

Improved Algorithms for Fully Dynamic Geometric Spanners and Geometric Routing

Lee-Ad Gottlieb *

Liam Roditty †

Abstract

For a set S of points in \mathbb{R}^d , a t -spanner is a sparse graph on the points of S such that between any pair of points there is a path in the spanner whose total length is at most t times the Euclidean distance between the points. In this paper, we show how to construct a $(1 + \epsilon)$ -spanner with $O(n/\epsilon^d)$ edges and maximum degree $O(1/\epsilon^d)$ in time $O(n \log n)$. A spanner with similar properties was previously presented in [6, 8]. However, using our new construction (coupled with several other innovations) we obtain new results for two fundamental problems for constant doubling dimension metrics:

The first result is an essentially optimal compact routing scheme. In particular, we show how to perform routing with a stretch of $1 + \epsilon$, where the label size is $\lceil \log n \rceil$ and the size of the table stored at each point is only $O(\log n/\epsilon^d)$. This routing problem was first considered by Peleg and Hassin [11], who presented a routing scheme in the plane. Later, Chan *et al.* [6] and Abraham *et al.* [1] considered this problem for doubling dimension metric spaces. Abraham *et al.* [1] were the first to present a $(1 + \epsilon)$ routing scheme where the label size depends solely on the number of points. In their scheme labels are of size of $\lceil \log n \rceil$, and each point stores a table of size $O(\log^2 n/\epsilon^d)$. In our routing scheme, we achieve routing tables of size $O(\log n/\epsilon^d)$, which is essentially the same size as a label (up to the factor of $1/\epsilon^d$).

The second and main result of this paper is the first **fully** dynamic geometric spanner with poly-logarithmic update time for both insertions and deletions. We present an algorithm that allows points to be inserted into and deleted from S with an amortized update time of $O(\log^3 n)$.

*Courant Institute, New York University, New York NY, 10012. E-mail: adi@cs.nyu.edu. Author's work supported in part by NSF grant number IDM 0414763.

†Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel. E-mail: liam.roditty@weizmann.ac.il.

1 Introduction

A graph $H \subseteq G$ is a t -spanner of G if $\delta_H(u, v) \leq t\delta_G(u, v)$, where $\delta_G(u, v)$ ($\delta_H(u, v)$) denotes the shortest path distance between u and v in G (H).

In geometric settings G is the complete graph of a set S of n points, where the weights of the edges of G are the distances between the points of S in \mathbb{R}^d . Vaidya [16], Salowe [15] and Callahan and Kosaraju [5] showed how to compute a geometric $(1 + \varepsilon)$ -spanner with $O(n/\varepsilon^d)$ edges in $O(n \log n)$ time. Recently, Har-Peled and Mendel [10] presented a construction of spanners with similar properties for metric spaces with a constant doubling dimension.

Geometric spanners have received much attention in the past. A large number of papers dealing with various aspects of geometric spanners have been published in the last two decades. The problem of constructing geometric spanners with special properties, such as small weight, small hop diameter and small degree, has also been extensively studied. It is well known that a spanner of constant degree can be built in $O(n \log n)$ time (see [8]). Further, Chan *et al.* [6] showed that this result can be generalized to metric spaces of low doubling dimension.

In this paper we present a simple $O(n \log n)$ time construction of a geometric $(1 + \varepsilon)$ -spanner with $O(n/\varepsilon^d)$ edges and maximum degree is $O(1/\varepsilon^d)$. The importance of our new construction lies mostly in its applications. Based on our new construction, and incorporating several other new ideas, we obtain two new results. The first result is an essentially optimal routing scheme for doubling dimension metric spaces, and the second is the first fully dynamic geometric spanner algorithm that breaks the polynomial dependency on n in the update time. That is, we obtain the first fully dynamic geometric spanner that supports updates to S in amortized time poly-logarithmic in the number of points in S .

In what follows, we denote with α the *aspect ratio* of a set of points S – the ratio of the maximum pairwise distance to the minimum pairwise distance of points in S . A metric is said to be of a constant doubling dimension if a ball with radius r can be covered by at most a constant number of balls of radius $r/2$.

Routing is perhaps one of the most fundamental problems in networks. In this paper we consider compact routing schemes, where the goal is to achieve efficient tradeoffs between the size of the routing tables and the stretch of the route on which the messages are sent. There are two versions of this problem. In the labeled model, the designer is allowed to assign to each node a short label that can be used for routing. In the name-independent model an adversary assigns the node labels. We will consider the problem of labeled compact routing on a metric space. In this model the designer must first choose for each node a set of outgoing links, thereby inducing a directed overlay graph on which the routing will transpire. This model was introduced by Peleg and Hassin [11], who considered routing in the plane. Later works considered routing in doubling dimension metric spaces. Almost all previous results required a label size which depends on α , d and ε ; the only result that avoids such dependencies is that of Abraham *et al.* [1]. They presented a $1 + \varepsilon$ routing scheme with label size $\lceil \log n \rceil$ in which the routing table of each point is of size $O(\log^2 n/\varepsilon^d)$. In our new routing scheme we maintain the stretch and label size, while reducing the size of the routing table of each point to only $O(\log n/\varepsilon^d)$. Recently, Konjevod *et al.* [12] obtained a similar result to the one of Abraham *et al.* .

The problem of maintaining a spanner of a set of points S under insertions and deletions has been extensively studied. Arya, Mount and Smid [2] obtained poly-logarithmic update time for a restricted model in which the updates were assumed to be random. That is, a point to be deleted is assumed to have been selected randomly from S , while a point to be inserted is assumed to be a random point of the new point set. In [4], Bose, Gudmundsson and Morin presented an algorithm that supports only insertions in poly-logarithmic time. Recently Gao, Guibas and Nguyen [9] considered dynamic and kinetic geometric spanners, and obtained a fully dynamic geometric spanner with a worst case update time of $O(\log \alpha)$. Most recently, Roditty [14] presented the first fully dynamic algorithm for a geometric $(1 + \varepsilon)$ -spanner with an

update time that is independent of α . Insertions into S are supported in $O(\log n)$ amortized time while deletions require $\tilde{O}(n^{1/3})$ amortized time (where \tilde{O} is used to hide a small poly-logarithmic factor). The spanner has $O(n/\varepsilon^d)$ edges and, as in Gao *et al.* [9], the only assumption is that the metric space has a constant doubling dimension.

In this paper we present a fully dynamic $(1 + \varepsilon)$ -spanner of size $O(n/\varepsilon^d)$. Insertions and deletions to S are supported in $O(\log^3 n)$ amortized time. A very small modification to our construction results in insertions being supported in $O(\log^2 n)$ amortized time while deletions still require $O(\log^3 n)$ amortized time; we defer further details to the full version of this paper.

The backbone of this paper is a new and simple construction of a constant degree spanner. The construction is then used to obtain the improved compact routing scheme. The fully dynamic spanner is obtained by modifying the skeleton of the constant degree spanner and making several new observations concerning the hierarchy of discrete centers. These observations may also be useful for other problems, as the hierarchy of discrete centers is used in numerous applications in geometry.

The rest of this extended abstract is organized as follows: In the next section we review some necessary background for our construction. In Section 3 we describe the construction of a constant degree spanner. In Section 4 we further describe the main tool used in the spanner construction. In Section 5 we describe the compact routing scheme. In Section 6 we present the poly-logarithmic fully dynamic spanner.

2 The hierarchical structure of discrete centers

In this section we describe the spanner construction of Gao *et al.* [9]. The backbone of their construction is a partition of the points set into a hierarchical structure of discrete centers. This partition is similar to the one that was used independently by Krauthgamer and Lee in the context of proximity search [13]. Let S be a set of points in a metric space of doubling dimension d , and α be the aspect ratio of S . A subset of points $C \subseteq S$ is a set of r -discrete centers of S if and only if all points of C are at a minimal distance of r from each other, and every point in S is at a distance of at most r from at least one point of C . The hierarchy of discrete centers is defined as follows. There are $\log \alpha + 1$ sets of discrete centers $\mathcal{C} = \{C^0, C^1, C^2, \dots, C^{\log \alpha}\}$ such that $S = C^0 \supseteq C^1 \supseteq C^2 \dots \supseteq C^{\log \alpha}$, and C^i is a set of 2^i -discrete centers of C^{i-1} . (For simplicity, one assumes that any two points of S are at a distance of at least 1.) It is easy to see that the set $C^{\log \alpha}$ contains a single point.

On top of the hierarchy \mathcal{C} is defined a corresponding tree $T(\mathcal{C})$. Each point in $S = C^0$ corresponds to a leaf node in $T(\mathcal{C})$, and all other occurrences of points in the hierarchy correspond to internal nodes in $T(\mathcal{C})$. The root of the tree is the node that corresponds to the single point in $C^{\log \alpha}$. For a point p , we denote the node that corresponds to $p \in C^i$ as $p(i)$. Also, if $q \in C^i$ and there exists a point $p \in C^{i+1}$ such that $|pq| \leq 2^{i+1}$, then $p(i+1)$ covers $q(i)$, and $q(i)$ is given a directed edge to $p(i+1)$. (If there are multiple points in C^{i+1} that satisfy this property, we give $q(i)$ a directed edge to the closest one.) Then $p(i+1)$ is the parent of $q(i)$, and may be denoted as $P(q(i))$. For a node x , its corresponding point is denoted as $\text{pnt}(x)$ and its level as $\text{lvl}(x)$. Let $p \in S$ and let $N(p) = \{p(d), p(d+1), p(d+2), \dots, p(h)\}$ be an ordered set of nodes from $T(\mathcal{C})$ that correspond to p . The nodes are ordered by their distance from the root, where $p(h)$ is closest to the root. The node $p(h)$ corresponds to the occurrence of p in the set C^h , which is the highest set in the hierarchy that contains p . The node $p(d)$ corresponds to the occurrence of p in the set C^d , which is the deepest set in the hierarchy that contains p . Given the point p let $h(p)$ ($d(p)$) be the level of the highest (deepest) set that contains p . (Later on we will see that the deepest set that contains a point is different for every point and is determined by the edges that a point has in the spanner.)

The spanner G is constructed on top of the hierarchy defined above. Let $p \in S$, and let $p(i) \in N(p)$ be a node that corresponds to p in $T(\mathcal{C})$. (Recall that $p(i)$ is the node that corresponds to the occurrence of p in C^i .) We define the set of edges of $p(i)$ as follows

$$E(p(i)) = \{(p, q) \mid |pq| \leq c \cdot 2^i \wedge q \in C^i\},$$

where c is set to $4 + 16/\varepsilon$. For an edge $e \in E(p(i))$ we have $\text{lvl}(e) = i$. The spanner edges of the point p are $\cup_{j=d(p)}^{h(p)} E(p(j))$. Based on the definition of edges we slightly change the definition of $N(p)$ to reflect the actual degree of p . If $\{p(i), p(i+1), \dots, p(j)\}$ is a maximal sequence of nodes, such that $E(p(k)) = \phi$ for every $k \in [i, j]$, then we merge these nodes into a single node that inherits its incoming edges in the tree from $p(j)$ and its outgoing edges in the tree from $p(i)$. We refer to the new node as a *merged node* and its level indicator returns the pair (i, j) to reflect the range of levels that it covers. (If it happens to be that $i = d(p)$ then we simply set $d(p) = j + 1$.) The number of nodes a point has in the tree is proportional to the number of edges from different levels that touch the point in the spanner. To clarify our notations note that we use p_i to denote the i -th node in the set $N(p) = \{p_1, p_2, \dots, p_i, \dots, p_j\}$ and $p(i)$ to denote the node that corresponds to $p \in C^i$. When we wish to enumerate points and not nodes we use superscript, for example the points p^1, p^2, \dots, p^i .

We now restate a simple lemma which is given in [9].

Lemma 1 (Packing Lemma) *If all points in a set $U \in R^d$ are of at least a distance r away from each other, then there are at most $(2R/r + 1)^d$ points in U within any ball X of radius R .*

It follows from the lemma that the maximum number of children a node $x \in T(C)$ may have is 5^d . We denote this number as β . It also follows from the lemma that the number of edges a node x may have is at most $(1 + 2c)^d$. We denote this number as γ .

The spanner of Gao *et al.* has a stretch of $(1 + \epsilon)$, its size is $O(n/\epsilon^d)$, and it can be constructed in $O(n \log \alpha)$ time. Insertions and deletions are supported in $O(\log \alpha)$ time. In the next section we present an incremental construction of the spanner of Gao *et al.* that will allow us to give a clearer description of the construction of our constant degree spanner, which will be given immediately afterwards.

2.1 An incremental construction of the spanner

Let p be the new point. Assume that we are given the deepest point q such that $q \in C^j$ and $|pq| \leq 2^{j+1}$. (By *deepest* we mean that there is no point $r \in C^k$ for which $|pr| \leq 2^{k+1}$ and $k < j$.) We now begin a search for a point that covers p . If such a point q' exists in level j , then the node that represents it in level j has an edge incident on $q(j)$. This follows from the fact that $|pq'| \leq 2^j$ and from the fact that $|pq| \leq 2^{j+1}$ – hence, $|qq'| \leq |pq'| + |pq| \leq 2^j + 2^{j+1} \leq c2^j$ and $(q, q') \in E(q(j))$. We can therefore scan the list of edges $E(q(j))$ for such a node.

If such a point was not found in level j we continue the search in the edge list of the parent node of $q(j)$ at level $j + 1$. Note that if this node also represents the point q then we are done, since the point q covers p in level $j + 1$. Each time that a covering point is not found the search proceeds to the next level. If no covering point is found in the hierarchy, the hierarchy is extended upwards by increasing the level of the root to a level that allows it to cover p .

Note that in every level visited by the search, the only nodes whose edge lists are scanned are those nodes that represent points that will have an edge with the new point p . Thus, the total cost of the search process is proportional to the final size of $N(p)$. We begin the construction of $N(p)$, as well as the corresponding edge list of its nodes, from the point that covers p . Assume that p is covered in level i . We set $h(p)$ to $i - 1$ and insert the node $p(i - 1)$ in $N(p)$. It is easy to see that if p shares an edge with a point in level $i - 1$, then their covering points share an edge in level i . Thus, to find the edges of $E(p(i - 1))$ we scan the edge list of the point that covers p , and the children of every point in the edge list are tested to see whether they share an edge with p in level $h(p)$. We then proceed to add the necessary nodes to $N(p)$ in deeper levels. In each step we use the edge list of the last node that was added to $N(p)$ to create the edge

list for the new node that will be inserted into $N(p)$. The total cost of inserting p into S is $O(|N(p)|\beta\gamma)$ plus the cost of finding q . Thus, the total cost of this process can be bounded by $O(1/\varepsilon^d \log \alpha)$.

In [14] we showed how to construct the spanner in $O(n \log n)$ time. The run time follows from the fact that q – the deepest point such that $q \in C^j$ and $|pq| \leq 2^{j+1}$ – can be found in time $O(\log n)$. The main idea behind the search for q is to use a standard data structure technique known as the heavy path or centroid path decomposition to decompose $T(\mathcal{C})$. To create the first heavy path we begin with the root of the hierarchy which becomes the first node on the path. The next node on the path is the child of the root with the largest number of descendants. We repeat on this process until we reach a leaf. We then have one heavy path. We continue this decomposition in every node whose parent is on the previously computed heavy path. Each heavy path is stored in a biased skip list [3, 7], where the weight of a node is the number of nodes in the subtree rooted at the node minus the weight of the node’s heavy path child. The weight of the path is the total number of nodes on the path plus their weight. It is shown in [14] (based on an idea similar to the one used by [7]) that given p, q can be found in $O(\log n)$ time. Combining this with the fact that the total number of edges in the spanner is $O(n/\varepsilon^d)$, we derive that the cost of constructing the spanner is $O(n \log n)$. We refer the reader to [14] for exact details. The resulting spanner may have points whose degree is $\Omega(\gamma \cdot \log \alpha)$. In what follows we show how to construct a constant degree spanner.

3 A constant degree spanner

In this section we show how to construct a spanner with constant degree in $O(n \log n)$ time. Let $\Delta = 3 \log c + 2$, where c is set to $4 + 16/\varepsilon$. (The purpose for this choice will become clear in Section 4.) We present a technique for reducing the degree of points whose degree is too large. We refer to this technique as the *degree reduction process*. This process enables us to reduce the number of nodes in the tree that correspond to the same point to $O(\Delta)$. The main ingredient used by the degree reduction process is *artificial covering*, in which a point p is covered artificially in a certain level by a nearby point z , thus preventing p from appearing in higher levels. The degree reduction process is applied to every point p whose set $N(p)$ has more than 9Δ nodes.

In our construction of a constant degree spanner we combine the following two ideas. The first idea is to allow a point to be in two separate, non-intersecting sequences of levels. Recall that in the original hierarchical structure a point p is a discrete center in each and every set $C^{d(p)}, \dots, C^{h(p)}$. In the hierarchical structure that we will employ to create the spanner, a point will have a *real* instance denoted as p^r and an *imaginary* instance denoted as p^{im} . The point p is a discrete center in each of the distinct sets $C^{d(p^r)}, \dots, C^{h(p^r)}$ and $C^{d(p^{im})}, \dots, C^{h(p^{im})}$, where $d(p^r) < h(p^r) < d(p^{im}) < h(p^{im})$, $h(p^r) - d(p^r) = O(\Delta)$ and $h(p^{im}) - d(p^{im}) = O(\Delta)$. If a point has only one instance, then we will consider that instance to be a real instance. Roughly speaking, p ’s real instance ensures a $(1 + \varepsilon)$ distance approximation to other points, and p ’s imaginary instance bounds the degree of a different point by artificially covering it. The second idea is to reduce the separation requirement of discrete centers. Recall that in the hierarchy employed by Gao *et al.* any two points in C^i are separated by a distance of 2^i . In our hierarchy points that are already in the spanner are required to be separated in C^i by a distance of only 2^{i-1} . We refer to a separation of 2^i as a *strong* separation and to a separation of 2^{i-1} as a *weak* separation.

The input to our construction is the spanner of Gao *et al.* It follows from the previous section that such a spanner can be constructed in $O(n \log n)$ time. The first step of our construction is to find all points whose set of nodes is larger than 9Δ . We order these points by their depth in the hierarchy. Then, we perform a degree reduction process for each point in the list, starting with the deepest one.

The process of degree reduction proceeds as follows. Let $N(p) = \{p_1, p_2, \dots, p_{3\Delta}, \dots, p_{\ell-3\Delta}, \dots, p_\ell\}$, where $\text{lvl}(p_1) = d(p)$, $\text{lvl}(p_\ell) = h(p)$ and $\ell > 9\Delta$. Assume that p is the deepest point whose set of nodes has more than 9Δ nodes. Our goal is to reduce the number of nodes that correspond to p to 6Δ , where half of these

nodes correspond to the real part of p and half correspond to the imaginary part of p . More precisely, we wish to produce $N(p^r) = \{p_1, p_2, \dots, p_{3\Delta}\}$ and $N(p^{im}) = \{p_{\ell-3\Delta+1}, \dots, p_\ell\}$. We obtain this by imposing an artificial cover point on p in level $\text{lvl}(p_{3\Delta}) + 1$. Then all the nodes that represent p in levels higher than $\text{lvl}(p_{3\Delta})$ are simply removed from $N(p)$. It will become clear later why we leave the last 3Δ nodes of p as its imaginary part.

The point to be inserted into $C^{\text{lvl}(p_{3\Delta})+1}$ as p 's artificial cover is found using the artificial cover point search described in the next section. The artificial cover search receives as input a point p and the level η in which p must be covered. The search returns in $O(\log n)$ time a point z which has not been previously used as an artificial cover, and for which $|pz| \leq 2^{\eta-\Delta}$. We insert z into level $\text{lvl}(p_{3\Delta}) + 1$ and then into each level above it until $N(z)$ has 3Δ nodes. Note that this is the second appearance of z in the hierarchy, and hence this is the imaginary instance of z which is denoted with z^{im} . It follows from the next lemma that using the edge lists of p , we can construct the set $N(z^{im})$ and the corresponding edge lists.

Lemma 2 *Let $p \in C^a, \dots, C^b$ in the hierarchy that corresponds to the spanner of Gao et al. , and let z be a point such that $|pz| \leq 2^{a-1}$. Let $x \in C^i$, where $i \in [a, b-1]$. If $|xz| \leq c2^i$ then $(p, \text{pnt}(P(x))) \in E(p(i+1))$.*

Proof: We know that $|pz| \leq 2^i$, $|xz| \leq c2^i$ and $|x \text{pnt}(P(x))| \leq 2^{i+1}$. Thus, $|p \text{pnt}(P(x))| \leq |pz| + |x \text{pnt}(P(x))| + |xz| \leq 2^i + 2^{i+1} + c2^i \leq c2^{i+1}$. (Recall that $c = 4 + 16/\epsilon$.) \square

Since $|pz| \leq 2^{\text{lvl}(p_{3\Delta})}$, it follows from the above lemma that the construction of both $N(z^{im})$ and the edges of z^{im} in level $j \geq \text{lvl}(p_{3\Delta}) + 1$ can be accomplished by scanning the edge list of p in level $j + 1$ and testing whether the children of each point in the list have an edge with z . Once $N(z^{im})$ reaches 3Δ nodes, we must search for a different artificial cover for z^{im} . Let¹ $N(z^{im}) = \{z_1^{im}, z_2^{im}, \dots, z_{3\Delta}^{im}\}$; we set $d(z^{im}) = \text{lvl}(z_1^{im})$ and $h(z^{im}) = \text{lvl}(z_{3\Delta}^{im})$. Then we initiate a search for an artificial cover point for z^{im} starting at level $h(z^{im}) + 1$. This is done in the same manner as was done for p . The point that artificially covers z^{im} in level $h(z^{im}) + 1$ is at a distance of at most $2^{h(z^{im})-\Delta}$ from z^{im} , which implies that it is at a distance of at most $2^{h(z^{im})}$ from p . Thus, by Lemma 2, we can compute its corresponding nodes and edges in level $h(z^{im}) + 1$ and above using p . A similar argument holds for the entire series of artificial cover points that will be used to replace p .

Note that when z^{im} artificially covers p , it also must cover the points between level $d(z^{im})$ and level $h(z^{im})$ that p had previously covered. Since $|z^{im}p| \leq 2^{d(z^{im})}$, a point q from level $j - 1$ that was covered by p in level $j \in [d(z^{im}), h(z^{im})]$ will be covered by z^{im} either in level j or in level $j + 1$. In the later case q is inserted into level j and a corresponding node is inserted into $N(q)$. It follows that an artificial cover process may increase the number of nodes corresponding to a point, but only by one. Note also that q may violate the strong separation in level j , but will not violate the weak separation in that level. (A point that is covered in level $h(z^{im})$ by p is either covered by z^{im} in that level or by the artificial cover of z^{im} in the next level). Once an artificial cover point covers some point, it will never be replaced by some other point. This is true for any artificial cover that participates in the degree reduction process for p . An artificial cover may also create new edges. However, such edges only touch their opposite endpoints in their top $\log c$ levels, and thus their effect is negligible.

The degree reduction process of p terminates when we reach an artificial cover \bar{z}^{im} which satisfies $h(\bar{z}^{im}) \geq d(p^{im})$. Let y be the point that \bar{z}^{im} artificially covers. Instead of using \bar{z}^{im} to cover y we use p^{im} . Thus, we insert y into higher and higher levels until it is covered by p^{im} . Once the degree reduction process of p is finished, we move to the next point in the list of points whose nodes set has more than 9Δ nodes. We stop when we have performed degree reduction for every point in the list.

Note that $|y\bar{z}^{im}| \leq 2^{d(\bar{z}^{im})-\Delta}$. Thus, any edge that y has in level i corresponds to an edge that \bar{z}^{im} has in level i as well or in levels $i + 1$ or $i - 1$. It implies that the number of nodes that correspond to y in the

¹ z_b^a refers to the b -th node of $N(z^a) = \{z_1, \dots, z_b, \dots\}$

range $[d(\bar{z}^{im}), h(\bar{z}^{im})]$ is at most three times the number of nodes that correspond to \bar{z}^{im} in that range. Thus, even after the addition of nodes to $N(y)$ its size is $O(\Delta)$.

There are two more issues that we need to address. The first is that during the degree reduction process when an artificial cover is inserted into a level it may violate the strong separation requirement of two discrete centers in that level. To this end we have relaxed the separation requirement that the points in C^i has to fulfill, and only a weak separation of 2^{i-1} is required. We can show that a weak separation is fulfilled by the artificial cover. A point z that is inserted into level i is at most $2^{i-\Delta}$ away from p . When p was originally in this level it fulfilled a strong separation, thus, z cannot be at a distance of less than 2^{i-1} from some other point in that level. The weak separation increases the number of edges a point may have per level, but only by a constant factor.

The second issue is updating the underlying data structure of biased skip list that maintains the centroid path decomposition of the hierarchy. During the degree reduction process at most 3Δ nodes that correspond to a certain artificial cover are added to the hierarchy. Considering a specific such node, the cost of updating the underlying data structure is $O(\log n)$ using standard implementation of biased skip list, see [3, 7]. Although the biased skip lists were originally required solely for the initial construction of Gao et al., we insist on updating them for use in a different task: The $O(\log n)$ search for an artificial cover requires us to determine the number of tree descendants of a given point, and this information can be obtained easily from the biased skip list structure.

In the next theorem we summarize the main result of this Section. Let H be the resulted graph.

Theorem 3 *The graph H is a $(1 + \varepsilon)$ spanner whose maximum degree is $O(1/\varepsilon^d)$. Its size is $O(n/\varepsilon^d)$, and the construction time of the spanner is $O(n \log n)$.*

Proof: Stretch. We show that if $x, y \in S$ then $\delta^H(x, y) \leq (1 + \varepsilon)|xy|$. Let i be the minimal integer that satisfies $|xy| + 2^{i+2} \leq c2^i$. Let \hat{x} be the ancestor of x in level i and let \hat{y} be the ancestor of y in level i . The distance from x to \hat{x} is at most 2^{i+1} , and similarly, the distance from y to \hat{y} is at most 2^{i+1} . Thus, $|\hat{x}\hat{y}| \leq |xy| + 2^{i+2} \leq c2^i$ and there is an edge between \hat{x} and \hat{y} in H . Notice also that since every point is connected to its parent in the spanner there is always a path in H between a point and its ancestor in level i whose length is 2^{i+1} . We can bound $\delta^H(x, y)$ as follows: $\delta^H(x, y) \leq |x\hat{x}| + |\hat{x}\hat{y}| + |\hat{y}y| \leq 2^{i+2} + |\hat{x}\hat{y}| \leq 2^{i+3} + |xy|$. We also know that $|xy| + 2^{i+1} > c2^{i-1}$ as i is the minimal integer that satisfies $|xy| + 2^{i+2} \leq c2^i$. Combining it all together we get:

$$\frac{\delta^H(x, y)}{|xy|} \leq \frac{2^{i+3} + |xy|}{|xy|} \leq 1 + \frac{2^{i+3}}{c2^{i-1} - 2^{i+1}} =$$

$$1 + \frac{2^{i+3}}{(16/\varepsilon + 4) \cdot 2^{i-1} - 2^{i+1}} = 1 + \varepsilon$$

Degree. A point may be used as an artificial cover only once (this is demonstrated in the next section), thus, each point has at most two instances. In each instance a point participates in $O(\Delta)$ different levels. In every level it may have γ edges. Hence, the degree of a vertex in H is at most $O(\Delta/\varepsilon^d) = O(1/\varepsilon^d)$

The proof that the size of the spanner is $O(n/\varepsilon^d)$ and the construction time is $O(n \log n)$ is deferred to the full version. \square

4 The search for an artificial cover

The artificial cover search algorithm receives as input a point p and a level i . It finds in $O(\log n)$ time a point z for which $|zp| \leq 2^{i-\Delta}$. The artificial cover search also guarantees that z has not been previously used as an artificial cover of some other point.

Let $p \in S$ and let $N(p) = \{p_1, p_2, \dots, p_{3\Delta}, \dots, p_\ell\}$, where $\text{lvl}(p_1) = d(p)$ and $\text{lvl}(p_\ell) = h(p)$ and $\ell > 9\Delta$. Let p be the deepest point in the hierarchy whose set $N(p)$ has at least 9Δ nodes. Let $\eta = \text{lvl}(p_{3\Delta})$. Our goal is to find a point z for which $|zp| \leq 2^{\eta-\Delta}$. When z is inserted into $C^{\eta+1}$ it covers p and the number of nodes that remain in $N(p)$ is 3Δ .

The search proceeds as follows. The nodes $p_1, \dots, p_{3\Delta}$ remain in $N(p)$ and the nodes $p_{\Delta+2\log c}, \dots, p_{2\Delta-2\log c}$ are the starting point of our search. We search in their edge lists for an edge whose other endpoint is a real instance. If such an edge is found then its endpoint is set to be z . If none of the edges satisfy this requirement, then search proceeds by picking one of these edges and moving to its other endpoint, which is known to be an imaginary instance. (The edge is picked according to a decision rule that we will be described later.) The same procedure is repeated now with the new point. We will show that choosing any of these edges would guarantee that the search will terminate with a real, unused point. However, to ensure a good runtime the choice of edge must be made prudently. The search algorithm is given in Figure 1.

The rest of this section is organized as follows. First, we show that the search indeed finds a real point at a distance of at most $2^{\eta-\Delta}$ away from p . Then, we prove the uniqueness property of the search, that is, a point returned by the algorithm could not have been previously used as an artificial cover point. We conclude by demonstrating that the search takes $O(\log n)$ time (this part is differ to the full version due to space limitations).

The following lemma stems from the way that the search is performed and is used to prove the next lemma.

Lemma 4 *Let $S^p = \{z^0, z^1, \dots, z^\ell, \dots\}$ be a sequence of points that were visited by a search. We have $2^{\text{lvl}(z^i)} \leq |z^i z^{i+1}| \leq 2^{\text{lvl}(z_{2\Delta}^i) - \log c}$ and $h(z^{i+1}) - \log c \leq \text{lvl}((z^i, z^{i+1})) \leq h(z^{i+1})$. (Recall that $\text{lvl}(e)$ is the level of the edge e .)*

Proof: The edge (z^i, z^{i+1}) is from the edge list of one of the nodes $z_{\Delta+2\log c}^i, \dots, z_{2\Delta-2\log c}^i$. Thus, it must be at least of length $2^{\text{lvl}(z_{2\Delta}^i)}$ and at most of length $2^{\text{lvl}(z_{2\Delta}^i) - \log c}$. As this edge leads to z^{i+1} it must be that $\text{lvl}((z^i, z^{i+1})) \leq h(z^{i+1})$. It cannot be that $h(z^{i+1}) - \log c > \text{lvl}((z^i, z^{i+1}))$ as a conflict will arise between z^i and z^{i+1} . \square

We now prove that the algorithm finds a real point.

Lemma 5 *Let $p \in S$. The algorithm **artificial-cover** returns a real point z , that satisfies $|zp| \leq 2^{\eta-\Delta}$.*

Proof: Let $p = z^0, z^1, \dots, z^i, \dots$ be the sequence of points that were visited by the search. By combining Lemma 4 with the fact that $\text{lvl}(z_{\Delta}^{i-1}) \geq \text{lvl}(z_{2\Delta}^i) + \Delta - \log c$ we get:

$$\frac{|z^{i-1} z^i|}{|z^i z^{i+1}|} \geq \frac{2^{\text{lvl}(z_{\Delta}^{i-1})}}{2^{\text{lvl}(z_{2\Delta}^i) - \log c}} \geq 2^\Delta$$

It follows that the length of edges along the search path always decreases. This guarantees that the search is finite. Since the search cannot terminate at an imaginary point (such a point always has edges in which the search may continue), eventually a real point z is reached. The distance from p to z^1 is at most $2^{\text{lvl}(z_{2\Delta}^1) - \log c}$. Recall that $\eta = \text{lvl}(p_{3\Delta})$, and hence, we have $|pz^1| \leq 2^{\eta-\Delta-\log c}$. As every edge along the search path is smaller than the previous edge by a factor of 2^Δ it follows that $|pz| \leq 2^{\eta-\Delta}$. \square

We now show that a point cannot be found as an artificial cover of two different points.

Lemma 6 *A point is returned from an execution of the artificial cover search at most once.*

Proof: Let $S^p = \{z^0, z^1, \dots, z^i, \dots, z^{\ell_1}\}$ be the sequence of points that were visited by the search initiated from p and let $S^q = \{y^0, y^1, \dots, y^j, \dots, y^{\ell_2}\}$ be the sequence of points that were visited by the search initiated from q . Assume for the sake of contradiction that the sequence of points z^i, \dots, z^{ℓ_1} equals

to y^j, \dots, y^{ℓ_2} . For simplicity of presentation let $r = z^i = y^j$, $s = y^{j-1}$ and $t = z^{i-1}$. We examine the edges (y, x) and (z, x) . We know that $|yx| \leq c2^{\text{lvl}((y,x))}$ and $|zx| \leq c2^{\text{lvl}((z,x))}$. From Lemma 4 we know that $\text{lvl}((z, x))$ and $\text{lvl}((y, x))$ are in the range $[h(x) - \log c, h(x)]$. This implies that $|\text{lvl}((y, x)) - \text{lvl}((z, x))| \leq \log c$, and hence, we get:

$$\begin{aligned} |yz| &\leq c2^{\text{lvl}(y,x)} + c2^{\text{lvl}(z,x)} \leq c2^{\text{lvl}(y,x)} + c2^{\text{lvl}(y,x) + \log c} \\ &\leq c2^{\text{lvl}(y,x) + \log c + 1} = 2^{\text{lvl}(y,x) + 2 \log c + 1}. \end{aligned}$$

We know that the point y appears in every level from level $\text{lvl}((y, x))$ till level $h(y)$, where $h(y) \geq \text{lvl}((y, x)) + \Delta$. The point z appears in every level from level $d(z) \leq \text{lvl}((z, x)) - \Delta$ till level $h(z) \geq \text{lvl}((z, x)) + \Delta$. Combining it with the fact that $|\text{lvl}((z, x)) - \text{lvl}((y, x))| \leq \log c$ we get that $d(z) \leq \text{lvl}((y, x)) + \log c - \Delta$ and $h(z) \geq \text{lvl}((y, x)) - \log c + \Delta$. If $\Delta \geq 3 \log c + 2$ we get a contradiction as both z and y are at level $\text{lvl}((y, x)) + 2 \log c + 2$ although they are too close to be included together in this level as they are at a distance of at most $2^{\text{lvl}((y,x)) + 2 \log c + 1}$. \square

We have shown that the artificial cover search returns a point that is at most $2^{\eta - \Delta}$ away from the input point. We further proved the uniqueness property of the search. We need only demonstrate that the search terminates in $O(\log n)$ time. Due to space constraints we relegate the rest of this section (which is rather technical) to the full version.

5 An optimal compact routing scheme in metric spaces

In this section, we turn to the problem of routing information between points in S . We present an optimal routing scheme that routes on the spanner, and therefore achieves a $1 + \epsilon$ routing *stretch*. (The routing stretch between two points p and q is the ratio of their interpoint routing distance to their true distance. The routing stretch of a scheme is the maximum routing stretch among the points.) This scheme requires each point to store only $1/\epsilon^d$ labels of (optimal) size $\lceil \log n \rceil$, yielding a storage bound of $O(\log n/\epsilon^d)$ per point. This improves upon the result of Abraham *et al.* [1] for labeled routing schemes, as their construction has storage size $\Theta(\log^2 n/\epsilon^d)$.

Label assignment. We begin by describing our label assignment, which mimics the one employed by [1]. Recall that we arranged the points in a tree defined by the hierarchy, and merged to removed nodes with no edges. The labels are assigned using an Eulerian tour of this tree. Each leaf encountered by the tour is assigned a value based on the order it was encountered, with the first leaf given the value 0 and each subsequent leaf given the value incremented by one. This value is the label of the leaf, and is stored with $\lceil \log n \rceil$ bits. Now each internal node is assigned two labels (each of size $\lceil \log n \rceil$) that together constitute its *range*: The first label is equal to the arithmetically smallest label among the node's children, and the second value is equal to the arithmetically largest value among the children. (Note that, unlike the construction of [1], we assign labels to every internal node.)

It follows then that an internal node with labels ℓ_1 and ℓ_2 (and range $[\ell_1, \ell_2]$) possesses $\ell_2 - \ell_1$ leaf descendants. Further, it follows that the label of each leaf descendant, say ℓ , falls in the range $[\ell_1, \ell_2]$: $\ell_1 \leq \ell \leq \ell_2$. This crucial property of the labeling scheme allows the tree to support ancestor queries with only two label comparisons: A leaf node v with label ℓ is the descendant of internal node u with range $[\ell_1, \ell_2]$ if and only if $\ell_1 \leq \ell \leq \ell_2$.

For each node, its labels are stored in the corresponding point. Further, the labels of all nodes with edges incident on this node are also stored in the corresponding point. Since a point p appears in $|N(p)|$ positions in the tree, it stores no more than $2|N(p)|/\epsilon^d$ labels. It follows that the entire scheme requires $O(\log n/\epsilon^d)$ space per point, and $O(n \log n/\epsilon^d)$ space overall.

Routing algorithm. We now present the routing scheme, which utilizes the labeling to route messages on the spanner. Point p wishes to route a message to point q , but p knows only the label ℓ associated with the leaf instance of q . p must discover the path to q in the spanner. The search for the path proceeds as follows:

p iterates through the nodes that correspond to its real instance, that is the set $N(p^r) = \{p(i), \dots, p(j)\}$. It starts with the node $p(i)$, where an ancestor query is performed to see if $p(i)$ is a tree ancestor of q ; if it is, then the message is relayed down the tree from $p(i)$ to q . If $p(i)$ is not an ancestor of q , then we perform ancestor queries on the nodes with edges incident on $p(i)$ to determine if any of them is a tree ancestor of q ; if an ancestor is found, then the message is relayed to this node, and then down the tree. If an ancestor is not found, then the search continues in the next iteration with node $p(i + 1)$.

If the top node in $N(p^r)$, $p(j)$, is reached and an ancestor can still not be located, then the search continues with the parent of $p(j)$. As before, an ancestor query is performed on this node and then on the nodes that share an edge with this node. If an ancestor is not found, the search continues with the next parent. The search will necessarily detect an ancestor at some point, since it will eventually reach the root.

The aforementioned routing algorithm correctly finds the shortest path from u to v in the spanner. The proof (omitted) is by construction of the spanner and is similar to the proof of Theorem 3. We conclude this section with the following Theorem.

Theorem 7 *There exists a routing algorithm that routes on the spanner with the following properties: The stretch of the routing is $1 + \epsilon$, the size of the labels is $\lceil \log n \rceil$, and the total storage per point is $O(\log n / \epsilon^d)$.*

6 Fully dynamic spanner with poly-logarithmic update time

In this section we show how to turn the constant degree spanner (described in Section 3) into a fully dynamic spanner, where the amortized update time under point insertions and deletions is $O(\log^3 n)$. In the fully dynamic spanner the degree will not be constant, and Δ is set to be $\Theta(\log n \cdot \log c) = \Theta(\log n)$. We can assume that we start with an empty set of points; in the case that an initial set of points is given, then we can simply compute a spanner whose maximal degree is $\Theta(\log n)$ using the construction from Section 3. The algorithm works in phases. In each phase we have $\Omega(n)$ updates, and when a phase ends we construct a new spanner from scratch in time $O(n \log^2 n)$, using the technique described in Section 3.

In the dynamic case, the procedure for both deletions and the insertion of artificial covers may require a point to be given a new parent, and this results in a cut and link of that point's subtree. To this end, the underlying centroid path structure must be modified to allow links and cuts. The new structure is found in the full version. Insertions and deletions, links and cuts, and both containment and artificial cover searches now require $O(\log^2 n)$ time.

We begin by describing the algorithm for inserting a new point into the spanner, and then describe the algorithm for deleting a point from the spanner. We conclude by demonstrating that the amortized running time for both insertions and deletions is $O(\log^3 n)$.

Insert: When a new point p is inserted we find the deepest point q such that $q \in C^j$ and $|pq| \leq 2^{j+1}$. Using q we search for the point that covers p . The cost of this search is proportional to the number of edges that p will be assigned, that is $O(|N(p)|)$. As we maintain the underlying data structure of biased skip list and since the containment requirement is fulfilled (that is, if $q \in C^j$ and $|pq| \leq 2^{j+1}$, then for ancestor \hat{q} of q in level $i > j$ we have $|p\hat{q}| \leq 2^{j+1}$) the point q can be found in $O(\log^2 n)$ time. After p is inserted and $N(p)$ is computed, we check whether $|N(p)| > 9\Delta$. If so, we begin a degree reduction process for p . The cost of the process is charged from the artificial cover points that participate in it.

An edge that is added to the spanner as a result of the insertion of p will have as one of its endpoints the point p , or perhaps an artificial cover point z that replaced p in a degree reduction process of p . Let (p, q)

$((z, q))$ be such an edge. The cost of adding (p, q) ((z, q)) is charged to p (z). Now, such an edge may have a secondary effect; it might create a new node in $N(q)$, and this might cause $|N(q)|$ to exceed 9Δ . To limit this effect, nodes that are added in the top $2 \log c$ levels of $N(q)$ are not counted towards the limit on the size of $N(q)$, and only nodes that are strictly below level $h(q) - 2 \log c$ are counted. This implies that (p, q) must be in $E(p(i))$ where $i > h(p) - 2 \log c$ as otherwise the separation property would be violated. (A similar argument applies to (z, q)). A more careful look reveals two things, the first is that an edge (z, q) cannot touch q below level $h(q) - 2 \log c$, since this would cause q to have a conflict either with z or with the point that covers z . The second is that there cannot be two points q and q' such that both $|N(q)|$ and $|N(q')|$ become greater than 9Δ as a result of the insertion of p – this would result in a conflict between q and q' . Thus, a degree reduction is done to at most one point.

If the new point p has fewer than 9Δ nodes, then it remains untouched. However, it might be that a new edge (p, q) may cause to $N(q)$ to exceed the allowed bound. As we count only edges that are added below level $h(q) - 2 \log c$, the separation property ensures that at most one point may fall into this category. A degree reduction process is done to q . If q is already been used as an imaginary point elsewhere then after performing a degree reduction to q^r the old imaginary instance of q is replaced by another point simply by searching an artificial cover for the point q^r 's imaginary instance covers. The insertion of p is charged with this cost.

Delete: Let p be a point with two instances that is to be deleted. For every point that is covered by p we must find a point that covers it while still maintaining the desired properties of the hierarchy. We start with the real instance of p , and take the deepest node that corresponds to p^r and covers at least one other point q . Let $i = h(q)$. We insert q into level $h(q)$ and continue to insert q in subsequent higher levels until q is covered. (Note that to cover q with the new node may require a cut and link in the tree.) We then revisit level i and check if there are still uncovered points in this level. If there are, then we pick one of these points arbitrarily and repeat the same insertion process that was done to point q . We repeat this until there are no uncovered points left in i ; we can then proceed to the first level above i that contains an uncovered point. The process terminates when every point that was covered by p^r is covered. Since the point that covers p^r at level $h(p^r) + 1$ appears in many levels above $h(p^r)$, the process must terminate no later than level $h(p^r) + 2$.

The cost of the new edges added as a result of the deletion and subsequent insertions into higher levels are charged to the deletion. As p^r itself is artificially covered, an edge that is added or deleted during the process the node that corresponds to its other endpoint is below the top $2 \log c$ levels of that endpoint and such a change has no effect on the size of the node set. During the deletion process, however, there are points whose top level is increased, (for example the point q). For each such point, if its set of nodes exceeds the allowed bound, a degree reduction process is initiated. We deal with the deletion of p^{im} in a way similar to a point insertion. Let z be the point that is artificially covered by p^{im} ; then z is inserted into level $d(p^{im})$ and into each subsequent level until z is covered. The cost of this is the cost of a containment search, plus the cost of updating the set $N(z)$. If $|N(z)| > 9\Delta$ then a degree reduction process is initiated for z , and as before the cost is charged to the artificial cover points used in the process. If $|N(z)| \leq 9\Delta$, then the cost is charged to the deletion. As z is very close to p^{im} , the effect of such a change is limited and no further updates are required. We will further discuss these changes later in this section.

When the point to be deleted p possesses only a real instance, then the previously mentioned process for the deletion of a real instance commences. The main difference now is that the point that covers p will not necessarily covers the point that replaces p in p 's top level, and in fact this point may have to be inserted into much higher levels than $h(p) + 1$. This issue is handled in a manner similar to the insertion of a new point and to the deletion of an imaginary instance. The effect of such a deletion is symmetrical to the effect of the insertion of p : There might be one edge (p, q) where q is imaginary, $(p, q) \in E(q(i))$ and $i < h(q) - 2 \log c$. The deletion of (p, q) may reduce the size of $N(q)$ to less than 3Δ . In such a case we extend the point that q artificially covers to cover q 's range as well. If the size restriction is violated

then we undertake a degree reduction process for the point that q artificially covered. The deletion is then charged with an additional cost of $O(\log^3 n)$, which allows q to be used as an artificial cover in the future.

Degree reduction process: We now look more closely at the degree reduction process. Our main concern is that during the process an artificial cover that is forced into level i will violate the separation requirement. This is exactly the reason why we have two types of separations at level i and why Δ is set to $\Theta(\log n)$; we refer to a separation of 2^i as a *strong* separation and to a separation of 2^{i-1} as a *weak* separation. When a new point is inserted into the spanner, and thus into the hierarchy, the point is required to fulfill a strong separation. When a point is inserted into a level as an artificial cover it is only required to fulfill a weak separation.

In the next lemma we show that a weak separation can be violated during a degree reduction process only after $\Omega(n)$ updates and by that stage we have already reconstructed the spanner from scratch.

Lemma 8 *Let η be an arbitrary level in the hierarchy. Two points of C^η may violate the weak separation only after $\Omega(n)$ updates.*

Proof: It follows from Lemma 5 that the distance between the point on which we perform a degree reduction process and an artificial cover point that participates in the process starting from level η is $2^{\eta-\Delta}$. In our case $\Delta = \Theta(\log n)$. An update operation causes at most one change at a given level η . At first all the points are 2^η away from each other. During each update one point may move at most $2^\eta/\Omega(n)$. Thus, only after $\Omega(n)$ updates the weak separation may be violated. \square

Note that when an artificial cover is inserted into level η , edges that touched the original point may now touch the artificial cover in level $\eta + 1$ or $\eta - 1$, and not only in level η . However (in a manner similar to Lemma 8) such edges can touch a future artificial cover in level $\eta \pm 2$ only after $\Omega(n)$ updates. Thus, we associate such edges with the node that corresponds to the artificial cover point in level η , and no new nodes are added. Note also that a point that had been covered by a point in level η and is now covered by an artificial cover point in level $\eta + 1$ will be covered in level $\eta + 2$ only after $\Omega(n)$ updates. This enlarges the point's node set only by one and only once per phase; thus, we can ignore this enlargement.

When we undertake a degree reduction process for an imaginary instance of a point in the hierarchy that is not artificially covered, we leave the top 3Δ nodes while the nodes below are artificially covered. If the point is artificially covered by another point then the situation is a bit more complicated. Let x be the artificial cover of y and let y be the artificial cover of z . Assume that $|N(z)| > 9\Delta$. By the way an artificial cover is found we know that the first edge on the search path from z to its artificial cover is an edge that emanates from z at least $\Delta + 2 \log c$ levels below $h(z)$ and at least $\Delta + 2 \log c$ levels above $d(z)$. We would like to keep this situation also after the degree reduction of z . If this edge is located in the edge list of one of the 4Δ top nodes in $N(z)$ then we artificially cover z with a point z' starting from $\Delta + 2 \log c$ nodes below that edge and up to level $h(z)$. It follows that y is the artificial cover of z' and also as it stems from the above discussion on edges the first edge of the search path from z' to its artificial cover y is located in the right place. When this edge is located in a deeper node than the 4Δ top nodes of $N(z)$ we extend the range of y so it will replace part of the range of z . More specifically, we update $N(z)$ and $N(y)$ by moving the 3Δ top nodes of $N(z)$ to be the 3Δ bottom nodes of $N(y)$. The size of $N(z)$ is less than 9Δ but the size of $N(y)$ is too large and we do a degree reduction for y . We will leave in $N(y)$ only the 3Δ nodes that were transform from $N(z)$. Let y' be the artificial cover of y . It is inserted into the first level above the level of the top node among the 3Δ nodes that were transformed from z to y and it replaces y from that level till $h(y)$. Notice that the search path to x that was started at y at least $\Delta + 2 \log c$ levels below $h(y)$ and at least $\Delta + 2 \log c$ levels above $d(y)$ will start in the same place in y' .

We conclude the section by analyzing the cost of insertions and deletions. The cost incurred by a point in order it to be an artificial cover is $O(\log^3 n)$: An artificial cover replaces a point that covers other points in $O(\Delta)$ levels and this may generate $O(\Delta)$ link and cut operations each in a cost of $O(\log^2 n)$. The cost

of inserting a point p is comprised of three different parts. The first is the containment search cost which is $O(\log^2 n)$. The second is the cost of creating the set $N(p)$ which is $O(\gamma \cdot |N(p)|)$. The last is a credit of $O(\log^3 n)$ that p receives for future use as an artificial cover. The cost of constructing $N(P)$ might exceed $O(\log^3 n)$ if $|N(p)|$ very large. However in such a case a degree reduction process is performed for p , and the cost of the process, as well as the cost of creating $N(p)$, will be charged to the artificial cover points that participate in the process. The cost of deleting a point p is similar to the cost of the insertion. Removing p and finding new cover points for the points that were covered by p costs at most $O(\Delta \log^2 n)$, since $|N(p^r)| + |N(p^{im})| = O(\Delta)$. Degree reduction processes might be started both when p^r and p^{im} are deleted and when p has only one instance. As before the cost of the degree reduction process is charged to the artificial cover points that participate in the process. There is also a point whose imaginary instance may be deleted as it no longer has 3Δ nodes after the deletion of p . This point receives the extra credit of $O(\log^3 n)$ to allow it to be an artificial cover in the future. Removing the real part of a point may also generate link and cut operations in the underlying data structure whose cost is $O(\log^2 n)$. Thus the deletion cost is $O(\log^3 n)$. The spanner is constructed from scratch after $\Theta(n)$ updates.

The next theorem follows from the above discussion:

Theorem 9 *Let S be a set of points in a doubling dimension metric space of dimension d . A $(1 + \varepsilon)$ spanner of size $O(n/\varepsilon^d)$ of the points in S can be maintained in an amortized update time of $O(\log^3 n)$ for both an insertion of a point to S and a deletion of a point from S .*

References

- [1] I. Abraham, C. Gavoille, A. V. Goldberg, and D. Malkhi. Routing in networks with low doubling dimension. In *26th International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society Press, 2006.
- [2] S. Arya, D. M. Mount, and M. Smid. Dynamic algorithms for geometric spanners of small diameter: Randomized solutions. *Computational Geometry: Theory and Applications*, 13:91–107, 1999.
- [3] A. Bagchi, A. L. Buchsbaum, and M. T. Goodrich. Biased skip lists. *Algorithmica*, 42, 2005.
- [4] P. Bose, J. Gudmundsson, and P. Morin. Ordered theta graphs. *Computational Geometry: Theory and Applications*, 28:11–18, 2004.
- [5] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k -nearest-neighbors and n -body potential fields. *J. ACM*, 42:67–90, 1995.
- [6] T-H. Chan, A. Gupta, B.M. Maggs, and S. Zhou. On hierarchical routing in doubling metrics. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 2005.
- [7] R. Cole and L. Gottlieb. Searching dynamic point sets in spaces with bounded doubling dimension. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2006.
- [8] G. Das, G. Naraimhan, and J. Salowe. A new way to weigh malnourished Euclidean graphs. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 215–222, 22–24 1995.
- [9] J. Gao, L. Guibas, and A. Nguyen. Deformable spanners and applications. In *Annual ACM Symposium on Computational Geometry*, 2004.
- [10] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006.

- [11] Y. Hassin and D. Peleg. Sparse communication networks and efficient routing in the plane. *DIST-COMP: Distributed Computing*, 14, 2001.
- [12] Goran Konjevod, Andréa W. Richa, and Donglin Xia. Optimal scale-free compact routing schemes in networks of low doubling dimension. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 939–948, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.
- [13] R. Krauthgamer and J. Lee. Navigating nets: Simple algorithms for proximity search. In *Symposium on Discrete Algorithms*, 2004.
- [14] L. Roditty. Fully dynamic geometric spanners. In *To Appear in the 23rd Annual ACM Symposium on Computational Geometry*, 2007.
- [15] J. S. Salowe. Constructing multidimensional spanner graphs. *Int. J. Comput. Geometry Appl*, 1(2):99–107, 1991.
- [16] P. M. Vaidya. A sparse graph almost as good as the complete graph on points in K dimensions. *Discrete & Computational Geometry*, 6:369–381, 1991.

```

algorithm artificial-cover( $p$ )
while (1)
    for  $j \leftarrow \Delta + 2 \log c$  to  $2\Delta - 2 \log c$ 
        if  $\exists(p, z) \in E(p(i)) \wedge z$  is real
            return  $z$ 
     $p \leftarrow \text{next}(p)$ 

```

Figure 1: The search for an artificial cover

7 Appendix

7.1 Proof of Theorem 3, continued.

Size. Assume that the length of the edge (p, q) is the minimum length in H . Let $k = \lfloor \log |pq| \rfloor$. One of p or q are covered in level $k + 1$; without loss of generality, assume that p is the one covered in level $k + 1$. Combining Lemma 1 with the fact that any point is at least 2^k away from p , we get that the number of edges p has in level $k - j$ is at most $(1 + 2c/2^j)^d < (4c/2^j)^d$, where $0 \leq j \leq \log c$. Hence, the total number of edges incident to p is $\sum_0^{\log c} (4c/2^j)^d \leq 2(4c)^d$. If p has only one instance we remove it from H . If p has both a real instance and an imaginary instance then we remove just the edges that correspond to the real instance of p (which are obviously shorter than those that correspond to the imaginary instance and thus are treated first). We repeat on this process until the graph is empty. The total number of edges is $O(n/\varepsilon^d)$.

Construction time. The construction time of the initial spanner is $O(n \log n)$, so we only need to show that the total cost of performing degree reductions is $O(n \log n)$. The cost of reducing the degree of a given point is distributed among the points that take part in the degree reduction process; that is, those points that function as artificial covers. The cost associated with a specific artificial cover process is comprised of two parts. The first is the cost of finding the point to be used as the artificial cover point, and the second is the cost of inserting this point into the necessary levels of the hierarchy. As was mentioned before (and will be proven later), the cost of finding an artificial cover point is $O(\log n)$. The cost of inserting an artificial cover point into the hierarchy is $O(\Delta/\varepsilon^d \log n)$ (where the $\log n$ factor comes from the update of the biased skip lists). Thus, an artificial cover point will be charged with $O(\log n(1 + \Delta/\varepsilon^d))$ work. It follows that the total cost of executing degree reductions for all points is $O(n \log n)$.

7.2 The search for an artificial cover

We describe how to ensure that artificial cover is found in $O(\log n)$ time. The main question is how a next point is selected in the scenario in which the search at the current point yields only imaginary points. The basic idea is to identify two candidate points that are far apart, such that a search proceeding from one point cannot intersect a search proceeding from the other. We then choose one of these points. This allows us to bisect the point set of possible covers, and achieve logarithmic search time.

Let $p_{\Delta+4 \cdot \log c}$ and $p_{2\Delta-4 \log c}$ be two nodes that delimited a separation zone. Let $a = \text{lvl}(p_{2\Delta-4 \log c})$ and $b = \text{lvl}(p_{\Delta+4 \cdot \log c})$. Let (x, p) be an edge that satisfies, $c^{2^a-1} \leq |xp| \leq c^{2^a}$ (notice that if such an edge does not exist then we can change the separation zone by moving at most $\log c$ nodes above or below $p(a)$ until such an edge is found) and let (y, p) be an edge that satisfies, $|yp| \leq c^{2^b}$.

As a first step we show that the set of points visited by a search proceeding from x is disjoint from the set

of points visited by a search that proceeds from y . The distance between x and y is at least $c2^{a-1} - c2^b$. The total length of the search path from x to its last point is at most $c2^{a-\Delta+1}$. (This can be shown using similar arguments to those used in the proof of Lemma 5). Similarly, the total length of the search path from y to its last point is at most $c2^{b-\Delta+1}$. Thus, to prove the separation of the two search paths we have to show that $c2^{a-1} - c2^b > c2^{a-\Delta+1} + c2^{b-\Delta+1}$. By the choice of $a \gg b$ we can lower bound the left hand side by $c2^{a-1} - c2^b > c2^{a-2}$, and upperbound the right hand side by $c2^{a-\Delta+2} > c2^{a-\Delta+1} + c2^{b-\Delta+1}$. Thus, we have to prove that $c2^{a-2} > c2^{a-\Delta+2}$; but this is exactly the case when $\Delta > 4$.

It follows from the above discussion that when the search proceeds from x then only the set of points within a radius of $c2^{a-\Delta+1}$ from x are considered and when the search proceeds from y only the set of points within a radius of $c2^{b-\Delta+1}$ from y are considered. Moreover, the two sets are disjoint. Notice that by the choice of Δ for every constant value $c' \ll \Delta$ the points within a radius of $c2^{a-\Delta+c'}$ from x are separated from the points within a radius of $c2^{b-\Delta+c'}$ from y .

In the next lemma we provide the tool that is needed by the search in order to proceed with a smaller set of points.

Lemma 10 *Let $z^0, z^1, \dots, z^i, \dots, z^\ell$ be a sequence of points that were visited by a search initiated from z^0 . Let z^i be an arbitrary point in the sequence and let $t = \text{lvl}((z^i, z^{i+1}))$. Let $q \in S$ be a point within a radius of $c2^{t+1}$ from z^i that participates in an arbitrary subtree whose root is in level $t+2$. There is an edge from the root of that subtree to z^i in level $t+2$.*

Proof: We know that $|z^i q| \leq c2^{t+1}$. The distance from q to its ancestor in level $t+2$ is at most 2^{t+3} . Thus, the distance from z^i to the ancestor of q in level $t+2$ is at most $c2^{t+1} + 2^{t+3}$. By the choice of c it is less than $c2^{t+2}$, thus, there is an edge between z^i to the ancestor of q in level $t+2$. \square

Recall that the hierarchy defines a tree. Hence, the tree structure can be used to figure out how many points are in a given subtree. Based on the above lemma we can make the choice between x and y as follows. We compare the total number of points that are in subtrees rooted at points that have an edge with x in level $a - \Delta + 2$ to the total number of points that are in subtrees rooted at points that have an edge with y in level $b - \Delta + 2$. Computing these two numbers is done in time proportional to the number of edges x and y have in these levels. The points that are counted with x are at most $c2^{a-\Delta+2} + 2^{a-\Delta+3} \leq c2^{a-\Delta+3}$ away from x and the points that are counted with y are at most $c2^{b-\Delta+2} + 2^{b-\Delta+3} \leq c2^{b-\Delta+3}$ away from y . A point that is counted with x cannot be counted with y , since the set of point within a radius of $c2^{a-\Delta+3}$ from x are separated from the set of point within a radius of $c2^{b-\Delta+3}$ from y . Note also that every point within a radius of $c2^{a-\Delta+1}$ from x is counted and every point within a radius of $c2^{b-\Delta+1}$ from y is counted. The choice between x and y is simple. We go to the one that has fewer points. This guaranties a $O(\log n)$ steps in the search as in every step we proceed with a set of points that is at most half of the size of the previous set of points.

7.3 Centroid path decomposition

Here we discuss how to maintain the underlying search structure under links and cuts to the tree $T(\mathcal{C})$. We modify the structure so that insertions and deletions, links and cuts require $\Theta(\log^2 n)$ time. Finding the number of descendants of a node can be done in $O(\log n)$ time. The modifications will cause the cost of executing containment and artificial cover searches to grow to $\Theta(\log^2 n)$.

The underlying search structure is a centroid path decomposition of the tree $T(\mathcal{C})$. For each centroid path, we store its nodes in a biased skip list of [3] (as opposed to using the modified biased skip lists of [7]). As before, the weight of a node is equal to 1 plus the number of descendants in its off-path subtrees. Given

$T(\mathcal{C})$, it is possible to calculate its centroid path decomposition as well as the weight of each node in $O(n)$ time. Further, the total cost of building biased skip lists for all centroid paths from scratch is $O(n \log n)$.

Recall that the cost of reweighting a node in the skip list is $O(\log n)$. Further, the cost of linking two skip lists or breaking a single skip list – and hence the cost of fusing two tree paths, or breaking a single tree path – also is $O(\log n)$.

We describe the algorithm for links and cuts, of which insertions and deletions are simply special cases. Suppose a subtree rooted at node u is cut from its parent v , or a new subtree is added to v , meaning that its root u is added as a child of v . (We assume for the newly added subtree that its centroid path decomposition has already been determined, and each centroid path has its nodes stored in biased skip lists.) This subtraction or addition affects the route from v to the root, causing at most $O(\log n)$ changes to the centroid path segments this route traverses. In the case of an addition, the $O(\log n)$ changes occur at the top of each centroid path segment, since the top node may now be the heaviest child of its parent. In the case of a subtraction, the lowest centroid path segment may now be broken into $O(\log n)$ pieces. (Finding where the path is broken is not hard, and is omitted here.) Each such change may involve some combination of splitting a centroid path, joining two centroid paths, or adjusting weight of a constant number of nodes. It follows that each change to the centroid path decomposition requires a constant number of operations of the biased skip list, each operation at a cost of $O(\log n)$. All updates to the centroid path may be therefore be done in $O(\log^2 n)$ time.

We now discuss the costs of updates for $T(\mathcal{C})$: By the previous analysis, inserting or removing a node, or linking a node to a new parent, all require $O(\log^2 n)$ work. Inserting into the tree a sequence of m nodes, where each node of the sequence is a child of the previous node, can be done in time $O(\log^2 n + m \log m)$ time. First, a biased skip list is build for these nodes at a cost of $O(m \log m)$. Then the biased skip list is linked into $T(\mathcal{C})$ at a cost of $O(\log^2 n)$. In this paper, we are particularly concerned with the case where $m = O(\log n)$; the cost of inserting such a sequence is $O(\log^2 n)$.

A final outstanding issue that needs to be addressed is implementing the procedure that, given a node u , determines in which centroid path it lies. This is a necessary subroutine for the containment search; the containment search calls this subroutine $\Theta(\log n)$ times. Since each biased skip list is associated with a single centroid path, it suffices for u to discover which biased skip list contains it. This can be done by climbing up to the top level of the skip list, and takes $\Theta(\log n)$ time. It follows that a containment search requires $\Theta(\log^2 n)$ work. Similarly, the number of descendants of a node u can be determined by summing the weights of all nodes to the right of u in the skip list; this requires a single descent in the skip list, which can be done with $O(\log n)$ work. Since this is a subroutine for the artificial cover search, the search now requires $\Theta(\log^2 n)$ time.