Online, Dynamic, and Distributed Embeddings of Approximate Ultrametrics

Michael Dinitz[*]

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213 USA
mdinitz@cs.cmu.edu

Abstract. The theoretical computer science community has traditionally used embeddings of finite metrics as a tool in designing approximation algorithms. Recently, however, there has been considerable interest in using metric embeddings in the context of networks to allow network nodes to have more knowledge of the pairwise distances between other nodes in the network. There has also been evidence that natural network metrics like latency and bandwidth have some nice structure, and in particular come close to satisfying an ϵ-three point condition or an ϵ-four point condition. This empirical observation has motivated the study of these special metrics, including strong results about embeddings into trees and ultrametrics. Unfortunately all of the current embeddings require complete knowledge about the network up front, and so are less useful in real networks which change frequently. We give the first metric embeddings which have both low distortion and require only small changes in the structure of the embedding when the network changes. In particular, we give an embedding of semimetrics satisfying an ϵ-three point condition into ultrametrics with distortion $(1+\epsilon)^{\log n+4}$ and the property that any new node requires only $O(n^{1/3})$ amortized edge swaps, where we use the number of edge swaps as a measure of "structural change". This notion of structural change naturally leads to small update messages in a distributed implementation in which every node has a copy of the embedding. The natural offline embedding has only $(1+\epsilon)^{\log n}$ distortion but can require $\Omega(n)$ amortized edge swaps per node addition. This online embedding also leads to a natural dynamic algorithm that can handle node removals as well as insertions.

1 Introduction

Many network applications, including content distribution networks and peer-to-peer overlays, can achieve better performance if they have some knowledge of network latencies, bandwidths, hop distances, or some other notion of distance between nodes in the network. Obviously the application could estimate these

[*] This work was supported in part by an NSF Graduate Research Fellowship and an ARCS Scholarship, and was partially done while the author was an intern at Microsoft Research-Silicon Valley

distances by an on-demand measurement, but they get even more benefit from an instantaneous estimate that does not require recurring measurements. Many solutions have been proposed for this problem, most of which involve measuring either a random or a carefully chosen subset of pairwise distances and then embedding into a low-dimension coordinate space, e.g. a Euclidean space [6, 7, 12, 9, 10]. Because of this methodology, the problem of giving good approximations of network latencies is sometimes referred to as the *network coordinate problem*.

We take our inspiration from the work of Abraham et al. [1] who tried to use embeddings into *tree metrics* instead of into Euclidean space. Through experimental measurements they discovered that many networks have latencies that "locally" look like trees, and in particular satisfy an ϵ-four point condition that they invented. They gave an embedding into a single tree which has small distortion for small ϵ and allows instantaneous distance measurements like in the coordinate case, but also due to its tree structure has a natural notion of hierarchy and clustering that corresponds to an intuitive model of the Internet. However, their embedding was mainly of theoretical interest, since in practice it required having global knowledge of the complete network and complete recomputation whenever the network changed. Since real networks change regularly and do not normally allow a global viewpoint, the applicability of their tree embedding was unclear.

In this paper we take a first step towards a practical tree embedding for network distances. We define an ϵ-three point condition (as opposed to the four point condition of [1]) and show that there are good embeddings into ultrametrics when ϵ is small. Since ultrametrics are a special case of tree metrics this is also a tree embedding, but they have the extra property that they can be represented by *spanning* trees, so no Steiner nodes are required. They also satisfy a "bottleneck" condition which allows ultrametrics to represent bandwidths (actually 1 over the bandwidth) particularly well. We do not want to assume a static network like in previous work, so we give an *online* embedding which works no matter what order nodes in the network arrive (but assumes that nodes never leave) and a *dynamic* embedding which extends the online embedding to handle the case of nodes leaving.

In this paper we allow our online embedding to change the already-embedded points when a new one arrives, unlike the traditional model of online algorithms which does not allow change. However, an important part of this work is *minimizing* that change, and in fact trading it off with the distortion. We try to minimize the "structural change" of the embedding, where we measure the structural change by the number of edges in the disjoint union of the tree before a node insertion and the tree after a node insertion. We show that there is a tradeoff between distortion and structural change: the offline algorithm that minimizes distortion has large structural change, while by slightly relaxing the distortion requirement we can decrease the structural change by a polynomial amount. This immediately results in a bound on the communication complexity of a distributed algorithm that sends update messages across an underlying routing fabric.

Aside from the original practical motivation, we believe that our embedding is also of theoretical interest. Designing online embeddings is a very natural problem, but has proven surprisingly difficult to solve. In fact, there is basically no previous work on embeddings in any model of computation other than the standard offline full information model, even while online, dynamic, and streaming algorithms are becoming more important and prevalent. While in many cases it is easy to see that it is basically impossible to give an online embedding that doesn't change the embedding of previous points at all, we hope that this paper will begin the study of online embeddings that minimize structural change, or at least tradeoff structural change for extra distortion.

1.1 Definitions and Results

A *semimetric* is a pair (V, d) where $d : V \times V \to \mathbb{R}_{\geq 0}$ is a distance function with three properties that hold for all $x, y \in V$. First, $d(x, y) \geq 0$. Second, $d(x, y) = 0$ if and only if $x = y$. Lastly, $d(x, y) = d(y, x)$. Note that we are not assuming a metric space, so we do not assume the triangle inequality. Instead, throughout this paper we assume that the underlying semimetric is approximately an ultrametric, in the following sense.

Definition 1 (ϵ-three point condition). *A semimetric (V, d) satisfies the ϵ-*three point condition *if $d(x, y) \leq (1 + \epsilon) \max\{d(x, z), d(y, z)\}$ for all $x, y, z \in V$. We let $\mathsf{3PC}(\epsilon)$ denote the class of semimetrics satisfying the ϵ-three point condition.*

Ultrametrics are exactly the metrics that satisfy the 0-three point condition, i.e. $\mathsf{ULT} = \mathsf{3PC}(0)$ where ULT is the set of all ultrametrics. So for any given semimetric, the smallest ϵ for which it satisfies the ϵ-three point condition is a measure of how "close" it is to being an ultrametric. Clearly every metric satisfies the 1-three point condition, since by the triangle inequality $d(x, y) \leq d(x, z) + d(y, z) \leq 2 \max\{d(x, z), d(y, z)\}$. Thus when the semimetric actually satisfies the triangle inequality, we know that it is in $\mathsf{3PC}(1)$.

This definition is in contrast to the ϵ-four point condition defined in [1], which is a condition on the lengths of matchings on four points rather than the lengths of edges on three points. In particular, a metric (V, d) satisfies the ϵ-four point condition if $d(w, z) + d(x, y) \leq d(w, y) + d(x, z) + 2\epsilon \min\{d(w, x), d(y, z)\}$ for all $x, y, z, w \in V$ such that $d(w, x) + d(y, z) \leq d(w, y) + d(x, z) \leq d(w, z) + d(x, y)$. This definition also has the property that all metrics satisfy the 1-four point condition, but instead of being related to ultrametrics it is related to trees, in that tree metrics are exactly the metrics satisfying the 0-four point condition.

While ultrametrics are just semimetrics in $\mathsf{3PC}(0)$, we will represent them as trees over the point set. Given a tree $T = (V, E)$ with weights $w : E \to \mathbb{R}_{\geq 0}$, for $x, y \in V$ let $P(x, y)$ denote the set of edges on the unique path in T between x and y. Define $d_T : V \times V \to \mathbb{R}_{\geq 0}$ by $d_T(x, y) = \max_{e \in P(x,y)} w(e)$. It is easy to see that (V, d_T) is an ultrametric, and that every ultrametric can be represented in this way. We say that (V, d_T) is the ultrametric *induced* by T.

Since these are the only embeddings we will use, we define the contraction, expansion, and distortion of an embedding in the obvious ways. Given a semimetric (V, d) and a tree T on V, the *expansion* of the embedding is $\max_{u,v \in \binom{V}{2}} \frac{d_T(u,v)}{d(u,v)}$, the *contraction* of the embedding is $\max_{u,v \in \binom{V}{2}} \frac{d(u,v)}{d_T(u,v)}$, and the *distortion* is the product of the expansion and the contraction.

In this paper we want to solve embedding problems *online*, when vertices arrive one at a time and we want a good embedding at every step. An online ultrametric embedding is simply an algorithm that maintains a tree (and thus an induced ultrametric) on the vertices that have arrived, and outputs a new spanning tree when a new node arrives. There are two measures of the quality of an online embedding: its distortion and its update cost. An online embedding has distortion at most c if every tree that it outputs has distortion at most c relative to the distances between the nodes that have already arrived.

The *update cost* of the algorithm is the average number of edges in the tree that have to be changed when a node arrives. Let $\mathcal{A}(\epsilon)$ be an online ultrametric embedding, and suppose that it is run on a semimetric $(V, d) \in \mathsf{3PC}(\epsilon)$ where the order in which the nodes arrive is given by π. Let $T_i = (V_i, E_i)$ be the tree given by the $\mathcal{A}$ after the ith node arrives. Then $cost(\mathcal{A}(\epsilon), V, d, \pi) = \sum_{i=1}^{|V|} |E_i \setminus E_{i-1}|/|V|$. We define the overall update cost of $\mathcal{A}(\epsilon)$ to be its worst case cost over semimetrics satisfying the ϵ-three point condition and ordering of the nodes of the semimetric, i.e. $\sup_{(V,d) \in \mathsf{3PC}(\epsilon), \pi:[n] \to V} cost(\mathcal{A}, v, d, \pi)$.

Now we can state our main result in the online framework. All logs are base 2 unless otherwise noted.

Theorem 1. *For any $\epsilon \geq 0$ there is an online ultrametric embedding algorithm for $\mathsf{3PC}(\epsilon)$ which has distortion at most $(1+\epsilon)^{\lceil \log n \rceil + 4}$ and update cost at most $n^{1/3}$, where n is the number of points in the semimetric.*

The dynamic setting is similar to the online setting, except that nodes can also leave after they arrive. The cost is defined similarly, as the average number of edges changed by an operation, where now an operation can be either an arrival or a departure. We modify our online algorithm to handle departures, giving a similar theorem.

Theorem 2. *For any $\epsilon \geq 0$ there is a dynamic ultrametric embedding for $\mathsf{3PC}(\epsilon)$ which has distortion at most $(1+\epsilon)^{\lceil \log n \rceil + 5}$ (where n is the number of nodes currently in the system) and update cost at most $O(n_{max}^{1/3})$ (where n_{max} is the maximum number of nodes that are ever in the system at one time).*

Note that this notion of update cost is essentially the communication complexity of the natural distributed implementation. In particular, suppose that every node in a network has a copy of the tree (and thus the embedding). When a new node arrives, it obtains a copy of the tree from some neighbor, and then runs an algorithm to determine what the new embedding should be. The update cost is just the expected size of the update message which must be sent across the network to inform all nodes of the new embedding, so the communication complexity of the algorithm is at most $O(tn_{max})$ times the update cost where t is the number of operations (since for each operation we have to send an update to every node in the system). We describe this application in more detail in the full version, but the main result is the following upper bound.

Theorem 3. *There is a distributed dynamic ultrametric embedding algorithm for* $\mathsf{3PC}(\epsilon)$ *that allows every node to estimate the distance between any two nodes up to a distortion of* $(1+\epsilon)^{\lceil \log n \rceil + 5}$ *while only using space* $O(n)$ *at each node and communication complexity* $O(tn_{max}^{4/3})$ *in the dynamic arrival model.*

1.2 Related Work

Our work is heavily influenced by the work of Abraham et al. [1], who gave an embedding of metrics satisfying the related ϵ-four point condition into tree metrics. While our techniques are not the same (since we focus on ultrametrics), the motivation (network latency prediction) is, and part of the original motivation for this work was an attempt to make a distributed version of the algorithm of [1].

This work also fits into the "local vs. global" framework developed by [3] and continued in [5]. These papers consider embeddings from one space into a class of spaces where it is assumed that all subsets of the original metric of size at most some k embed well into the target class. It is easy to see that a semimetric satisfies the ϵ-three point condition if and only if every subset of size 3 embeds into an ultrametric with distortion at most $(1+\epsilon)$. They essentially solve the offline problem for ultrametrics by showing that if every subset of size k embeds into an ultrametric with distortion at most $1+\epsilon$ then the entire space embeds into an ultrametric with distortion at most $(1+\epsilon)^{O(\log_k n)}$. The case of $k = 3$ is special, though, since it is the smallest value for which it is true that if every subset of size k embeds isometrically into an ultrametric then the entire space can be embedded isometrically into an ultrametric. Thus it is the most important case, and we tailor our algorithms to it.

We also note that the definitions of the ϵ-three point condition given by Abraham et al. [1] and our definition of the ϵ-four point condition are closely related to the notion of δ-hyperbolicity defined by Gromov in his seminal paper on hyperbolic groups [8]. In particular, his notion of hyperbolicity is an additive version of our notions. While we do not use any of the results or techniques of [8] directly, the methods and results are somewhat similar.

Finally, while there has been extensive work in the theoretical computer science community on metric embeddings, none of it has been concerned with models of computation other than the standard offline case. This seems to be a glaring omission, and this work is the first to provide positive results. We hope that it will be the first in a line of work examining metric embeddings in other models, especially online, dynamic, and streaming computation.

2 Online Embedding

We begin with the following lemma, which is an easy corollary of [3, Theorem 4.2]. We will use it throughout to bound the contraction of our embeddings.

Lemma 1. *Let (V, d) be a semimetric in $\mathsf{3PC}(\epsilon)$. Then for any spanning tree T on V and nodes $u, v \in V$, $d(u, v) \leq (1 + \epsilon)^{\lceil \log n \rceil} d_T(u, v)$*

This immediately implies that the minimum spanning tree is a good embedding:

Theorem 4. *Let T be a minimum spanning tree of the semimetric $(V, d) \in \mathsf{3PC}(\epsilon)$. Then $d_T(u, v) \leq d(u, v) \leq (1 + \epsilon)^{\lceil \log n \rceil} d_T(u, v)$ for all $u, v \in V$*

Proof. We know from Lemma 1 that $d(u, v) \leq (1 + \epsilon)^{\lceil \log n \rceil} d_T(u, v)$, so it just remains to prove that $d_T(u, v) \leq d(u, v)$. Let e be the maximum length edge on the path between u and v in T, and let ℓ be its length. If $\ell > d(u, v)$, then the spanning tree obtained by adding the edge $\{u, v\}$ and removing e is smaller than T, which is a contradiction since T is a minimum spanning tree. Thus $\ell \leq d(u, v)$, and so by definition $d_T(u, v) \leq d(u, v)$. $\square$

It is actually easy to see that this is basically the same embedding used by [3], and furthermore that by replacing the metric distance with the Gromov product (as defined in [8]) and min/max with max/min we recover the embedding of hyperbolic spaces given in [8].

While the MST is a good embedding from the standpoint of distortion, computing and maintaining it requires possibly changing many edges on every update. For example, consider the metric on $\{x_1, \ldots, x_n\}$ that has $d(x_i, x_j) = 1 - i\delta$ for $i > j$, where δ is extremely small. If the nodes show up in the network in the order $x_1, x_2, \ldots, x_n$, then after x_i enters the system the MST is just the star rooted at x_i. So to maintain the MST we have to get rid of every edge from the old star and add every edge in the new one, thus requiring $\Omega(n)$ average edge changes per update.

One thing to note about this example is that it is not necessary to maintain the MST. Instead of relocating the center of the star to the new node every time, if that new node just adds one edge to the first node then the induced ultrametric only expands distances by a small amount. This suggests that maybe it is enough to just add the smallest edge incident on v when v arrives. Unfortunately this is not sufficient, as it is easy to construct semimetrics in $\mathsf{3PC}(\epsilon)$ for which this algorithm has distortion $(1+\epsilon)^{\Omega(n)}$. So maintaining the MST has good distortion but large update cost, while just adding the smallest edge has small update cost but large distortion. Can these be balanced out, giving small distortion and small update cost? In this section we show that, to a certain extent, they can.

We define the *k-threshold algorithm* as follows. We maintain a spanning tree T of the vertices currently in the system. When a node u enters the system, examine the edges from u to every other node in increasing order of distance. When examining $\{u, v\}$ we test whether $d_T(u, v) > (1 + \epsilon)^k d(u, v)$, and if so we add $\{u, v\}$ to T and remove the largest edge on the cycle that it induces. If not, then do not add $\{u, v\}$ to T. A formal definition of this algorithm is given in Algorithm 1.

```
Input: Weighted tree $T = (V, E, w : E \to \mathbb{R}_{\geq 0})$, new vertex $u$, distances
        $d : u \times V \to \mathbb{R}_{\geq 0}$
Output: Weighted tree $T' = (V \cup \{u\}, E', w')$
Sort $V$ so that $d(u, v_1) \leq d(u, v_2) \leq \ldots \leq d(u, v_n)$;
Let $V' = V \cup \{u\}$, let $E' = E \cup \{\{u, v_1\}\}$, and set $w(\{u, v_1\}) = d(u, v_1)$;
for $i = 2$ to $n$ do
    Let $e$ be the edge of maximum weight on the path between $u$ and $v_i$ in
    $T' = (V', E')$;
    if $w(e) > (1 + \epsilon)^k d(u, v_i)$ then
        $E' \leftarrow E' \setminus \{e\} \cup \{\{u, v_i\}\}$;
        $w(\{u, v_i\}) = d(u, v_i)$;
    end
end
return $T' = (V', E', w)$
```

Algorithm 1: k-threshold algorithm

This algorithm has the useful property that the embedded distance between two nodes never increases. More formally, consider a single step of the algorithm, in which it considers the edge $\{u, v\}$ and either adds it or does not. Let T be the tree before and T' the tree after this decision (note that both T and T' share the same node set V).

Lemma 2. *$d_{T'}(x, y) \leq d_T(x, y)$ for all nodes $x, y \in V$*

Proof. If the algorithm does not add $\{u, v\}$ then $T = T'$, so $d_T(x, y) = d_{T'}(x, y)$. If it adds $\{u, v\}$ then it removes the largest edge on the path in T between u and v, say $\{w, z\}$ (without loss of generality we assume that the path from u to v in T goes through w before z). Clearly $d(w, z) > (1 + \epsilon)^k d(u, v)$ or else the algorithm would not have added $\{u, v\}$. If the path in T between x and y does not go through $\{w, z\}$ then it is the same as the path between x and y in T', so $d_T(x, y) = d_{T'}(x, y)$. If the path does go through $\{w, z\}$ then the path from u to w is unchanged, every edge on the path in T' from w to z has length at most $d(w, z)$, and the path from z to v is unchanged. Thus $d_{T'}(x, y) \leq d_T(x, y)$. □

Given this lemma, the following theorem is immediate.

Theorem 5. *After adding n nodes, the k-threshold algorithm results in an embedding into an ultrametric with distortion at most $(1 + \epsilon)^{k + \lceil \log n \rceil}$*

Proof. We know from Lemma 1 that the contraction is at most $(1 + \epsilon)^{\lceil \log n \rceil}$, so we just need to show that the expansion is at most $(1 + \epsilon)^k$. Let u and v be any two vertices, and without loss of generality let u enter the system after v.

Then when u enters the system it chooses to either add the edge to v or not, but in either case immediately after that decision the expansion of $\{u, v\}$ is at most $(1+\epsilon)^k$. Lemma 2 implies that this distance never increases, and thus the expansion is always at most $(1+\epsilon)^k$. □

The hope is that by paying the extra $(1+\epsilon)^k$ in distortion we get to decrease the number of edges added. Our main theorems are that this works for small k, specifically for k equal to 2 and 4.

Theorem 6. *The 2-threshold algorithm has update cost at most $n^{1/2}$*

Theorem 7. *The 4-threshold algorithm has update cost at most $n^{1/3}$*

We believe that this pattern continues for larger k, so that the k-threshold algorithm would have update cost $n^{2/(k+2)}$. Unfortunately our techniques do not seem strong enough to show this, since they depend on carefully considering the local structure of the embedding around any newly inserted node, where the definition of "local" increases with k. This type of analysis becomes infeasible for larger k, so a more general proof technique is required. However, the cases of small k are actually the most interesting, in that they provide the most "bang-for-the-buck": if our conjecture is true, then when k gets larger the same increase in distortion provides a much smaller decrease in update cost.

The following lemma about the operation of the k threshold algorithm will be very useful, as it gives us a necessary condition for an edge to be added that depends not just on the current tree but on all of the edges that have been inserted by the algorithm.

Lemma 3. *Suppose that u arrives at time t, and let v be any other node that is already in the system. Suppose there exists a sequence $u = u_0, u_1, \dots, u_k = v$ of nodes with the following three properties:*

1. *The edge $\{u, u_1\}$ is considered by the k-threshold algorithm before $\{u, v\}$*
2. $d(u_i, u_{i+1}) \leq (1+\epsilon)^k d(u, v)$
3. *If $\{u_i, u_{i+1}\}$ has never been inserted by the algorithm, then* $d(u_i, u_{i+1}) \leq d(u, v)$

Then the k-threshold algorithm will not add the edge $\{u, v\}$

Proof. For any two nodes x and y, let $x \sim y$ denote the path between them in the algorithm's tree just before it considers $\{u, v\}$. We know from the first property that $d(u, u_1) \leq d(u, v)$, and so when $\{u, v\}$ is considered the maximum edge on $u \sim u_1$ is at most $(1+\epsilon)^k d(u, u_1) \leq (1+\epsilon)^k d(u, v)$. Fix i, and suppose that $\{u_i, u_{i+1}\}$ was already inserted by the algorithm. Then when it was inserted the maximum edge on the path between u_i and u_{i+1} was just that edge, which had length $d(u_i, u_{i+1})$. So by Lemma 2, the maximum edge of $u_i \sim u_{i+1}$ is at most $d(u_i, u_{i+1})$, which by the second property is at most $(1+\epsilon)^k d(u, v)$. On the other hand, suppose that $\{u_i, u_{i+1}\}$ was not inserted by the algorithm. Then when it was considered the path between u_i and u_{i+1} had maximum edge at most

$(1+\epsilon)^k d(u_i, u_{i+1})$, and so by Lemma 2 and the third property we know that the maximum edge of $u_i \sim u_{i+1}$ is at most $(1+\epsilon)^k d(u_i, u_{i+1}) \leq (1+\epsilon)^k d(u, v)$.

So for all $i \in \{0, \dots, k-1\}$, the path $u_i \sim u_{i+1}$ has maximum edge at most $(1+\epsilon)^k d(u, v)$. Since $u \sim v$ is a subset of $u \sim u_1 \sim \dots \sim u_k$, this implies that $u \sim v$ has maximum edge at most $(1+\epsilon)^k d(u, v)$, so the k-threshold algorithm will not add the edge $\{u, v\}$. □

2.1 2-threshold

We now prove Theorem 6. Let G_t be the "all-graph" consisting of every edge ever inserted into the tree by the 2-threshold algorithm before the tth node arrives. In order to prove Theorem 6 we will use the following lemma, which states that nodes which are close to each other in G_t have smaller expansion than one would expect.

Lemma 4. *Let u and v be any two nodes. If there is some t_0 such that there is a path of length 2 between u and v in G_{t_0}, then for all $t \geq t_0$ the expansion of $\{u, v\}$ in the induced ultrametric is at most $(1+\epsilon)$*

Proof. Let $u - w - v$ be a path of length two in G_t. We want to show that $\max\{d(u, w), d(w, v)\} \leq (1+\epsilon) d(u, v)$, since this along with Lemma 2 will imply that the the largest edge on the path between u and v in G_t has length at most $(1+\epsilon) d(u, v)$. If $d(u, v) \geq d(u, w)$, then $d(w, v) \leq (1+\epsilon) \max\{d(u, w), d(u, v)\} = (1+\epsilon) d(u, v)$, which would prove the lemma. Similarly, if $d(u, v) \geq d(w, v)$ then $d(u, w) \leq (1+\epsilon) \max\{d(w, v), d(u, v)\} = (1+\epsilon) d(u, v)$, again proving the lemma. So without loss of generality we assume that $d(u, w) > d(u, v)$ and $d(w, v) > d(u, v)$. Together with the ϵ-three point condition this implies that $d(u, w) \leq (1+\epsilon) d(w, v)$ and $d(w, v) \leq (1+\epsilon) d(u, w)$. We will now derive a contradiction from these assumptions, which proves the lemma since it implies that either $d(u, v) \geq d(u, w)$ or $d(u, v) \geq d(w, v)$.

Suppose that u is the last node of the three to enter the system. Then the sequence u, v, w satisfies the properties of Lemma 3, so the algorithm would not add $\{u, w\}$, giving a contradiction since we assumed that $\{u, w\}$ was in G_t. If v is the last of the three to arrive, then v, u, w satisfies the properties of Lemma 3, so $\{v, w\}$ would not have been added, giving a contradiction. If w is the last node of the three to be added, then we can assume by symmetry that $\{u, w\}$ was considered by the algorithm before $\{w, v\}$, in which case w, u, v satisfies the properties of Lemma 3, so $\{w, v\}$ would not have been added. Thus no matter which of the three arrives last we get a contradiction, proving the lemma. □

We will also need to relate the girth of a graph to its sparsity, for which we will use the following simple result that can be easily derived from [4, Theorem 3.7] and was stated in [2] and [11]. Recall that the girth of a graph is the length of the smallest cycle.

Lemma 5. *If a graph G has girth at least $2k+1$, then the number of edges in G is at most $n^{1+\frac{1}{k}}$*

Now we can use a simple girth argument to prove Theorem 6.

Proof of Theorem 6. Suppose that G_{n+1} (which is the final graph of all edges ever inserted by the algorithm) has a cycle of length $h \leq 4$ consisting of the nodes $x_1, x_2, \ldots, x_4$, where x_1 is the last of the h nodes on the cycle to have entered the system. Without loss of generality, assume that $\{x_1, x_2\}$ is considered by the algorithm before $\{x_1, x_h\}$. Let e be the largest edge on the cycle other than $\{x_1, x_h\}$ and $\{x_1, x_2\}$. In the case of $h = 4$, Lemma 4 implies that $e \leq (1+\epsilon)^{h-3} d(x_2, x_h)$, and if $h = 3$ then this is true by definition since then $e = \{x_2, x_h\}$. But now we know from the ϵ-three point condition that this is at most $(1+\epsilon)^{h-2} \max\{d(x_1, x_2), d(x_1, x_h)\} = (1+\epsilon)^{h-2} d(x_1, x_h) \leq (1+\epsilon)^2 d(x_1, x_h)$. Thus every edge on the cycle is at most $(1+\epsilon)^2 d(x_1, x_h)$, so applying Lemma 3 to $x_1, x_2, \ldots, x_h$ implies that $\{x_1, x_h\}$ would not be added by the algorithm, and therefore would not be in G_{n+1}. Thus G_{n+1} has girth at least 5, so by Lemma 5 we have that the number of edges added by the algorithm is at most $n^{3/2}$, and thus the 2-threshold algorithm has update cost at most $n^{1/2}$. □

2.2 4-threshold

In order to prove Theorem 7 we will prove a generalization of Lemma 4 for the 4-threshold algorithm. The proof follows the same outline as the proof of Lemma 4: first we show a series of sufficient conditions, and the assuming them all to be false we derive a contradiction with the operation of the algorithm. In one case this technique is insufficient, and instead we have to assume that not only are the sufficient conditions false, but so is the lemma, and then this is sufficient to derive a contradiction.

Lemma 6. *Let u and v be any two nodes, and let $h \leq 4$. Then if there is some t_0 such that there is a path of length h between u and v in G_{t_0}, then for all $t \geq t_0$ the expansion of $\{u, v\}$ in the induced ultrametric is at most $(1+\epsilon)^{h-1}$*

Proof. The case of $h = 1$ is obvious, since a path of length 1 is just an edge and Lemma 2 implies that the embedded distance from then on does not increase. The case of $h = 2$ is basically the same as Lemma 4, just with $k = 4$ instead of $k = 2$. Since the proof is basically the same, we defer it to the full version.

For the case of $h = 3$, consider a path $u - x - y - v$ in G_t. As before, we want to show that $\max\{d(u, x), d(x, y), d(y, v)\} \leq (1+\epsilon)^2 d(u, v)$, since along with Lemma 2 this will prove the lemma. Due to symmetry, we can assume without loss of generality that $d(u, x) \geq d(v, y)$. So there are two cases, corresponding to whether the maximum edge on the path is $\{u, x\}$ or whether it is $\{x, y\}$. In either case the theorem is trivially true if the maximum edge is at most $d(u, v)$, so without loss of generality we will assume that it is greater that $d(u, v)$.

Suppose that $\{u, x\}$ is the maximum edge on the path. Then we know from the $h = 2$ case that $d(u, x) \leq (1+\epsilon) d(u, y) \leq (1+\epsilon)^2 \max\{d(u, v), d(v, y)\}$. If $d(u, v) \geq d(v, y)$ then this proves the lemma, so without loss of generality we can assume that $d(u, v) < d(v, y)$, which implies that $d(u, x) \leq (1+\epsilon)^2 d(v, y)$. Now by using Lemma 3 we can derive a contradiction by finding an edge in G_t that the algorithm would not have added. If u is the last node of u, x, y, v to enter the system, then the sequence u, v, y, x satisfies the properties of Lemma 3, so the edge $\{u, x\}$ would not have been added. If x is the last of the four to enter then we can apply Lemma 3 to the sequence x, y, v, u, so again the edge $\{u, x\}$ would not have been added. If v is the last of the four to enter then we can apply Lemma 3 to the sequence v, u, x, y, so the edge $\{v, y\}$ would not have been added. Finally, if y is the last of the four to arrive then if $\{x, y\}$ is considered before $\{y, v\}$ then applying Lemma 3 to the sequence y, x, u, v contradicts the addition of the edge $\{y, v\}$, and if $\{y, v\}$ is considered before $\{x, y\}$ then applying Lemma 3 to the sequence y, v, u, x contradicts the addition of the edge $\{x, y\}$. Thus no matter which of the four nodes enters last we can find an edge in G_t that the algorithm would not have added.

The second case is that $\{x, y\}$ is the largest of the three edges. In this case we want to prove that $d(x, y) \leq (1+\epsilon)^2 d(u, v)$. We know from the $h = 2$ case that $d(x, y) \leq (1+\epsilon)d(u, y) \leq (1+\epsilon)^2 \max\{d(u, v), d(y, v)\}$, which implies that we are finished if $d(u, v) \geq d(y, v)$, and thus without loss of generality $d(y, v) > d(u, v)$ and $d(x, y) \leq (1+\epsilon)^2 d(y, v)$. Similarly, we know that $d(x, y) \leq (1+\epsilon)d(v, x) \leq (1+\epsilon)^2 \max\{d(u, v), d(u, x)\}$, so without loss of generality $d(u, x) > d(u, v)$ and $d(x, y) \leq (1+\epsilon)^2 d(u, x)$. So all three of the edges are larger than $d(u, v)$, and they are all within a $(1+\epsilon)^2$ factor of each other. Thus no matter which of the four nodes enters last there is a sequence to which we can apply Lemma 3 in order to find an edge in G_t which the algorithm wouldn't add. If the last node is u then the sequence is u, v, y, x and the edge is $\{u, x\}$, if the last node is x then the sequence is x, u, v, y and the edge is $\{x, y\}$, if the last node is y then the sequence is y, v, u, x and the edge is $\{x, y\}$, and if the last node is v then the sequence is v, u, x, y and the edge is $\{v, y\}$. Thus we have a contradiction in every case, which proves the lemma for $h = 3$.

For the $h = 4$ case, consider a path $u - x - z - y - v$ of length 4 in G_t. We want to show that $\max\{d(u, x), d(x, z), d(z, y), d(y, w)\} \leq (1+\epsilon)^3 d(u, v)$. By symmetry there are only two cases: either $d(u, x)$ is the maximum or $d(x, z)$ is the maximum. Let e be the length of this maximum edge. The lemma is clearly true if $e \leq d(u, v)$, so in both cases we can assume that the maximum edge is at least $d(u, v)$. We also know from the $h = 3$ case that $e \leq (1+\epsilon)^2 d(u, y) \leq (1+\epsilon)^3 \max\{d(u, v), d(y, v)\}$, so if $d(u, v) \geq d(y, v)$ then the lemma is true. So without loss of generality we can assume that $d(y, v) > d(u, v)$ and $e \leq (1+\epsilon)^3 d(y, v)$.

We start with the case when the maximum edge is $\{x, z\}$, i.e. $e = d(x, z)$. Then from the $h = 3$ case we know that $d(x, z) \leq (1+\epsilon)^2 d(x, v) \leq (1+\epsilon)^3 \max\{d(u, x), d(u, v)\}$. If $d(u, v) \geq d(u, x)$ we are finished, so without loss of generality we can assume that $d(u, x) > d(u, v)$ and $d(x, z) \leq (1+\epsilon)^3 d(u, x)$. Now no matter which of the five vertices enters last, we can derive a contradiction using Lemma 3. This is because all of $d(u, x), d(x, z), d(v, y)$ are greater than $d(u, v)$ (which is the only edge in the cycle not in G_t) and are all within a $(1+\epsilon)^3$ factor of each other. $d(z, y)$ is not necessarily larger than $d(u, v)$ or close to the other edges, but it's enough that it is smaller than $d(x, z)$ since that means that it is not too large and if it is very small then we will not have to use it to derive a contradiction. So if u enters last then the $\{u, x\}$ edge would not be added, if x or z enters last then the $\{x, z\}$ edge wouldn't be added, if v enters last then the $\{y, v\}$ edge would not be added, and if y enters last whichever of $\{z, y\}$ and $\{y, v\}$ is considered second would not be added. These are all contradictions, which finishes the case of $e = d(x, z)$.

The final case, when the maximum edge is $\{u, x\}$, is slightly more complicated because now it is not obvious that z has to be incident to a large edge, which we need in order to derive the type of contradictions that we have been using. If $d(x, y) \leq d(v, y)$, though, then this isn't a problem and we can easily derive the same kind of contradictions using Lemma 3: if u or x enters the system last among u, x, y, v (note the exclusion of z) then the $\{u, x\}$ edge will not be added, and if v or y enters the system last of the four then then $\{v, y\}$ edge will not be added. This is because both $d(x, y)$ and $d(u, v)$ are at most $d(v, y)$, which is at most $d(u, x)$, and $e = d(u, x) \leq (1 + \epsilon)^3 d(v, y)$.

So without loss of generality we assume that $d(x, y) > d(v, y)$. We still get an immediate contradiction if the last of the five nodes to enter is not z since we still have that $d(u, v) < d(v, y) \leq d(u, x)$ and $d(y, v) \leq (1 + \epsilon)^3 d(u, x)$ and $d(x, z), d(y, z) \leq d(u, x)$. This implies that if it is u or x which arrives last then we can apply Lemma 3 to the sequence u, v, y, z, x (or its reverse) to get that $\{u, x\}$ will not be added. If it is v then the sequence v, u, x, z, y implies that $\{v, y\}$ will not be added, and if it y then the same around-the-cycle sequence construction implies that whichever of $\{y, z\}$ and $\{y, v\}$ is larger will not be added.

Thus the only difficult case is when z is the last of the five to arrive. Note that we are trying to prove that $d(u, x) \leq (1 + \epsilon)^3 d(u, v)$, so we will assume that $d(u, x) > (1 + \epsilon)^3 d(u, v)$ and derive a contradiction. If $d(v, y) < d(u, x)/(1 + \epsilon)^2$ then the ϵ-three point condition and the fact that $d(v, y) > d(u, v)$ imply that $d(u, y) < d(u, x)/(1 + \epsilon)$. Now the ϵ-three point condition implies that $d(x, y) \geq d(u, x)/(1+\epsilon)$, which in turn implies that $\max\{d(x, z), d(z, y)\} \geq d(u, x)/(1+\epsilon)^2$. But now we have our contradiction, since this means that whichever of the two edges incident on z is considered second is at most $(1 + \epsilon)^2$ times smaller than $d(u, x)$ and thus is also larger than $d(u, v)$, and so will not be added.

So finally we are left with the case that $d(v, y) \geq d(u, x)/(1+\epsilon)^2$. Recall that $d(x, y) > d(v, y)$, so $d(x, y) > d(u, x)/(1 + \epsilon)^2$ and thus $\max\{d(x, z), d(z, y)\} \geq d(u, x)/(1 + \epsilon)^3 > d(u, v)$. So we again have the contradiction that whichever of the two edges incident on z is considered second will not be added by the algorithm. □

With this lemma in hand we can now prove Theorem 7:

Proof of Theorem 7. Suppose that G_{n+1} has a cycle of length $h \leq 6$ consisting of the nodes $x_1, x_2, \ldots, x_h$, where x_1 is the last of the h nodes on the cycle to have entered the system. Without loss of generality, assume that $\{x_1, x_2\}$ is considered by the algorithm before $\{x_1, x_h\}$. Let e be the largest edge on the cycle other than $\{x_1, x_h\}$ and $\{x_1, x_2\}$. Lemma 6 implies that $e \leq (1 + \epsilon)^{h-3} d(x_2, x_h) \leq (1 + \epsilon)^{h-2} \max\{d(x_1, x_2), d(x_1, x_h)\} = (1 + \epsilon)^{h-2} d(x_1, x_h) \leq (1 + \epsilon)^4 d(x_1, x_h)$. Thus every edge on the cycle is at most $(1 + \epsilon)^4 d(x_1, x_h)$, so applying Lemma 3 to $x_1, x_2, \ldots, x_h$ implies that $\{x_1, x_h\}$ would not be added by the algorithm, and therefore would not be in G_{n+1}. Thus G_{n+1} has girth at least 7, so by Lemma 5 we have that the number of edges added by the algorithm is at most $n^{4/3}$, and thus the 4-threshold algorithm has cost at most $n^{1/3}$. □

A natural question is whether our analysis of the girth is tight. We show in the full version that it is: for any k (and thus in particular for $k = 2$ and $k = 4$) and any $\epsilon \geq 0$, there is a semimetric in $\mathsf{3PC}(\epsilon)$ and an ordering of the points such that the graph of all edges inserted by the k-threshold algorithm results in a cycle of length $k + 3$. Obviously we would like to also show that not only is the girth analysis tight, but so is the update cost analysis that we use it for. Unfortunately we are not able to do this, but for good reason. We conjecture that the graph of edges inserted by the k-threshold algorithm has girth at least $k + 3$ for all even integer k, and if this is true then proving the matching lower bound would solve the celebrated Erdös girth conjecture.

3 Dynamic Embedding

The algorithms in the previous section work in the online setting, when nodes enter the system one at a time and then never leave. While slightly more realistic than the standard offline setting, this still does not model real networks particularly well since it does not handle the removal of nodes, which happens all the time in reality. What we would really like is a *dynamic* embedding that can handle both insertions and removals with the same kind of guarantees that we obtained in Theorems 6 and 7. Fortunately it turns out that insertions are the difficult case, and removals are easy to handle by just allowing Steiner nodes in the embedding.

Our removal algorithm is as follows. We say that a node in the embedding is *active* if it is currently in the system and *inactive* if it is not. Nodes can change from being active to being inactive, but not from being inactive to active. Suppose that u leaves the system. Let a be the number of active nodes after u leaves, and let b be the number of inactive nodes. If $a \leq \alpha(a + b)$ (for some parameter $0 < \alpha < 1$ to be specified later) then remove all inactive nodes and let T be a minimum spanning tree on the active nodes. We call this a *cleanup* step. Otherwise do nothing, in which case u stays in the current tree as an inactive (i.e. Steiner) node. When a new node u enters the system we use the k-threshold algorithm to insert it, i.e. we consider the distances from u to the other active nodes in non-decreasing order and perform an edge swap if the current expansion is greater than $(1 + \epsilon)^k$. We call this the k-dynamic algorithm.

It is easy to see that Lemma 4 basically hold in this setting; it just needs to be changed to only hold for paths for which both of the endpoints are active. The proof of the lemma still goes through since for a path $u - x - v$ in which u and v are active, when the path was created x had to be active as well (since when an edge is formed its endpoints are active, so no matter which of the three nodes arrives last x must be active). Thus Theorem 6 still holds since when a cycle is formed in the G_t by the addition of a node u, clearly the two nodes adjacent to u in the cycle must be active so the modified version of Lemma 4 applies.

It is slightly more complicated to see that Lemma 6 still holds with the same change, that its guarantee only applies to paths where the endpoints are both active. The arguments have to be made much more carefully because we can only use smaller cases (e.g. depending on the $h = 3$ case when proving the $h = 4$ case) when we are guaranteed that both endpoints of the subpath that we are applying it to are active. We also cannot assume that any distance involving an inactive node is expanded by only $(1 + \epsilon)^k$, so property 3 of Lemma 3 has to be changed so that if $\{u_i, u_{i+1}\}$ has never been inserted by the algorithm then $d(u_i, u_{i+1}) \leq d(u, v)$ and u_i and u_{i+1} are both active. Due to the similarity with the proof of Lemma 6, we defer the proof to the full version.

We now show that this algorithm has small distortion in terms of the number of nodes actually in the system. Setting $\alpha = \frac{1}{2}$ gives a reasonable tradeoff, so that is what we will do. Due to space constraints we defer the proofs of theorems in this section to the full version, but we note that combining Theorems 8 and 9 for $k = 4$ gives us Theorem 2.

Theorem 8. *Suppose that at time t there are n nodes in the system and T is the tree that the k-dynamic algorithm is currently maintaining. Then for all x, y in the system, $\frac{d(x,y)}{(1+\epsilon)^{\lceil \log n \rceil + 1}} \leq d_T(x, y) \leq (1 + \epsilon)^k d_T(x, y)$*

Define the cost of an operation to be the number of edge insertions that it requires, so the cost of an insertion is the same as in the online case and the cost of a removal is 0 if it does not result in a cleanup and the $a - 1$ if it does (recall that a is the number of active nodes).

Theorem 9. *The amortized cost of an operation in the k-dynamic algorithm (for $k = 2$ or 4) is at most $O(n_{max}^{2/(k+2)})$, where n_{max} is the largest number of nodes that are ever in the system at one time.*

4 Conclusion and Open Questions

We have shown that when a semimetric satisfies an ϵ-three point condition (or equivalently has the property that every subset of size three embeds into an ultrametric with distortion at most $1 + \epsilon$) it is possible to embed it online into a single ultrametric in a way that has both small distortion and small "structural change". We measure structural change by the number of edges that have to be added to the tree representing the ultrametric, which in a distributed setting is a natural goal since it results in small update messages when the network changes. Furthermore, a trivial example shows that the best offline embedding might require very large structural change, while we get a polynomial reduction in the structural change by losing only a constant factor in the distortion.

The obvious open question is whether the k-threshold algorithm is good for all values of k. In particular, if the analog of Lemma 6 holds for all k then the $\Theta(\log n)$ threshold algorithm would have update cost at most $O(1)$ and distortion only $(1+\epsilon)^{O(\log n)}$. We believe that this is true, but our techniques do not extend past $k = 4$. In particular, the proof of Lemma 6 proceeds via case analysis, and when k gets large there are just too many cases. Nevertheless, we conjecture that for even k the graph of all edges ever inserted by the k-threshold algorithm has girth at least $k + 3$, and thus has update cost at most $n^{\frac{2}{k+2}}$.

Acknowledgements. We would like to thank Dahlia Malkhi for pointing out this line of research and for many helpful discussions.

References

1. I. Abraham, M. Balakrishnan, F. Kuhn, D. Malkhi, V. Ramasubramanian, and K. Talwar. Reconstructing approximate tree metrics. In *PODC '07: Proceedings of the twenty-sixth annual ACM Symposium on Principles of Distributed Computing*, pages 43–52, New York, NY, USA, 2007. ACM.
2. I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete Comput. Geom.*, 9(1):81–100, 1993.
3. S. Arora, L. Lovász, I. Newman, Y. Rabani, Y. Rabinovich, and S. Vempala. Local versus global properties of metric spaces. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM Symposium on Discrete Algorithm*, pages 41–50, New York, NY, USA, 2006. ACM.
4. B. Bollobás. *Extremal graph theory*, volume 11 of *London Mathematical Society Monographs*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1978.
5. M. Charikar, K. Makarychev, and Y. Makarychev. Local global tradeoffs in metric embeddings. In *FOCS '07: Proceedings of the forty-eighth annual IEEE Symposium on Foundations of Computer Science*, pages 713–723, 2007.
6. M. Costa, M. Castro, A. Rowstron, and P. Key. Pic: Practical internet coordinates for distance estimation. In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 178–187, Washington, DC, USA, 2004. IEEE Computer Society.
7. F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. *SIGCOMM Comput. Commun. Rev.*, 34(4):15–26, 2004.
8. M. Gromov. Hyperbolic groups. In *Essays in group theory*, volume 8 of *Math. Sci. Res. Inst. Publ.*, pages 75–263. Springer, New York, 1987.
9. H. Lim, J. C. Hou, and C.-H. Choi. Constructing internet coordinate system based on delay measurement. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, pages 129–142, New York, NY, USA, 2003. ACM.
10. L. Tang and M. Crovella. Virtual landmarks for the internet. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, pages 143–152, New York, NY, USA, 2003. ACM.
11. M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24 (electronic), 2005.
12. L. wei Lehman and S. Lerman. Pcoord: Network position estimation using peer-to-peer measurements. In *NCA '04: Proceedings of the Network Computing and Applications, Third IEEE International Symposium on (NCA'04)*, pages 15–24, Washington, DC, USA, 2004. IEEE Computer Society.

A Distributed Embedding

We consider the setting in which there is an underlying routing fabric, e.g. the Internet, over which all messages are sent. Each node would like to know not only its distance from the other nodes in the network, but also their distances from each other, i.e. each node wants a global view of the network. One trivial solution would be for every node to store all $\binom{n}{2}$ distances in the network. This has two drawbacks: each node needs to use $\Theta(n^2)$ words to store all of the distances (we assume that a single distance can be stored in a word), and when a new node joins the network it has to send a message to every other node with all of its distances, so $\Omega(n^2)$ words need to be sent across the network ($n-1$ distances need to be sent to all $n-1$ other nodes). A natural way to improve this is for each node to store a copy of an embedding, which hopefully will take less space to store and will cost little to update if the addition of a node does not change it much.

We would like an embedding that has low distortion, but can also be efficiently updated when nodes join and leave. Our notion of "efficiency" is communication complexity, namely the number of bits or words that have to be sent across the network to update the embedding. But this is exactly the measure that our online and dynamic results bound!

More formally, suppose that we are in the online setting, and there are n insertions total. We assume that it takes one word to contain either a node ID or a distance in the network. We also assume that all messages use the routing fabric, so each message has to be sent individually (i.e. we do not allow network coding, multicast, or any other technique which uses some knowledge of the underlying fabric). Then the *communication complexity* of a distributed algorithm is the total number of words sent across the network. So the communication complexity of the naive algorithm is $\Omega(n^3)$, since for each of the n operations we need to send a message of $\Omega(n)$ words to $\Omega(n)$ different nodes.

On the other hand, consider the following algorithm. When a node joins the network it first sends a request to another node for the current embedding, which when represented as a tree requires $O(n)$ words of communication. It then probes all other nodes in the network to find out its distance to all of them, resulting in another $O(n)$ words. Once it knows all of its distances it runs the 4-threshold algorithm locally, keeping track of every edge swap made. When it has finished running the algorithm it sends out a list of these edge swaps to every node in the network. The amortized number of swaps is only $O(n^{1/3})$ by Theorem 7, and thus sending this message to every node requires $O(n^{4/3})$ words of communication. Doing this for every node that arrives gives us the following theorem.

Theorem 10. *The above distributed embedding allows every node to estimate the distance between any two nodes up to a distortion of* $(1+\epsilon)^{\lceil \log n \rceil + 4}$ *while only using space* $O(n)$ *at each node and communication complexity* $O(n^{7/3})$ *in the online arrival model.*

It is easy to see that $\Omega(n^2)$ is a lower bound on the communication complexity of any distributed online embedding, since when a new node arrives it has to at the very least notify every other node in the network that it has arrived. Even assuming that this notification only takes one word, this means that the total communication complexity is at least $\sum_{i=1}^{n}(i-1) = \Omega(n^2)$.

Similarly, we can use Theorems 8 and 9 to give a similar result for the dynamic case, with distortion $(1+\epsilon)^{\lceil \log n \rceil + 5}$ and communication complexity $O(tn_{max}^{4/3})$, where t is the total number of operations.

B Proofs for Dynamic Case

In this appendix we give proofs for the two theorem from Section 3.
Proof of Theorem 8. Let N be the number of nodes in the embedding (so $N \geq n$). Our algorithm explicitly maintains the invariant that $n \geq \alpha N$, since if after a removal the invariant is false then the cleanup step makes it true again. Since T is on N nodes, Lemma 1 implies that $d(x,y) \leq (1+\epsilon)^{\lceil \log N \rceil} d_T(x,y) \leq (1+\epsilon)^{\lceil \log(n/\alpha) \rceil} d_T(x,y)$, which proves the first inequality since we set $\alpha = \frac{1}{2}$. Let $t' \leq t$ be the last time that a cleanup step was performed. Then immediately after the cleanup the tree is a MST, so if x and y were both active at time t' then by Theorem 4 the expansion between at time t' is at most 1, and thus by Lemma 2 their expansion is at most 1 at time t as well. Now let x and y be active at time t, but suppose that x was not in the system at time t'. Then whichever of x and y entered last considered adding the edge between them, so by the definition of the k-threshold algorithm the expansion between them is at most $(1+\epsilon)^k$ and by Lemma 2 does not increase after that. □
Proof of Theorem 9. Let t_i be the time of the ith cleanup operation. We break up the operation of the algorithm into intervals corresponding to times between cleanup operations, so the first interval is from $t_0 = 1$ until t_1, the ith interval is from t_{i-1} to t_i, and the final interval is from the last cleanup operation until the final operation of the algorithm. Let i be an interval other than the final one, so there is a cleanup step at t_i. Let a_i be the number of active nodes in the embedding just before the cleanup at t_i, let b_i be the number of inactive nodes, and let $n_i = a_i + b_i$ be the number of nodes in the embedding before the cleanup. Since we are performing a cleanup we know that $a_i \leq \alpha n_i$ and thus $b_i \geq (1-\alpha)n_i$. Each inactive node results from a remove operation during this interval, so $t_i - t_{i-1} \geq (1-\alpha)n_i$.

We will charge the total cost of this interval (including the cleanup at t_i) to the operations in the interval. We know from Theorems 6 and 7 that the total cost of the node insertions in this interval is at most $n_i^{1+2/(k+2)}$ and the cost of the cleanup is at most $a_i \leq \alpha n_i$, so the total cost of the interval is at most $n_i^{1+2/(k+2)} + \alpha n_i$. Charging this cost to each operation individually, we get that the charge to a each operation is at most $(n_i^{1+2/(k+2)} + \alpha n_i)/(t_i - t_{i-1}) \leq (n_i^{1+2/(k+2)} + \alpha n_i)/((1-\alpha)n_i) \leq \frac{2}{1-\alpha} n_i^{2/(k+2)}$. Since we did not perform a cleanup at time $t_i - 1$ we know that $a_i + 1 \geq \alpha n_i$, which means that the charge is at most $\frac{2}{1-\alpha}(\frac{a_i+1}{\alpha})^{2/(k+2)}$.

Let f be the final interval. Since we do not perform a cleanup at the end of f we know that $a_f \geq \alpha n_f$. Theorems 6 and 7 imply that the total cost of this interval is at most $n_f^{1+2/(k+2)} \leq (\frac{a_f}{\alpha})^{1+2/(k+2)}$. Since obviously a_f and every a_i is at most n_{max} and $t_f \geq n_{max}$, we get that the amortized cost per operation is at most

$$\begin{aligned}\frac{t_f \frac{2}{1-\alpha}(\frac{a_i+1}{\alpha})^{2/(k+2)} + (\frac{a_f}{\alpha})^{1+2/(k+2)}}{t_f} &\leq \frac{2}{1-\alpha}\left(\frac{n_{max}+1}{\alpha}\right)^{2/(k+2)} + \frac{n_{max}^{2/(k+2)}}{\alpha^{1+2/(k+2)}} \\ &\leq O\left(n_{max}^{2/(k+2)}\left(\frac{1}{\alpha^{2/(k+2)}(1-\alpha)} + \frac{1}{\alpha^{1+2/(k+2)}}\right)\right) \\ &= O(n_{max}^{2/(k+2)})\end{aligned}$$

thus proving the theorem. □

C Tightness of Girth Analysis

In this section we show that the k-threshold algorithm can result in a graph with girth $k+3$ when k is even, implying that the analysis of the girth in the previous sections is essentially tight.

The example we use has $k+3$ vertices: v_0 will be the last one to enter the system, and for $i \in [(k+2)/2]$ we have two vertices, $v_{i,0}$ and $v_{i,1}$, which we say are at *level* i. For consistency we will say that v_0 is at level 0. The distances between the nodes are given by the following rules:

1. $d(v_0, v_{i,a}) = (1+\epsilon)^{2(i-1)}$ for all i, a
2. $d(v_{i,0}, v_{i,1}) = (1+\epsilon)^{2i-1}$ for all i
3. $d(v_{i,a}, v_{(i+1),a}) = (1+\epsilon)^{2i-1}$ for all i
4. $d(v_{i,a}, v_{j,b}) = (1+\epsilon)^{2(j-1)}$ for all i, a, j, b with $i < j$ not covered by the previous rules

Let $h = (k+2)/2$. The nodes enter in "decreasing" order: the first node is $v_{h,0}$, then $v_{h,1}$, then $v_{(h-1),0}$, then $v_{(h-1),1}$, etc. until $v_{1,1}$. Finally v_0 enters. Call this ordering π. The claim is that these distances satisfy the ϵ-three point condition, and if the nodes enter according to π then the k-threshold algorithm will result in an all-graph that is a $k+3$ cycle $v_0, v_{1,0}, v_{2,0}, \ldots, v_{h,0}, v_{h,1}, v_{(h-1),1}, \ldots, v_{1,1}, v_0$.

Lemma 7. *The above distances satisfy the ϵ-three point condition*

Proof. Let $x = v_{i,a}, y = v_{j,b}, z = v_{\ell,c}$ be three arbitrary nodes (other than v_0), and without loss of generality assume that $i \leq j \leq \ell$. Suppose that $i = j$, which means that $d(x,y) = (1+\epsilon)^{2i-1}$. If $\ell \neq i+1$ then $d(x,z) = d(y,z) = (1+\epsilon)^{2(\ell-1)} > d(x,y)$, so the ϵ-three point condition is satisfied. If $\ell = i+1$ then c must equal either a or b, so without loss of generality suppose that $c = a$. In this case $d(x,z) = (1+\epsilon)^{2i-1}$ and $d(y,z) = (1+\epsilon)^{2(\ell-1)} = (1+\epsilon)^{2i}$, so $d(x,y) = d(x,z) < d(y,z)$ and $d(y,z) = (1+\epsilon)d(x,z)$, so again the ϵ-three point condition is satisfied. Similarly, if $j = \ell$ then $d(y,z) = (1+\epsilon)^{2(j-1)}$. Without loss of generality, suppose that $a = c$. From rules 3 and 4 we know that

$d(x,z) \leq d(x,y)$, and from rules 2 and 4 we know that $d(x,z) \geq d(y,z)/(1+\epsilon)$ and thus the ϵ-three point condition is satisfied.

Now suppose that $i < j < \ell$. Then it is easy to see that $d(x,z) \geq d(y,z) \geq d(x,y)$, so we need to show that $d(x,z) \leq (1+\epsilon)d(y,z)$. From rule 4 we know that $d(x,z) = (1+\epsilon)^{2(\ell-1)}$. If $\ell = j+1$ and $b = c$ then by rule three $d(y,z) = (1+\epsilon)^{2j-1} = (1+\epsilon)^{2\ell-3}$ in which case $d(x,z) = (1+\epsilon)d(y,z)$ as required. Otherwise from rule four we have that $d(y,z) = d(x,z)$, which satisfies the ϵ-three point condition.

Finally, suppose that $x = v_0$, and y and z are as before. If y and z are at the same level, then $d(x,y) = d(x,z) \geq d(y,z)/(1+\epsilon)$, which proves the lemma. If y and z are not at the same level then $d(x,y)$ is the smallest of the three distances and $d(x,z)$ is the largest, so we need to show that $d(x,z) \leq (1+\epsilon)d(y,z)$. Clearly either rule three or rule four applies to $d(y,z)$. If it is rule three then $d(y,z) = d(x,z)/(1+\epsilon)$, and if it is rule four then $d(y,z) = d(x,z)$, so in either case the ϵ-three point condition is satisfied. Since we have considered all cases, this proves the lemma. □

Lemma 8. *If the nodes enter the system according to ordering π, then the graph of all edges inserted by the k-threshold algorithm is a $(k+3)$-cycle.*

Proof. Suppose that the node $v_{i,0}$ just entered the system. Then according to the ordering π, the only nodes already in the system are the ones at higher levels. It is clear from rules three and four that the smallest distance involving the new node is $d(v_{i,0}, v_{(i+1),0})$, so the k-threshold algorithm will begin by adding that edge. Clearly all other distances involving the new edge are expanded by at most $(1+\epsilon)^k$, so no other edges will be added.

Now suppose that the node $v_{i,1}$ just entered the system. Then $d(v_{i,0}, v_{i,1}) = d(v_{i,1}, v_{(i+1),1})$, and they are both smaller than any other distance involving the new node. Assume that the algorithm makes a bad choice, adding $\{v_{i,1}, v_{(i+1),1}\}$ instead of $\{v_{i,0}, v_{i,1}\}$. After that edge insertion, no distances are expanded by more than $(1+\epsilon)^k$ so no other edges will be inserted.

Thus by the time we try to insert v_0, the current graph is just the path from $v_{1,0}$ up the levels to $v_{h,0}$ then to $v_{h,1}$ then down the levels to $v_{1,1}$. Now when we add v_0, the two smallest distances are to $v_{1,0}$ and $v_{1,1}$, both of which equal 1. Due to symmetry, we can assume without loss of generality that the algorithm tries to add $\{v_0, v_{1,0}\}$ before $\{v_0, v_{1,1}\}$. Then after adding the first edge, the ultrametric distance from v_0 to $v_{1,1}$ is the length of the largest edge on the path, which is $d(v_{h,0}, v_{h,1}) = (1+\epsilon)^{2h-1} = (1+\epsilon)^{k+1}$. Thus the distance between v_0 and $v_{1,1}$ is expanded by more than $(1+\epsilon)^k$, so the k-threshold algorithm will add that edge also. This clearly results in a $(k+3)$-cycle. □

This proof assumed that the k-threshold algorithm breaks ties in the worst possible way. While convenient for exposition, this assumption is not strictly necessary. It is easy to see that you can slightly decrease the inter-level distances by multiples of some δ that is arbitrarily close to 0 without losing Lemma 7, which will force the algorithm to choose the edge to the next highest level instead of the intra-level edge.