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 with a given upper bound deg on the maximum degree, where the running time is parameterized with respect to deg, 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.

*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.

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

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

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. 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 parametrize 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 parametrized 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 polynomially large spread, we get can 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] The spread is the ratio of the largest to smallest distance in the metric, sometimes referred to as the aspect ratio.

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 non-trivial approximation algorithm, 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-spanner of a weighted graph, if it exists, coincides with the minimum spanning tree, and therefore can be computed efficiently. Nevertheless, computing t-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-spanners or 2-spanners, if they exist, while finding t-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-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 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 non-trivial 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.)

[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.

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 of a metric space. 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 2.2), 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 1.2. *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 be adapted to compute tree spanners of finite metric spaces. Here, the input is a finite metric space, and the tree spanner must be chosen from the set of all spanning trees of the complete graph representing the metric space. Note this theorem is different from Theorem 1.1 as the weight of an edge (x, x') in the tree spanner, for any $x, x' \in X$, must be equal to $d_X(x, x')$.

Theorem 1.3. *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 the above theorem only considers spanning trees from metric complete graphs. We can strengthen this result to an algorithm for computing tree spanners for general *weighted graphs*, however, the running time depends on the maximum allowable degree for the tree spanner.

Theorem 1.4. *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.*

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 bounded. Under the same set of conditions, Corollary 1.2 gives a constant factor approximation algorithm for embedding X into any tree. Again, under the same conditions, Theorem 1.3 gives a PTAS for computing the minimum stretch geometric tree spanner, and Theorem 1.4 gives a PTAS for computing the minimum stretch bounded degree tree spanner of a weighted graph.

[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 $\log(1/\varepsilon)(1/\varepsilon)(O(\delta_{opt}^2/\varepsilon))^{\lambda}$. 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.

Note to reviewer. Due to the technical nature of the problems and results, our definitions, theorems, and algorithms are fairly detailed and just their statements alone would not fit in the remaining space, let alone their proofs and descriptions of how they fit together. Thus we leave these details to the full version, and so here, after briefly defining terminology and some basic facts, with the remaining space we instead choose to give an informal overview of the problem. In particular, our overview is designed to be fairly informative, such that after reading it, the details in the full version can be seen as just an implementation of the main ideas outlined in the overview.

2 Preliminaries

Metrics. 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 problem). 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

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

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. 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 look for a such an embedding.

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 2.2 ([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.*

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

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 2.2 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 at set of additional constraints (on edge weights). The details of these constraints and their enforcement can be found in the full version of the paper.

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 on 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.

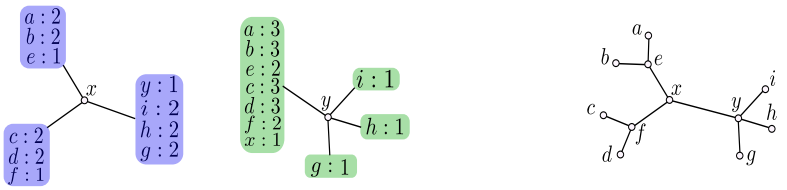

Figure 1: Left: views at x and y, right: a tree that is an extension of both views. To keep the figure readable, unweighted distance are used, though in general weights are allowed.

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.

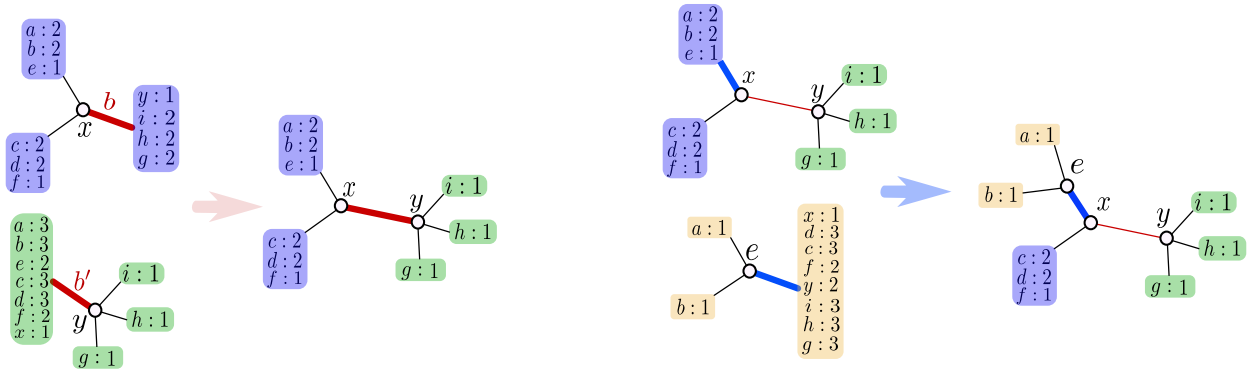

Figure 2: Left: stitching together a view at x and a view at y (to build the edge (x, y)), right: stitching a view at e to the view at x (to build the subtree of x, y, and e).

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 image of any other $y \in X$ under this embedding, and so requiring consistency of distances can be shown to imply that the view at y 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 $y \in X$, the x-y distance in the view has a value between $d_X(x, y)$ and $\delta d_X(x, y)$.

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 the full version of the paper 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 the 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 our 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.

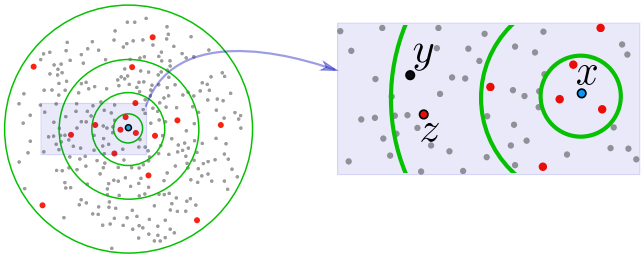

Figure 3: Left: hierarchical nets around x (net points are red, x is blue), right: z is used to estimate the distance to y in x's view.

To 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 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 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

[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.

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

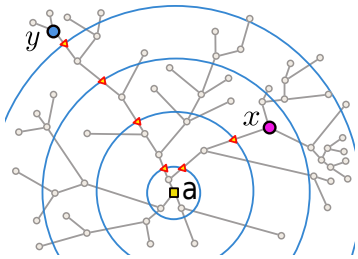

Figure 4: Anchor (yellow square), beacon rings (blue circles), and x-to-y mile marks (red triangles).

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. 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, our new running time is then bounded by,

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

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 $O(\text{poly}(n\Delta))$.

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.

References

[ABF+99] Richa Agarwala, Vineet Bafna, Martin Farach, Mike Paterson, and Mikkel Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). *SIAM J. Comput.*, 28(3):1073–1085, February 1999.

[BCIS05] Mihai Badoiu, Julia Chuzhoy, Piotr Indyk, and Anastasios 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] Mihai Badoiu, Kedar Dhamdhere, Anupam Gupta, Yuri Rabinovich, Harald Räcke, R. Ravi, and Anastasios Sidiropoulos. Approximation algorithms for low-distortion embeddings into low-dimensional spaces. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 119–128, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[BDH+08] Mihai Bădoiu, Erik D. Demaine, Mohammadtaghi Hajiaghayi, Anastasios Sidiropoulos, and Morteza Zadimoghaddam. Ordinal embedding: Approximation algorithms and dimensionality reduction. In *Proceedings of the 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008 on Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques*, APPROX '08 / RANDOM '08, pages 21–34, Berlin, Heidelberg, 2008. Springer-Verlag.

[BIS07] Mihai Badoiu, Piotr Indyk, and Anastasios 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, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[Bun71] Peter Buneman. The Recovery of Trees from Measures of Dissimilarity. In D. G. Kendall and P. Tautu, editors, *Mathematics the the Archeological and Historical Sciences*, pages 387–395. Edinburgh University Press, 1971.

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

[CDN+10] Victor Chepoi, Feodor F. Dragan, Ilan Newman, Yuri Rabinovich, and Yann Vaxes. Constant approximation algorithms for embedding graph metrics into trees and outerplanar graphs. In Maria J. Serna, Ronen Shaltiel, Klaus Jansen, and Jose D. P. Rolim, editors, *APPROX-RANDOM*, volume 6302 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2010.

[CHL07] Otfried Cheong, Herman Haverkort, and Mira 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, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[DH98] Michael J. Demmer and Maurice Herlihy. The arrow distributed directory protocol. In *Proceedings of the 12th International Symposium on Distributed Computing*, DISC '98, pages 119–133, London, UK, UK, 1998. Springer-Verlag.

[DK14] Feodor F. Dragan and Ekkehard 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] Yuval Emek and David Peleg. Approximating minimum max-stretch spanning trees on unweighted graphs. volume 38, pages 1761–1781, Philadelphia, PA, USA, December 2008. Society for Industrial and Applied Mathematics.

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

[EW05] David Eppstein and Kevin A. Wortman. Minimum dilation stars. In *Proceedings of the Twenty-first Annual Symposium on Computational Geometry*, SCG '05, pages 321–326, New York, NY, USA, 2005. ACM.

[FFL+13] Michael Fellows, Fedor V. Fomin, Daniel Lokshtanov, Elena Losievskaja, Frances Rosamond, and Saket Saurabh. Distortion is fixed parameter tractable. *ACM Trans. Comput. Theory*, 5(4):16:1–16:20, November 2013.

[FGvL11] Fedor V. Fomin, Petr A. Golovach, and Erik Jan van Leeuwen. Spanners of bounded degree graphs. *Inf. Process. Lett.*, 111(3):142–144, January 2011.

[GKL03] Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 534–543, 2003.

[Gup01] Anupam 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, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[HIL98] Johan Håstad, Lars Ivansson, and Jens Lagergren. Fitting points on the real line and its application to RH mapping. In *Proceedings of the 6th Annual European Symposium on Algorithms*, ESA '98, pages 465–476, London, UK, UK, 1998. Springer-Verlag.

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

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

[KRS04] Claire Kenyon, Yuval Rabani, and Alistair Sinclair. Low distortion maps between point sets. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 272–280, New York, NY, USA, 2004. ACM.

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

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

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

[NR15] Amir Nayyeri and Benjamin Raichel. Reality distortion: Exact and approximate algorithms for embedding into the line. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 729–747, 2015.

[NR17] Amir Nayyeri and Benjamin 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 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 724–736, 2017.

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

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