &\le \left(1 + \frac{6}{c}\right) \cdot \delta \cdot \left(d_X(x,z) + \frac{2d_X(x,z)}{(c-1)\delta}\right) \\ &\le \left(1 + \frac{6}{c}\right) \cdot \delta \cdot \left(1 + \frac{2}{c-1}\right) \cdot d_X(x,z) \\ &\le \left(1 + \frac{20}{c-1}\right) \cdot \delta \cdot d_X(x,z) \end{aligned}$$

The inequalities above follow by triangle inequality, Lemma 4.10, triangle inequality, Inequality (4.7), and since $\delta \geq 1$ in order. Next, we show the lower bound.

$$\begin{aligned}
&d_T(x, z) \geq d_T(x, nn_s(z)) - d_T(nn_s(z), z)\\
&\geq \frac{c-6}{c} \cdot d_X(x, nn_s(z)) - \frac{(c+6)\delta}{c} \cdot d_X(nn_s(z), z)\\
&\geq \frac{c-6}{c} \cdot (d_X(x, z) - d_X(z, nn_s(z)))\\
&- \frac{(c+6)\delta}{c} \cdot d_X(nn_s(z), z)\\
&\geq \frac{c-6}{c} \cdot d_X(x, z) - \left(\frac{(c+6)\delta^2}{c} + \frac{c-6}{c}\right) \cdot \frac{d_X(x, z)}{(c-1)\delta}\\
&\geq \left(1 - \frac{14}{c-1}\right) \cdot d_X(x, z)
\end{aligned}$$

The inequalities follow from Lemma 4.10, Inequality (4.7) and the triangle inequality.

4.5 Dynamic programming

In the previous section we defined the notion of a consistent set $\mathcal{V}$ of plausible views over the entire set X. This definition can be naturally extended to views over subsets of X. Specifically, let $Y \subseteq X$, let $x \in Y$, let $\mathcal{V}$ be a set of plausible views at the vertices of Y, and let $T = (Y, E_T, w_T)$ be a positively weighted tree. We say that $\mathcal{V}$ is a ***consistent set*** over subtree T with root x if there are bijections for each $y \in Y \setminus \{x\}$, $\ell_y : adj(y) \to \{1, \ldots, \mathsf{deg}_y\}$, and a bijection $\ell_x : adj(x) \to \{1, \ldots, \mathsf{deg}_x\} \backslash \{A^b_x\}$, such that:

(1) For each edge $e = (u, v) \in E_T$ and the corresponding views $V_u, V_v \in \mathcal{V}$, we have that V_u and V_v are consistent over $(\ell_u(e), \ell_v(e))$, and $w_T(e) = |A^d_u - A^d_v|$.

(2) For any view $V_u \in \mathcal{V}$, the following three conditions are equivalent (i) $A^b_u = null$, (ii) $A^d_u = 0$, and (iii) $u = x = \mathsf{a}$.

Comparing with our previous definition of consistency, observe that $\mathcal{V}$ is a consistent set over subtree T with root a, if and only if $\mathcal{V}$ is a consistent set over tree T.

Lemma 4.12. *Let X be an n-point metric space, with doubling dimension λ, and spread Δ. For any $\delta > 0$, $\epsilon > 0$, and $\mathsf{deg} > 0$ there is a*

$$n^2 \cdot \left(\frac{\Delta}{\epsilon}\right)^{\log_\delta(\mathsf{deg}/\epsilon)(O(\delta^2/\epsilon))^\lambda}$$

time algorithm to compute a $(1+\epsilon)\delta$ distortion embedding of X into a tree T of maximum degree deg if $\delta \geq \delta_{opt}(X, \mathsf{deg})$. If $\delta < \delta_{opt}(X, \mathsf{deg})$, this algorithm either computes an embedding of distortion $(1+\epsilon)\delta$ or (correctly) decides that $\delta < \delta_{opt}(X, \mathsf{deg})$.

Proof. First suppose that $\delta \geq \delta_{opt}(X, \mathsf{deg})$. Lemma 4.6 then guarantees the existence of a set of views, one for each $x \in X$, with parameters $(\mathsf{deg}, \sigma, c, (1+\sigma)\delta)$ that are consistent over some tree T. (Since if there is $(\mathsf{deg}, \sigma, c, (1+\sigma)\delta_{opt})$ set of consistent views then there is a $(\mathsf{deg}, \sigma, c, (1+\sigma)\delta)$ set of consistent views). Moreover, Lemma 4.11 implies that if there is a consistent set of views over some T with parameters $(\mathsf{deg}, \sigma, c, (1+\sigma)\delta)$ then T defines an embedding with distortion at most

$$\begin{aligned}
&\left(\left(1 + \frac{20}{c-1}\right) \Big/ \left(1 - \frac{14}{c-1}\right)\right) \cdot (1+\sigma) \cdot \delta\\
&\leq \left(1 + \frac{20}{c-1}\right) \cdot \left(1 + \frac{28}{c-1}\right) \cdot (1+\sigma) \cdot \delta\\
&= \left(1 + \frac{20}{c-1} + \frac{28}{c-1} + \frac{560}{(c-1)^2}\right) \cdot (1+\sigma) \cdot \delta\\
&\leq \left(1 + \frac{608}{c-1}\right) \cdot (1+\sigma) \cdot \delta,
\end{aligned}$$

which is at most $(1+\epsilon)\delta$ for $c = \frac{3 \times 608}{\epsilon} + 1$ and $\sigma = \epsilon/3$.[7] So set c and σ to these values and let $\delta' = (1+\sigma)\delta$. Then the above two statements combined imply that to prove the lemma, it suffices to give an algorithm which finds any consistent set of views (over some tree) with parameters $(\mathsf{deg}, \sigma, c, \delta')$, if one exists, and otherwise returns $\delta < \delta_{opt}(X, \mathsf{deg})$. We now describe a recursive algorithm which we then memoize to compute such a set of views.

Consider any collection $\mathcal{V}$ of views, with exactly one view V_x for each $x \in X$, with parameters $(\mathsf{deg}, \sigma, c, \delta')$. Given a weighted tree T on vertex set X, we first consider the simpler task of checking whether $\mathcal{V}$ is consistent over T. As discussed above, this is equivalent to saying that $\mathcal{V}$ is consistent over subtree T with root a. Let $T_{\mathsf{a}} = T$ and $\mathcal{V}_{\mathsf{a}} = \mathcal{V}$, and for any $x \neq \mathsf{a}$ in X, let T_x denote the subtree rooted at x and not containing a, and similarly let $\mathcal{V}_x$ be the subset of views over the vertices in this subtree. Also, let $adj'(x)$ denote all neighbors of x other than the one on the path to a, that is $adj'(x)$ are the neighbors of x in T_x (note $adj'(x) = adj(x)$ if $x = \mathsf{a}$). Observe that for any $x \in X$, $\mathcal{V}_x$ is a consistent set of views over subtree T_x with root x if and only if for every $y \in adj'(x)$ (1) $\mathcal{V}_y$ is a consistent set of views over subtree T_y with root y, and (2) V_x and V_y are consistent over $e = (x, y)$ with $w_{T_x}(x, y) = |A^d_x - A^d_y|$. Note that this is a recursive statement. Thus to check consistency of $\mathcal{V}_x$ over T_x, condition (1) can be checked by recursion, where the base case is when

[7] Note that c can be made significantly smaller, however, in this paper to keep the calculations readable we are not optimizing constants.

$adj'(x) = \emptyset$, and condition (2) can be checked by the algorithm of Lemma 4.5. To check if the full set $\mathcal{V}$ is consistent over T, we apply this recursive algorithm with $x = \mathsf{a}$.

This immediately implies a recursive algorithm for the harder problem of determining whether there exists any such collection of views $\mathcal{V}$ consistent over some tree T. Namely, consider all possible views with parameters $(\mathsf{deg}, \sigma, c, \delta')$ that are centered at the anchor a. For each such view V_{a}, we recursively determine if there is a collection of views containing V_{a}, which is consistent over a subtree T with root a. To do so consider all possible partitions of $X \setminus \{\mathsf{a}\}$ into deg_{a} subsets (i.e. subtrees). Then for each subset Z in a given partition we try all possible views over all members in Z as the root view, and for each such view V_x, if the view is consistent with V_{a} over the edge (a, x) (note the weight of the edge will then be $|A^d_{\mathsf{a}} - A^d_x|$), we then recursively check whether there is a collection of views over Z containing V_x, which is consistent over any subtree T_Z with root x. The correctness of this approach is apparent from the discussion above, however, the running time is exponential. Specifically, Lemma 4.2 bounds the number of possible views we must consider, but remembering the subsets and guessing how they are partitioned takes exponential time. However, we can now make use of Lemma 4.4, which states that for any view V_x, one can compute the unique partition $P = \{p_1, \ldots, p_{deg_x}\}$ of $X \setminus \{x\}$, such that if there is a feasible extension of V_x to an embedding defined by a tree T, then the sets in P must be the sets of vertices form each component of $T \setminus \{x\}$. Thus if x is a root with a view V_x over some subset Z then we can assume $Z = \cup_{i \in \{1,2,\ldots,\deg_x\} \setminus \{A^b_x\}} \, p_i$, and thus Z does not need to be passed as a parameter to the recursive problem. Moreover, rather than guessing all possible partitions of Z in this subproblem, we just use the partition $P \setminus \{p_{A^b_x}\}$.

Each subproblem of this recursive procedure is defined by a root $x \in X$ and a view V_x. Thus we can setup a dynamic programming table, index by (x, V_x) pairs, and then fill the table using the above recursive procedure and memoization.

For the running time, there are n choices for x, and $(1/\sigma) \cdot (c\Delta)^{O(\log_{\delta'}(c \cdot \mathsf{deg}))(2c(\delta')^2)^\lambda}$ choices for V_x by Lemma 4.2. Thus, the size of the table is

$$\frac{n}{\sigma} \cdot (c\Delta)^{O(\log_{\delta'}(c \cdot \mathsf{deg}))(2c(\delta')^2)^\lambda}.$$

Since we use memoization, each table entry is filled only once. For each table entry, we first compute its partition, and then independently for each (non-anchor) subset in the partition, and for each view V_z at a member z in the subset we check if V_x and V_z are consistent (which itself includes plausibility checks), and if so check the table entry (z, V_z). Ignoring the time spent in recursive calls, our algorithm thus spends at most

$$\begin{aligned} &O(n \log_{\delta'}(c\Delta)) + \left(\mathsf{deg}\frac{n}{\sigma}(c\Delta)^{O(\log_{\delta'}(c \cdot \mathsf{deg}))(2c(\delta')^2)^\lambda}\right) \\ &\cdot O((2c(\delta')^2)^{2\lambda} \cdot \log_{\delta'}(c\Delta)) \\ &= \frac{n}{\sigma} \cdot (c\Delta)^{O(\log_{\delta'}(c \cdot \mathsf{deg}))(2c(\delta')^2)^\lambda} \end{aligned}$$

time per table entry, where the first term is the time it takes to compute the partition (Lemma 4.4), and last part of the second term is the time to check for a pair of views whether each is plausible and whether they are consistent (Lemma 4.3 and Lemma 4.5). Therefore, the total running time of the algorithms is

$$\begin{aligned} &\left(\frac{n}{\sigma} \cdot (c\Delta)^{O(\log_{\delta'}(c \cdot \mathsf{deg}))(2c(\delta')^2)^\lambda}\right) \\ &\cdot \left(\frac{n}{\sigma} \cdot (c\Delta)^{O(\log_{\delta'}(c \cdot \mathsf{deg}))(2c(\delta')^2)^\lambda}\right) \\ &= \frac{n^2}{\sigma^2} \cdot (c\Delta)^{O(\log_{\delta'}(c \cdot \mathsf{deg}))(2c(\delta')^2)^\lambda} \end{aligned}$$

As discussed above, in order to obtain a $1 + \epsilon$ approximation, we had set $c = O(\frac{1}{\epsilon})$, $\sigma = O(\epsilon)$, and $\delta' = (1 + \sigma)\delta$, and thus the running time of our algorithms is

$$\frac{n^2}{\epsilon^2}\left(\frac{\Delta}{\epsilon}\right)^{\log_\delta(\mathsf{deg}/\epsilon)(O(\delta^2/\epsilon))^\lambda} = n^2\left(\frac{\Delta}{\epsilon}\right)^{\log_\delta(\mathsf{deg}/\epsilon)(O(\delta^2/\epsilon))^\lambda}$$

Theorem 4.1. *Let X be an n-point metric space, with doubling dimension λ and spread Δ. For any $0 < \varepsilon < 1$ and $\mathsf{deg} > 1$, where $\delta_{opt} = \delta_{opt}(X, \mathsf{deg})$, there is an algorithm with running time*

$$n^2 \cdot \left(\frac{\Delta}{\varepsilon}\right)^{\log(\mathsf{deg}/\varepsilon) \cdot (1/\varepsilon) \cdot (O(\delta^2_{opt}/\varepsilon))^\lambda}$$

to compute a $(1 + \varepsilon)\delta_{opt}$ distortion embedding of X into a tree T of maximum degree deg.

Proof. Consider the set $L = \{\delta_i = (1 + \varepsilon/2)^i | 1 \leq i \leq n\Delta\}$. Our algorithm calls the procedure of Lemma 4.12 with $\epsilon = \varepsilon/3$ and $\delta = \delta_i$, in increasing order of δ_i, until it first successfully finds an embedding. Note that since X always embeds into a path with distortion at most $n\Delta$, our algorithm will always find an embedding. To bound the distortion of the computed embedding, let $1 \leq j \leq n\Delta$ be such that $(1 + \varepsilon/2)^{j-1} \leq \delta_{opt}(X, \mathsf{deg}) \leq (1 + \varepsilon/2)^j$. Then the procedure of Lemma 4.12 will return an embedding of distortion at most $(1 + \varepsilon/2)^j \cdot (1 + \varepsilon/3)$ if it is called with parameters $\delta = \delta_j = (1+\varepsilon/2)^j$ and $\epsilon = \varepsilon/3$. Thus we get an embedding with distortion at most

$$(1+\varepsilon/2)^j(1+\varepsilon/3) \le \delta_{opt}(1+\varepsilon/2)(1+\varepsilon/3) \le \delta_{opt}(1+\varepsilon).$$

It remains to bound the running time of our algorithm. We call the procedure of Lemma 4.12 $O(\log_{1+\varepsilon/2}(\delta_{opt}))$ times. The running time of each of these procedure calls is bounded by

$$n^2 \cdot \left(\frac{\Delta}{\epsilon}\right)^{\log_\delta(\mathsf{deg}/\epsilon)(O(\delta^2/\epsilon))^\lambda} = \tag{4.8} n^2 \cdot \left(\frac{\Delta}{\varepsilon}\right)^{\log_{1+\varepsilon/2}(3\cdot\mathsf{deg}/\varepsilon)(O(\delta^2/\varepsilon))^\lambda}.$$

We know via Taylor series expansion, that for $0 \le x \le 1$, $\log(1+x) \ge x - x^2/2 = (x/2)(2-x) \ge x/2$. Therefore since $0 < \varepsilon < 1$,

$$\log_{1+\varepsilon/2}(3\cdot\mathsf{deg}/\varepsilon) = \frac{\log(3\cdot\mathsf{deg}/\varepsilon)}{\log(1+\varepsilon/2)} \le \frac{\log(3\cdot\mathsf{deg}/\varepsilon)}{\varepsilon/2} = O\left(\frac{\log(\mathsf{deg}/\varepsilon)}{\varepsilon}\right)$$

Substituting in (4.8) we find out that the running time of each call is bounded by

$$n^2 \cdot \left(\frac{\Delta}{\varepsilon}\right)^{\log_{1+\varepsilon/2}(3\cdot\mathsf{deg}/\varepsilon)(O(\delta^2/\varepsilon))^\lambda} = n^2 \cdot \left(\frac{\Delta}{\varepsilon}\right)^{\log(\mathsf{deg}/\varepsilon)\cdot(1/\varepsilon)\cdot(O(\delta^2/\varepsilon))^\lambda}$$

Thus the total running time is

$$O(\log_{1+\varepsilon/2}(\delta_{opt}))\cdot n^2 \cdot \left(\frac{\Delta}{\varepsilon}\right)^{\log(\mathsf{deg}/\varepsilon)\cdot(1/\varepsilon)\cdot(O(\delta^2/\varepsilon))^\lambda} = n^2 \cdot \left(\frac{\Delta}{\varepsilon}\right)^{\log(\mathsf{deg}/\varepsilon)\cdot(1/\varepsilon)\cdot(O(\delta_{opt}^2/\varepsilon))^\lambda}$$

Corollary 5.1 in the next section shows that any tree with doubling dimensions λ can be embedded with $(1+\varepsilon)$ distortion into a tree with maximum degree $(O(1/\varepsilon))^\lambda$. Thus the above lemma statement can be strengthened to remove the degree assumption, yielding Theorem 1.1.

Proof. [Proof of Theorem 1.1] By Corollary 5.1, for any $\epsilon > 0$ there is a tree $T = (X, E_T, w_T)$ that defines an embedding of distortion at most $(1+\epsilon)\delta_{opt}(X)$ and that has maximum degree $O((1/\epsilon)^\lambda)$. Therefore, Theorem 4.1 gives a $(1+\varepsilon)$ approximation algorithm by setting deg to $O((1/\epsilon)^\lambda)$ and $\epsilon = \varepsilon/3$. The running time of the algorithm is

$$n^2\left(\frac{\Delta}{\varepsilon}\right)^{\lambda\log(1/\varepsilon)(1/\varepsilon)(O(\delta_{opt}^2/\varepsilon))^\lambda} = n^2\left(\frac{\Delta}{\varepsilon}\right)^{\log(1/\varepsilon)(1/\varepsilon)(O(\delta_{opt}^2/\varepsilon))^\lambda} = n^2\left(\frac{\Delta}{\varepsilon}\right)^{(O(\delta_{opt}/\varepsilon))^{2\lambda+1}}$$

where the last equality slightly weakens our run time bound in order to simplify the expression.

5 Bounded degree trees as host metrics

In this section, we show that a doubling metric space has a nearly optimal bounded degree tree spanner. In particular, this result implies that a doubling tree can be embedded into a bounded degree tree with same vertex set nearly isometrically. These results imply that considering bounded degree trees is sufficient when we study embedding into trees or computing geometric tree spanners. Specifically, they are used in the proofs of Theorem 1.1 and Corollary 1.2. Due to the space constraints the proof of the following is left to the full version.

Lemma 5.1. *Let (X, d_X) be a metric space of doubling dimension λ. Let $T = (X, E, w)$ be a tree with $w = d_X[E]$. Suppose the identity map from (X, d_X) to (X, d_T) has distortion δ. For any $0 < \varepsilon < 1$, there is a tree $T' = (X, E', w')$ with maximum degree $(O(\delta/\varepsilon))^\lambda$ such that $w' = d_X[E']$ and the identity map from (X, d_X) to $(X, d_{T'})$ has distortion at most $(1+\varepsilon)\delta$.*

Corollary 5.1. *Let $T = (V, E, w)$ be a tree of doubling dimension λ. For any $0 < \varepsilon < 1$, there is a tree $T' = (X, E', w')$ with maximum degree $(O(1/\varepsilon))^\lambda$ such that the identity map from (V, d_T) to $(V, d_{T'})$ has distortion at most $(1+\varepsilon)$.*

6 Tree spanners

In this section, we consider the closely related problem of finding tree spanners with minimum stretch. We start by considering a generalization of both the low-distortion trees and low-stretch tree spanners problems.

Let (X, d_X) be a finite metric space. Similar to the low-distortion embedding problem, we would like to embed X into a tree $T = (X, E_T, w_T)$. However, we have a set of constraints on possible edges and edge weights of T. Specifically, we have a ***constraint function*** $h : X \times X \to 2^{\mathbb{R}^+}$, which specifies the set of ***permitted weights*** for every pair of vertices $x, y \in X$ if we choose to include (x, y) in E_T. In particular, an edge (x, y) is banned if $h(x, y) = \emptyset$. A tree is a ***permitted tree*** if all its edge weights are permitted. Given (X, d_X) and h, the ***constrained embedding problem*** asks for the minimum distortion embedding of X into any permitted tree. Note that the minimum stretch tree spanner problem for a graph $G = (X, E_G, w_G)$ is equivalent to the constrained embedding problem for (X, d_G) and h, where $h(x, y) = \{w_G(x, y)\}$ if $(x, y) \in E_G$, and $h(x, y) = \emptyset$, otherwise. Moreover, the minimum distortion embedding problem is equivalent to the constrained embedding problem for (X, d_X) and h, where $h(x, y) = \mathbb{R}^+$ for all $x, y \in X$.

We modify the algorithm of Lemma 4.12 to solve the constrained embedding problem. Let $\boldsymbol{\delta_{opt}(X, \mathsf{deg}, h)}$ denote the minimum distortion of any embedding of (X, d_X) into a tree with vertex set X, maximum degree at most deg, and which is permitted with respect to h.

One issue is that our algorithm works with a discrete step size σ, while h might allow edge weights that are not integer multiples of σ. To resolve this issue, in a preprocessing step we change h to h', where $h'(x, y) = \{\lceil a/\sigma \rceil \cdot \sigma \mid a \in h(x, y)\}$. The proof of Lemma 4.1 implies that $\delta_{opt}(X, \mathsf{deg}, h') \leq (1 + \sigma)\delta_{opt}(X, \mathsf{deg}, h)$. Our algorithm then solves the problem for X, deg, and h' to find a tree $T' = (X, E_{T'}, w_{T'})$ that is permitted with respect to h'. To obtain a permitted tree with respect to h in a postprocessing step we modify $w_{T'}$ as follows. For each $(x, y) \in E_{T'}$, we set $w_T(x, y)$ to the largest permitted value (with respect to h) that is at most $w_{T'}(x, y)$. Let δ' and δ be distortions of identity maps from (X, d_X) to $(X, d_{T'})$ and from (X, d_X) to (X, d_T), respectively. Applying Lemma 4.1 in the reverse direction implies that $\delta \leq (1 + \sigma) \cdot \delta'$. Hence, the preprocessing and postprocessing introduce a factor of at most $(1 + \sigma)^2$ in the final distortion.

Now, let $\sigma > 0$, and let h' be the refined constraint function with step size σ. Also, let $\delta'_{opt} = \delta_{opt}(X, \mathsf{deg}, h')$. A modification of the argument of Lemma 4.6 shows the existence of a consistent set of views with parameters $(\mathsf{deg}, \sigma, c, \delta'_{opt})$ over a permitted tree T'. Given any consistent set of views over any tree Lemma 4.11 guarantees distortion at most $(1 - 14/(c - 1))^{-1} \cdot (1 + 20/(c - 1))\delta'_{opt}$ for the identity map to that tree.

We slightly modify the dynamic programming of Lemma 4.12 to compute a permitted tree with a consistent set of views over it. Specifically, whenever we check consistency between two views V_x and V_y over an edge (x, y), we make sure that $|A_x^d - A_y^d| \in h(x, y)$, that is $|A_x^d - A_y^d|$ is a permitted weight for (x, y). The rest of the algorithm remains intact. Together with the preprocessing and the postprcessing step, we obtain an algorithm that is guaranteed to return a tree with distortion at most

$$\left(1 - \frac{14}{c - 1}\right)^{-1} \left(1 + \frac{20}{c - 1}\right) \delta'_{opt} \leq \left(1 + \frac{608}{c - 1}\right) (1 + \sigma)^2 \delta_{opt}$$

So by setting $c = 7 \times 608/\epsilon + 1$ and $\sigma = \epsilon/7$, we are guaranteed that the distortion of our output is at most $(1 + \epsilon)\delta_{opt}$, and the following theorem follows.

Lemma 6.1. *Let X be an n-point metric space, with doubling dimension λ, and spread Δ. Also, let $h : X \times X \to 2^{\mathbb{R}^+}$ specify permitted edge weights. For any $\delta > 0$, $\epsilon > 0$, and $\mathsf{deg} > 0$ there is a*

$$n^2 \cdot \Delta^{\log_\delta(\mathsf{deg}/\epsilon)(O(\delta^2/\epsilon))^\lambda}$$

time algorithm to compute a $(1 + \epsilon)\delta$ distortion embedding of X into a permitted tree T (with respect to h) of maximum degree deg if $\delta \geq \delta_{opt}(X, \mathsf{deg}, h)$. If $\delta < \delta_{opt}(X, \mathsf{deg}, h)$, this algorithm either computes an embedding of distortion $(1 + \epsilon)\delta$ or (correctly) decides that $\delta < \delta_{opt}(X, \mathsf{deg}, h)$.

The general version of Theorem 4.1 follows, by the exact same proof.

Theorem 6.1. *Let X be an n-point metric space, with doubling dimension λ and spread Δ. Also, let $h : X \times X \to 2^{\mathbb{R}^+}$ specify permitted edge weights. For any $0 < \varepsilon < 1$ and $\mathsf{deg} > 1$, where $\delta_{opt} = \delta_{opt}(X, \mathsf{deg}, h)$, there is a*

$$n^2 \cdot \left(\frac{\Delta}{\varepsilon}\right)^{\log(\mathsf{deg}/\varepsilon) \cdot (1/\varepsilon) \cdot (O(\delta_{opt}^2/\varepsilon))^\lambda}$$

time algorithm to compute a $(1 + \varepsilon)\delta_{opt}$ distortion embedding of X into a permitted tree T of maximum degree deg.

Now, we are ready to prove our spanner results.

Proof. [Proof of Theorem 1.2] Let $h : X \times X \to 2^{\mathbb{R}^+}$ be defined as follows. For each $x, y \in X$, set $h(x, y) = \{w(x, y)\}$ if $(x, y) \in E$, and set $h(x, y) = \emptyset$ otherwise. For any $0 < \varepsilon < 1$, Theorem 6.1 finds a permitted embedding into a tree T of distortion at most $(1 + \varepsilon)\delta_{opt}((X, d_G), \mathsf{deg}, h)$. The constraint function ensures that T is a spanning tree of G. Also, as h allows all spanning trees of G and no other tree, we have $\delta_{opt}((X, d_G), \mathsf{deg}, h)$ is equal to the minimum stretch of all spanning trees. Note that the running time follows from that of Theorem 6.1 by slightly weakening (and simplifying) the exponent by writing $\log(\mathsf{deg}/\varepsilon) \cdot (1/\varepsilon) \cdot (O(\delta_{opt}^2/\varepsilon))^\lambda = \log(\mathsf{deg})(O(\delta_{opt}/\varepsilon))^{2\lambda+1}$.

Corollary 1.2 can be similarly proved from Theorem 6.1, by subsequently setting deg according to Lemma 5.1. See the full version for details.

References

[ABF$^+$99] R. Agarwala, V. Bafna, M. Farach, M. Paterson, and M. Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). *SIAM J. Comput.*, 28(3):1073–1085, 1999.

[BCIS05] M. Badoiu, J. Chuzhoy, P. Indyk, and A. Sidiropoulos. Low-distortion embeddings of general metrics into the line. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, STOC '05, pages 225–233. ACM, 2005.

[BDG$^+$05] M. Badoiu, K. Dhamdhere, A. Gupta, Y. Rabinovich, H. Räcke, R. Ravi, and A. Sidiropoulos. Approximation algorithms for low-distortion embeddings into low-dimensional spaces. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 119–128, 2005.

[BDH$^+$08] M. Bădoiu, E. Demaine, M. Hajiaghayi, A. Sidiropoulos, and M. Zadimoghaddam. Ordinal embedding: Approximation algorithms and dimensionality reduction. In *APPROX-RANDOM 2008*, pages 21–34, 2008.

[BIS07] M. Badoiu, P. Indyk, and A. Sidiropoulos. Approximation algorithms for embedding general metrics into trees. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 512–521, 2007.

[Bun71] P. Buneman. The Recovery of Trees from Measures of Dissimilarity. In *Mathematics in the Archeological and Historical Sciences*, pages 387–395. 1971.

[CC95] L. Cai and D. Corneil. Tree spanners. *SIAM J. Discret. Math.*, 8(3):359–387, August 1995.

[CDN$^+$10] V. Chepoi, F. Dragan, I. Newman, Y. Rabinovich, and Y. Vaxes. Constant approximation algorithms for embedding graph metrics into trees and outerplanar graphs. In *APPROX-RANDOM*, volume 6302 of *Lecture Notes in Computer Science*, pages 95–109, 2010.

[CHL07] O. Cheong, H. Haverkort, and M. Lee. Computing a minimum-dilation spanning tree is NP-hard. In *Proceedings of the Thirteenth Australasian Symposium on Theory of Computing - Volume 65*, CATS '07, pages 15–24, 2007.

[DH98] M. Demmer and M. Herlihy. The arrow distributed directory protocol. In *Proceedings of the 12th International Symposium on Distributed Computing*, DISC '98, pages 119–133, 1998.

[DK14] F. Dragan and E. Köhler. An approximation algorithm for the tree t-spanner problem on unweighted graphs via generalized chordal graphs. *Algorithmica*, 69(4):884–905, August 2014.

[EP08] Y. Emek and D. Peleg. Approximating minimum max-stretch spanning trees on unweighted graphs. volume 38, pages 1761–1781, December 2008.

[Epp00] D. Eppstein. Spanning trees and spanners. In Jörg-Rudiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, chapter 9, pages 425–461. 2000.

[EW05] D. Eppstein and K. Wortman. Minimum dilation stars. In *Proceedings of the Twenty-first Annual Symposium on Computational Geometry*, SCG '05, pages 321–326, 2005.

[FFL$^+$13] M. Fellows, F. Fomin, D. Lokshtanov, E. Losievskaja, F. Rosamond, and S. Saurabh. Distortion is fixed parameter tractable. *ACM Trans. Comput. Theory*, 5(4):16:1–16:20, November 2013.

[FGvL11] F. Fomin, P. Golovach, and E. van Leeuwen. Spanners of bounded degree graphs. *Inf. Process. Lett.*, 111(3):142–144, January 2011.

[GKL03] A. Gupta, R. Krauthgamer, and J. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *44th Symposium on Foundations of Computer Science (FOCS)*, pages 534–543, 2003.

[Gup01] A. Gupta. Steiner points in tree metrics don't (really) help. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 220–227, 2001.

[HIL98] J. Håstad, L. Ivansson, and J. Lagergren. Fitting points on the real line and its application to RH mapping. In *Proc. 6th Annual European Symposium on Algorithms*, ESA, pages 465–476, 1998.

[IM04] P. Indyk and J. Matoušek. Low-distortion embeddings of finite metric spaces. In *Handbook of Discrete and Computational Geometry*, pages 177–196, 2004.

[Ind01] P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, pages 10–, 2001.

[KRS04] C. Kenyon, Y. Rabani, and A. Sinclair. Low distortion maps between point sets. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC, pages 272–280, 2004.

[KW99] J. Kim and T. Warnow. Tutorial on phylogenetic tree estimation. In *Intelligent Systems for Molecular Biology*, 1999.

[LW08] C. Liebchen and G. Wünsch. The zoo of tree spanner problems. *Discrete Appl. Math.*, 156(5):569–587, March 2008.

[Mat13] J. Matoušek. *Lecture notes on metric embeddings*, 2013. Available at: `http://kam.mff.cuni.cz/~matousek/ba-a4.pdf`.

[NR15] A. Nayyeri and B. Raichel. Reality distortion: Exact and approximate algorithms for embedding into the line. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 729–747, 2015.

[NR17] A. Nayyeri and B. Raichel. A treehouse with custom windows: Minimum distortion embeddings into bounded treewidth graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 724–736, 2017.

[Pap15] I. Papoutsakis. Tree spanners of bounded degree graphs. *CoRR*, abs/1503.06822, 2015.

[PR01] D. Peleg and E. Reshef. Low complexity variants of the arrow distributed directory. *J. Comput. Syst. Sci.*, 63(3):474–485, November 2001.