

# An optimal dynamic spanner for doubling metric spaces

*Lee-Ad Gottlieb* \*

*Liam Roditty* †

June 23, 2008

## Abstract

For a set  $S$  of points in a metric space, a  $t$ -spanner is a 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 actual distance between the points. In this paper, we consider points residing in a metric space of doubling dimension  $\lambda$ , and show how to construct a dynamic  $(1 + \varepsilon)$ -spanner with constant degree and  $O(\log n)$  update time (when  $\lambda$  and  $\varepsilon$  are taken as constants). Our update time is optimal up to a constant.

---

\*Courant Institute, New York University, New York NY 10012. Email: [adi@cs.nyu.edu](mailto:adi@cs.nyu.edu). Author's work supported in part by NSF grant IIS 0414763.

†Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel. Email: [liam.roditty@weizmann.ac.il](mailto:liam.roditty@weizmann.ac.il).

# 1 Introduction

A graph  $H$  is a  $t$ -spanner of  $G$  if  $\delta_H(u, v) \leq t\delta_G(u, v)$ , where  $\delta_G(u, v)$  denotes the shortest path distance between  $u$  and  $v$  in  $G$ , and  $\delta_H(u, v)$  denotes the shortest path distance between  $u$  and  $v$  in  $H$ . A spanner can also be defined for a set of points residing in Euclidean space: Let  $S$  be a set of points in  $\mathbb{R}^d$ . The graph  $G$  is a complete graph whose vertices are the points of  $S$ , and the weight of every edge is the distance between its endpoints. A geometric  $t$ -spanner is then constructed for the graph  $G$ .

Geometric spanners have received a fair amount of attention in the past couple of decades. Various papers have dealt with the construction of geometric spanners with specific properties, such as linear number of edges, small weight (the weight of a spanner is the sum of the weights of its edges), small hop diameter and low degree. For points residing in low-dimensional Euclidean space, Vaidya [16], Salowe [14], Callahan and Kosaraju [5] and Soares [15] showed how to compute a geometric  $(1 + \varepsilon)$ -spanner with  $O(n/\varepsilon^d)$  edges in  $O(n \log n)$  time. It is also possible to build a spanner of constant degree in this time [7]. In the dynamic setting, where the problem is to *explicitly* maintain a set of edges that constitute a spanner of the point set, Arya *et al.* [3] obtained poly-logarithmic update time in the restricted model in which updates were assumed to be random: A point to be deleted is assumed to be selected at random from  $S$ , and a point to be inserted is assumed to be a random point of the new point set. Bose *et al.* [4] gave a semi-dynamic algorithm that supports insertions in poly-logarithmic time. Gao *et al.* [8] considered both dynamic and kinetic spanners (a kinetic spanner supports movement of the points), and gave a spanner with update time  $O(\log \alpha)$ , where  $\alpha$  is the *aspect ratio* of the set, the ratio between the largest and smallest interpoint distances of the set. This result is of interest when  $\alpha$  is small, which is often the case.

This paper is interested in the question of dynamic spanners for points that reside in a metric space with low *doubling dimension*. Let the space within radius  $r$  of a point be called the ball centered at that point. A point set  $X$  has doubling dimension  $\lambda$  if all points of  $X$  that are covered by a ball of radius  $r$  can be covered by  $2^\lambda$  balls of radius  $\frac{r}{2}$ . While a low Euclidean dimension implies a low doubling dimension (Euclidean metrics of dimension  $d$  have doubling dimension  $\Theta(d)$  [10]), doubling dimension is more general than Euclidean dimension, so all results for low doubling dimension apply to low Euclidean dimension as well. (Throughout this paper, we will take  $\lambda$  to be a constant, which hides multiplicative factors of  $2^{O(\lambda)}$ .) In this setting Har-Peled and Mendel [11] showed how to construct a static constant degree spanner in  $O(n \log n)$  time. Roditty [13] gave a dynamic spanner that supports insertions in  $O(\log n)$  amortized time and deletions in  $\tilde{O}(n^{1/3})$  amortized time (where the notation  $\tilde{O}$  is used to hide logarithmic factors). Very recently, these authors [9] gave a dynamic spanner that supports insertions in  $O(\log^2 n)$  amortized time and deletions in  $O(\log^3 n)$  amortized time.

In this paper, we improve on previous results by providing a  $(1 + \varepsilon)$ -spanner of constant degree that supports updates in  $O(\log n)$  worst case time. For insertions,  $O(\log n)$  is in fact optimal, since the task of inserting a point into a  $(1 + \varepsilon)$ -spanner subsumes within it the task of discovering a  $(1 + \varepsilon)$ -nearest neighbor of the new point. Since approximate nearest neighbor search is known to require logarithmic time [2] in the algebraic decision tree model, insertions into a spanner must require logarithmic time as well. We make no claims on the optimality of deletions.

A further application of our spanner is the maintenance of the closest pair of points in the set  $S$ . Note that in a  $(2 - \varepsilon)$ -spanner ( $\varepsilon > 0$ ), the pair (or pairs) of closest points must have an edge between them, or else their spanner stretch would be greater than  $2 - \varepsilon$ . By storing the edges in a heap based on weight, we can always return the closest pair in  $O(1)$  time.

**Comparison to previous work.** In [9] we showed how to obtain a spanner while building on the hierarchical partition introduced in [8] and [12]. These partitions had individual points appearing in as many as  $O(\log \alpha)$  levels, and so in order to avoid a dependence on  $\log \alpha$  in the update time of the hierarchy and the spanner, we introduced a complex process of point replacements that limited the number of times a

single point appeared in the hierarchy. This gave us amortized  $O(\log^3 n)$  update time, with degree  $\Theta(\log n)$ .

In this paper, we show that a much stronger result is possible if one is willing to make an additional assumption on the point set: We assume that even after a point is deleted from  $S$ , we can still query the distance between the deleted point and a point still in  $S$ , or the distance between two deleted points. Such an assumption is a rather common one and innocuous enough – it is clearly true of points in a well defined space such as the Euclidean space – but one could devise instances in which it does not hold, such as when the points represent wireless users who are deleted when they log off the network.

Making this assumption is necessarily to allow us to use the hierarchy and nearest neighbor structure of [6], where such an assumption is made. The advantage in using this hierarchy is that it can be updated in  $O(\log n)$  time as opposed to  $O(\log \alpha)$  time, eliminating the central hurdle we faced in building upon the hierarchy of [8] and [12]. However, it introduces a quite different, yet similarly difficult problem: This hierarchy contains within it points that have been deleted from the point set, and such points may not appear in the spanner. Using such a hierarchy to create a spanner is the central challenge addressed in this paper. We demonstrate a successful solution to this problem, resulting in a spanner with  $O(\log n)$  update time and constant degree.

**Well separated pairs decompositions.** We conclude the introduction with a comment concerning well separated pairs decompositions (WSPDs) and their relationship to spanners. Some definitions are necessary here. The diameter of a point set  $A$  ( $\text{diam}(A)$ ) is the largest inter-point distance in  $A$ . Let the distance between sets  $A$  and  $B$  be the minimum over all  $d(a, b)$ ,  $a \in A$  and  $b \in B$  (where  $d(\cdot, \cdot)$  is the distance function). Let  $A \otimes B = \{\{x, y\} \mid x \in A, y \in B\}$ .  $A$  and  $B$  are  $\frac{1}{\epsilon}$ -separated if  $\max(\text{diam}(A), \text{diam}(B)) \leq \epsilon \cdot d(A, B)$ . A WSPD of  $S$  with parameter  $\frac{1}{\epsilon}$  is a set of pairs  $\mathcal{W} = \{(A_1, B_1), \dots, (A_s, B_s)\}$  such that

- (i)  $A_i, B_i \subset S$  for every  $i$ .
- (ii)  $A_i \cap B_i = \emptyset$  for every  $i$ .
- (iii)  $\cup_{i=1}^s A_i \otimes B_i = S \otimes S$ .
- (iv)  $A_i$  and  $B_i$  are  $\frac{1}{\epsilon}$ -separated.

The data structure of Cole and Gottlieb [6] dynamically maintains a point set in low doubling dimension such that at any time a WSPD may be extracted with  $O(n)$  further work. A WSPD itself implies a spanner, so a spanner of the point set may also be extracted with  $O(n)$  work. (Such a spanner will not necessarily be of constant degree.) It follows that the structure of [6] implicitly implies a WSPD and a spanner, but this is of course not the same as the more difficult problem of dynamically maintaining an *explicit* set of edges for a spanner of the point set.

The rest of this paper is organized as follows. In the next section we describe some data structure tools that are essential for our algorithm. In Section 3 we review previous work on approximate nearest neighbor and relate it to spanner construction. In Section 4 we present our new dynamic spanner.

## 2 Preliminaries

**Colored ancestor and descendant queries.** Consider a tree with some nodes that are *colored*. A *colored ancestor query*  $\langle v, \mathbf{c} \rangle$  on node  $v$  and color  $\mathbf{c}$  asks for the lowest ancestor of  $v$  which is colored  $\mathbf{c}$ . (Elsewhere this is called a *marked ancestor query* [1].) Similarly, a *colored descendant query*  $\langle v, \mathbf{c} \rangle$  asks for the highest descendant of  $v$  which is colored  $\mathbf{c}$ . Multiple colored ancestors (or descendants) can be found by executing a series of queries, each subsequent query on the node returned by the previous query.

Using standard dynamic tree structures, colored ancestor and descendant queries can be supported (for a constant number of colors) in  $O(\log n)$  query and update time, under point insertions and deletions. For example, assume a centroid path decomposition of the tree is maintained dynamically, and for each centroid path the  $\mathbf{c}$ -colored nodes on that path are stored in a balanced tree – this all can be done in  $O(\log n)$  update time. Then an ancestor query  $\langle v, \mathbf{c} \rangle$  could be accomplished by first discovering the centroid path on which  $v$  is found, and then finding the highest  $\mathbf{c}$ -colored node on that path, all in  $O(1)$  time. If this highest  $\mathbf{c}$ -colored node is above  $v$ , then the binary tree is searched for the lowest  $\mathbf{c}$ -colored ancestor of  $v$  in  $O(\log n)$  time. If the highest  $\mathbf{c}$ -colored node is below  $v$  (or there are no  $\mathbf{c}$ -colored nodes on the path), then the search ascends to the first ancestral centroid path that has  $\mathbf{c}$ -colored nodes ( $O(\log n)$  time), and returns the lowest  $\mathbf{c}$ -colored node.

### 3 Hierarchical partitions and spanners

In this section we describe a hierarchical partition of points that inhabit a doubling dimension.

A subset of points  $X \subseteq Y$  is an  $\epsilon$ -discrete center set ( $\epsilon$ -net in the terminology of [12]) of  $Y$  if it satisfies the following invariants:

- (i) For every  $x, y \in X$ ,  $d(x, y) \geq \epsilon$ .
- (ii) Every point  $y \in Y$  is within distance  $\epsilon$  of some point  $x \in X$ .

We say that  $x$  covers  $y$  if  $x \in X$ ,  $y \in Y$  and  $d(x, y) \leq \epsilon$ . The previous conditions require that the points of  $X$  be spaced out, yet nevertheless cover all points of  $Y$ .

Let  $S$  be a set of points with doubling constant  $\lambda$ , and let  $\alpha$  be the aspect ratio of  $S$ , the ratio between the largest and smallest inter-point distance in  $S$ . (For ease of presentation, we assume that the minimum inter-point distance in  $S$  is 1.) The hierarchical partition is a hierarchy of discrete center sets. The bottom level of the hierarchy is the set  $Y_{2^0=1} = S$ , and the top level is the set  $Y_{2^{\log \alpha}}$  that contains only a single point. Each intermediate level  $i > 0$  of the hierarchy is represented by a set  $Y_{2^i}$  ( $i > 0$ ) which is a  $2^i$ -discrete center set of the set  $Y_{2^{i-1}}$ .

**Extracting a spanner.** The hierarchical partition can be used as a backbone for a geometric spanner. To this end a few more definitions are required. A point  $x \in Y_{2^i}$  is a parent of  $y \in Y_{2^{i-1}}$  if  $x$  covers  $y$ . If more than one point covers  $y$  then an arbitrary one of these points is chosen to be the parent of  $y$ . A point  $x$  is an ancestor of  $y$  if there exists a series of points  $\langle x, \dots, y \rangle$  such that each point in the series is a parent of the subsequent one. For a spanner  $H$ ,  $d_H(x, y)$  is the spanner distance between  $x$  and  $y$ .

We use the hierarchical partition to decide which edges are included in the spanner. There are three types of edges. The first type consists of intra-level edges, that is, edges that connect points in the same level when the distance between the points is below some threshold. Specifically, in level  $i$  we add an edge between any two points that are within distance  $5 \cdot 2^i$ . The second type includes edges that connect parents and children. The third type includes *refinement* edges. A 1-level refinement edge connects the children of a pair of points that have an intra-level edge or the same parent. In general, a  $j$ -level refinement edge connects a pair of points in level  $i$  if there is an intra-level edge between a pair of their ancestors in one of the levels  $(i, i + j]$ .

We now show that a simple spanner that is created on top of the hierarchical partition has a constant stretch factor. Let  $H$  be a spanner that contains intra-level edges, parent-child edges and only 1-level refinement edges.

**Lemma 1**  $d_H(x, y) \leq 9d(x, y)$  for every pair of points  $x, y \in S$ .

**Proof:** If  $d(x, y) \leq 5$  then  $x$  and  $y$  have an intra-level edge in level  $i = 0$ , and we are done. Otherwise, let  $x', y' \in Y_{2^i}$  be the first ancestors of  $x$  and  $y$  in the hierarchy which are connected with a 1-level refinement edge. It is easy to see that the ancestral relationship implies that  $d_H(x, x')$  and  $d_H(y, y')$  (and also  $d(x, x')$  and  $d(y, y')$ ) are less than  $2^i \sum_{j=0}^{\infty} \frac{1}{2^j} = 2 \cdot 2^i$ .

Now,  $d(x, x') + d(x, y) + d(y, y') \geq d(x', y')$ , and thus  $d(x, y) \geq d(x', y') - d(x, x') - d(y, y') > d(x', y') - 2 \cdot 2^i - 2 \cdot 2^i = d(x', y') - 4 \cdot 2^i$  while  $d_H(x, y) \leq d_H(x', y') + d_H(x', x) + d_H(y', y) < d(x', y') + 2 \cdot 2^i + 2 \cdot 2^i = d(x', y') + 4 \cdot 2^i$ . Hence, the stretch of the spanner is  $\frac{d(x', y') + 4 \cdot 2^i}{d(x', y') - 4 \cdot 2^i} = 1 + \frac{8 \cdot 2^i}{d(x', y') - 4 \cdot 2^i}$ . This term is maximized when  $d(x', y')$  is minimized. Since  $d(x', y') \geq 5 \cdot 2^i$ , the spanner has a stretch less than  $1 + \frac{8 \cdot 2^i}{5 \cdot 2^i - 4 \cdot 2^i} = 9$ . □

It follows easily from the proof of the lemma that if refinement edges of higher levels are added to the spanner then the stretch of the spanner is reduced. Suppose that  $j$ -level refinement edges are added, and let  $x'' (y'')$  be a descendant of  $x' (y')$  and ancestor of  $x (y)$  in level  $i - j + 1$ . Note that there is a refinement edge in spanner connecting  $x''$  and  $y''$ . We repeat the previous argument with  $x'$  and  $y'$  replaced respectively by  $x''$  and  $y''$ , and conclude that the stretch of the spanner is less than  $\frac{d(x'', y'') + 4 \cdot 2^{i-j+1}}{d(x'', y'') - 4 \cdot 2^{i-j+1}} = 1 + \frac{8 \cdot 2^{i-j+1}}{d(x'', y'') - 4 \cdot 2^{i-j+1}}$ . This term is maximized when  $d(x'', y'')$  is minimized. Since  $d(x'', y'') \geq d(x', y') - d(x'', x') - d(y'', y') > d(x', y') - 4 \cdot 2^i \geq 5 \cdot 2^i - 4 \cdot 2^i = 2^i$ , the stretch of the spanner is less than  $1 + \frac{8 \cdot 2^{i-j+1}}{2^i - 4 \cdot 2^{i-j+1}} = 1 + \frac{2}{2^{j-3} - 1}$ .

**Modified hierarchical partition.** [12] showed how to dynamically maintain a hierarchical partition in  $O(\log \alpha)$  update time. Note however that when  $\alpha = n^{\omega(1)}$ , then the update time is  $\omega(\log n)$ . Improving on this, Cole and Gottlieb [6] modified the hierarchical partition to support updates in  $O(\log n)$  time. Achieving this update time requires the use of auxiliary data structures and is intricate, but for our purposes we need only to highlight the changes to the hierarchical partition.

The new hierarchical partition is defined as follows. Similar to what was done before, the bottom level is the set  $Y_{5^0=1}$  that contains all the points, and the top level is the set  $Y_{5^{\log \alpha}}$  that contains only a single point. Each intermediate level  $i > 0$  of the hierarchy is represented by a set  $Y_{5^i}$  ( $i > 0$ ) which is a 5<sup>i</sup>-discrete center set of the set  $Y_{5^{i-1}}$ , where the discrete center set definition is slightly altered to satisfy the following invariants:

- (i)  $d(x, y) \geq 5^{i-1}$  for every  $x, y \in Y_{5^i}$ .
- (ii) Every point of  $Y_{5^{i-1}}$  is within distance  $3 \cdot 5^{i-1}$  of some  $x \in Y_{5^i}$ .

Notice that the second invariant, applied recursively, implies that every point of  $Y_{5^j}$ , where  $j < i$ , is within distance of  $4 \cdot 5^{i-1} - 5^j$  from some point of  $Y_{5^i}$ . We call this the *close-containment* property. (The choice of the constant 5 to define the radius of each level of the hierarchy is due to considerations discussed in [6].)

To achieve linear space, Cole and Gottlieb [6] showed how to represent the hierarchy implicitly. For simplicity, we will view the hierarchy as if it were maintained explicitly; this will have no asymptotic effect on the runtime of our algorithm. (For those familiar with the terminology of [6], we remove all jumps by continuously splitting the jumps until none remain.) The rules governing updates to the hierarchy are rather involved, however for our purposes it is sufficient to notice the following. An insertion of a point into the set  $S$  entails adding the new point to  $Y_1$ , and possibly adding some other points to other levels, however, only a constant number of such additions will be handled explicitly.

Moreover, when a point is deleted from  $S$  no change occurs to the hierarchy – this means that Steiner points (i.e., points that no longer belong to  $S$ ) are present in the structure. While [6] showed that the presence of these Steiner points does not interfere in ANN search, these points are not acceptable in the spanner. Our challenge then is to use a hierarchy that contains Steiner points as a tool for the extraction

of a spanner that contains no Steiner points. (As an aside, we note that the data structure in [6] must be rebuilt in the background after an appropriate number of deletions, in order to keep its size commensurate with the current size of the point set. We will therefore need to build, in the background, a spanner for the new structure. This rebuilding is a standard technique and has no asymptotic effect on the run time or the degree.)

From the modified hierarchy a spanning tree  $T$  is extracted. The spanning tree directly corresponds to the hierarchy: Its nodes are arranged in levels, and it has one node for each point in the hierarchy. Two nodes in the tree are connected if and only if their corresponding points are a parent-child pair. The machinery of [6] maintains  $T$  dynamically. Using an auxiliary structure, the spanning tree can support marked ancestor and descendant queries. Note that the spanning tree may contain nodes that correspond to Steiner points.

We conclude this section by introducing some terminology. A point  $x \in Y_{5^j}$  contains a point  $y \in Y_{5^i}$  if  $j > i$  and  $d(x, y) \leq 5^j$ . Similarly, a node  $v$  contains a node  $w$  if  $v$ 's point contains  $w$ 's point. (Note that  $x$  need not be an ancestor of  $y$ , nor  $v$  an ancestor of  $w$ , for this relationship to hold.) Points  $x$  and  $y$  are friends if they are found in the same level  $Y_{5^i}$  and  $d(x, y) \leq 2 \cdot 5^i$ . Nodes  $v$  and  $w$  are friends if their corresponding points are friends.

## 4 Spanner Construction

For points inhabiting a low doubling dimension, we show how to maintain a constant degree  $(1 + \epsilon)$ -spanner of a set of points under insertions and deletions of points in  $O(\log n)$  update time. We build upon the net structure of [6]; specifically, we assume that we have access to an implicit representation of the dynamic spanning tree  $T$  of the hierarchy.

**Modifying the spanning tree - Step 1.** We first prune  $T$  in a straightforward manner and create a new spanning tree  $T_1$ .

Let *real* nodes (or leaves) in  $T$  be nodes that correspond to non-deleted points, and Steiner nodes (or leaves) be those nodes that correspond to deleted points. We create  $T_1$  from  $T$  thus: First, we remove from  $T$  all Steiner leaves, as well as all nodes that have no real leaf descendants. Then we compress all single-child paths; that is, if there is a chain of nodes  $x_1, x_2, \dots, x_k$  where  $x_i$  is the only child of  $x_{i-1}$  for every  $i \in [2, k]$ , then we replace the chain with a single node. It follows that all remaining internal nodes have multiple children. The resulting tree is  $T_1$ . (For a node  $v \in T$  that survives in  $T_1$ , we may refer both to  $v \in T$  and to  $v \in T_1$ .)

Note that a single insertion or deletion to the point set results in the insertion or deletion of a single leaf node in  $T_1$ , and possibly a single internal node. The single internal node will be added if the insertion of the leaf causes a contracted single-child chain to be expanded.

Note that  $T_1$  contains nodes that correspond to Steiner points in  $S$ . Further, there are points in  $S$  that correspond to more than one node in  $T_1$ , as a single point may be in many levels of the hierarchy. The solution for both of these problems is the same and will be presented in the next section. To allow a clean representation of the ideas that will be presented in this section we define the set  $S'$ . Let  $S' \supseteq S$  contains all the points that correspond to nodes of  $T_1$ . In particular,  $S'$  may contain points that were previously deleted from  $S$  (the aforementioned Steiner points). Also,  $S'$  may contain multiple distinct points that all correspond to a single point in  $S$ : This reflects a scenario in which a single point in  $S$  corresponds to both a leaf node and also to multiple internal nodes in  $T_1$ . In such a case  $S'$  will have new points corresponding to the internal nodes. (The original point is also in  $S'$  and corresponds to the leaf node.) Hence, we obtain a unique point corresponding to each node of  $T_1$ .

In Section 4.1 we show how to construct a spanner  $P' = G(S', E^{SP'})$  for  $S'$  and maintain  $G(S', E^{SP'})$  dynamically. In Section 4.2 we will show that it is very simple to modify  $G(S', E^{SP'})$  to create a spanner

$P = G(S, E^{\text{SP}})$  for  $S$ . This modification involves swapping points of  $S' - S$  out of  $G(S', E^{\text{SP}'})$  and replacing them with points of  $S$ .

Next, we prove an important structural lemma for the hierarchy. This lemma will be used later on in the construction of the spanner.

**Lemma 2** *Let  $u \in Y_{5^k}$  and  $v \in Y_{5^\ell}$ , where  $k \ll \ell$ , be a child-parent pair in  $T_1$ . Let  $c > 1$  be a constant. If there is no node in  $T_1$  between levels  $k$  and  $\ell$  that contains  $u$  within its radius, then there exist only a constant number of nodes between levels  $k$  and  $\ell$  that have  $u$  within a distant of  $c$  times their radius.*

**Proof.** Let  $w \in Y_{5^m}$ , where  $k < m < \ell$ , be the highest node that has  $u$  within a factor of  $c$  times its radius. Since no node at these levels has  $u$  within its radius,  $d(u, w) \geq 5^m$ . By the close-containment property, the distance from  $w$  to any of its descendant is less than  $4 \cdot 5^{m-1}$ , so the distance from  $u$  to any descendant of  $w$  is at least  $5^m - 4 \cdot 5^{m-1} = 5^{m-1}$ .

Now every node of level  $m - 1 - \log_5 c$  or lower has a radius of  $5^{m-1}/c$  or less, so it follows that descendants of  $w$  below level  $m - 1 - \log_5 c$  cannot have  $u$  within a distance of  $c$  times their radius. Hence, only a constant number of descendants of  $w$  can have  $u$  within  $c$  times their radius and these are exactly the nodes whose level is between  $m - \log_5 c$  and  $m$ .

We now apply this argument iteratively: The next node on which we apply this argument is the second highest node that has  $u$  within a factor of  $c$  times its radius, but we consider this node only if it is not a descendent of  $w$ . We iterate until there are no nodes left to which the argument can be applied – that is, until there are no nodes in the relevant levels that have  $u$  within a factor of  $c$  times their radius, and were not considered or had an ancestor considered. We now show that only a constant number of such iterations are performed. It follows from packing considerations (a property of low doubling dimension) that there are only a constant number of nodes in level  $m$  to which the argument can be applied. For lower levels than  $m$  to which the argument is applied, note that any two such nodes must have a different ancestor in level  $m$  in the spanning tree  $T$ . This follows from the fact that in each iteration the highest available node was chosen. Again from packing considerations, only a constant number of different ancestors can exist in level  $m$  of  $T$ .  $\square$

#### 4.1 Construction of a spanner for the new set $S'$

We use  $T_1$  to create a spanner  $G(S', E^{\text{SP}'})$ . The set of edges  $E^{\text{SP}'}$  will be formed by applying a few rules to the tree  $T_1$ .

Before we begin the construction, recall that this spanner is for the new set  $S'$ , and may contain points that are no longer part of  $S$ . Further recall that a point of  $S$  that corresponds to more than one node of  $T_1$  is duplicated in  $S'$ , based on the number of nodes it corresponds to in  $T_1$ . Thus, every node in  $T_1$  is associated with a unique point in  $S'$ . Later in Section 4.2 we will show how to modify the spanner to contain only points of  $S$ , without the duplications and Steiner points found in  $S'$ . In what follows, when we say that we add an edge between two nodes, we mean that we add an edge between their corresponding points.

The first step in the spanner construction is to add to  $E^{\text{SP}'}$  a single edge for each parent-child pair of nodes in  $T_1$ .

The second type of edges to be added are replacement edges. More specifically, let  $w$  be a node at level  $k$  with a tree parent  $v$  at level  $\ell$ . To mimic the original extraction of a spanner from the explicit hierarchy (as was described in Section 3), we would need to add an edge from the point that corresponds to  $w$  to each point that corresponds to a nearby node in the intermediate levels between  $k$  and  $m$ . However, we cannot afford to do this, since the degree of a point would be too high, and also fast dynamic maintenance

would be impossible. To overcome this problem, we search for the lowest node  $w' \in Y_{5^m}$  ( $k < m \leq \ell$ ) which contains  $w$  within its radius. (In the degenerate case that no such node exists,  $v$  will function in the place of  $w'$ .) A description of how a search for  $w'$  is executed will be presented later. We add the edge  $\langle w, w' \rangle$  to the spanner, and  $w'$  will now serve as the *replacement* of  $w$  for levels  $m$  to  $\ell$ , that is, no edge will be added to the point that corresponds to  $w$  from nearby points that correspond to a node from level  $i$ , where  $m < i \leq \ell$ .

Now that we have added an edge from  $w \in Y_{5^k}$  to a node in a higher level, we turn to adding edges from  $w$  to nearby nodes in its own level. Let  $R$  be the set of nodes in  $Y_{5^k}$  which are within distance  $27 \cdot 5^k$  of  $w$ . (The choice of this constant will become clear in the proof of Theorem 4.) Connect  $w$  to every node of  $R$  that survives in  $T_1$ . For each node of  $R$  that does not survive in  $T_1$ , connect  $w$  instead to that node's highest surviving descendant. (However  $w$  will only connect to this descendant if the descendant does not have a replacement in level  $Y_{5^k}$  or lower.) Finding a highest surviving descendant can be accomplished through a colored descendant query – it suffices to stipulate that every node in  $T$  that survives in  $T_1$  is colored **r**(ed); we then execute a colored descendant query on the relevant node of  $R$  and **r**.

Finally, we have the refinement edges that can be added multiple times in order to produce a better spanner: For two connected nodes in  $T_1$  (for any type of connection they may have), a 1-level refinement edge connects the children of these nodes to each other and to the original nodes. Similarly, a 1-level refinement edge connects all the children of a single node. Subsequent levels of refinement edges connect two non-connected nodes of  $T_1$  if their parents in  $T_1$  are connected. For the most basic spanner we assume a single refinement step.

To summarize, there are four types of edges:

- (i) parent-child edges that reflect the parent-child relationship in  $T_1$ .
- (ii) edges to replacement nodes.
- (iii) intra-level edges between nearby nodes in the same level.
- (iv) refinement edges.

It remains only to describe how to execute a search for  $w'$ , the replacement of  $w$ . Recall that we stipulated that all nodes of  $T$  that survive in  $T_1$  be colored **r**(ed) in  $T$ . Further, let all nodes in  $T$  that are friends of the red nodes be colored **g**(reen). Note that since  $w'$  contains  $w$ , the ancestor of  $w$  in the same level of  $w'$  is necessarily a friend of  $w'$ . We execute a series of ancestor queries on **g** from  $w$ , and for each returned node we check if its red friends contains  $w$ . Each red friend has  $w$  within twice its radius, so by Lemma 2, after discovering a constant number of these **r**-colored nodes, we must encounter one that contains  $w$ .

Before turning to prove the main theorem of this section, we have the following lemma which will later help us prove that the spanner has constant degree.

**Lemma 3** *A node  $w'$  may serve as a replacement node for at most a constant number of other nodes.*

**Proof:** If  $w' \in Y_{5^m}$  serves as a replacement for a node  $w$  in level  $k$ , then the ancestor  $u$  of  $w$  in level  $m$  of  $T$  (the original spanning tree) did not survive in  $T_1$ . This implies that  $u$  has no real descendants other than  $w$  and  $w$ 's descendants. Notice that since  $w$  is in level  $k$  and it survived in  $T_1$  no descendant of  $w$  can have  $w'$  as its replacement as the level of the replacement must be less than the level of the parent. Thus, no descendant of  $u$  other than  $w$  can have  $w'$  as a replacement. Furthermore,  $u$  must be within distance  $2 \cdot 5^m$  of  $w'$  since  $d(w, w') \leq 5^m$  and by close-containment  $d(w', u) < 4 \cdot 5^{m-1}$ . There may only be a constant number of nodes in level  $Y_{5^m}$  in  $T$  within distance  $2 \cdot 5^m$  of  $w'$ , from which we may conclude that  $w'$  can serve as the replacement for only a constant number of nodes.  $\square$

The following theorem proves that the previous construction, with  $c$ -level refinement edges for some constant  $c$ , yields a constant degree spanner of the point set  $S'$ .

**Theorem 4** *The procedure described above creates a  $(1 + \epsilon)$ -spanner with constant degree.*

**Proof.** *Constant degree.* Consider a node  $w$ . We first focus on edges going from  $w$  to other nodes.  $w$  has an edge to its parent and another edge to its replacement node, and a constant number of intra-level edges to nodes in the same level.

We now focus on edges going from other nodes to  $w$ . From Lemma 3 it follows that at most a constant number of nodes have  $w$  as their replacement, so the number of these edges is bounded. Moreover, it follows from packing considerations that  $w$  has only a constant number of children, so the number of child-parent edges onto  $w$  is also constant. We now consider nodes that have intra-level edges to  $w$ . All of these nodes must be below the replacement node of  $w$  since it has been stipulated that nodes above the replacement node of  $w$  cannot connect to  $w$ . None of these nodes have  $w$  within their radius (since  $w$ 's replacement is the lowest node that has  $w$  within its radius), however all these nodes have  $w$  within a factor of 27 times their radius. It follows from Lemma 2 that there may be only a constant number of such nodes, and therefore a constant number of intra-level edges to  $v$ .

Since each node has a constant number of children, each refinement step increases this total by only a constant factor. As every node in  $S'$  corresponds to a unique point we conclude that the degree of the spanner for  $S'$  is constant.

*Stretch.* We now turn to prove the stretch of the spanner, but first we define the notion of a *replacement path*, a path from a node  $v \in Y_{5^k}$  towards a node in level  $Y_{5^\ell}$ : The path begins with  $v$ , and at each step the path continues with the current node's replacement. If the current node does not have a replacement, then the path continues with the current node's parent in  $T_1$ . The path terminates when the next candidate node is above  $Y_{5^\ell}$ . The spanner distance from  $v$  to any node  $u \in Y_{5^m}$  ( $m \leq \ell$ ) on  $v$ 's replacement path ( $d_{\text{SP}'}(v, u)$ ) is less than  $5^m \sum_{i=0}^{\infty} (\frac{1}{5})^i = \frac{5}{4}5^m$ , and of course this is an upper bound on  $d(v, u)$  as well.

Let  $p$  and  $q$  be two leaves, and choose  $j$  so that  $5^j \leq d(p, q) < 5^{j+1}$ . Let the last node of the replacement path from  $p$  ( $q$ ) to level  $j - 2$  be  $p'$  ( $q'$ ). It is clear that  $d(p', q') \leq d(p, q) + d(p, p') + d(q, q') \leq d(p, q) + \frac{5}{4}5^{j-2} + \frac{5}{4}5^{j-2} \leq d(p, q) + \frac{5}{2}5^{j-2}$ . It is also clear that  $d_{\text{SP}'}(p, q) \leq d_{\text{SP}'}(p', q') + d_{\text{SP}'}(p', p) + d_{\text{SP}'}(q', q) \leq d_{\text{SP}'}(p', q') + \frac{5}{4}5^{j-2} + \frac{5}{4}5^{j-2} = d_{\text{SP}'}(p', q') + \frac{5}{2}5^{j-2}$ . Below we show that even if only 1-level refinement edges are used, edge  $\langle p', q' \rangle$  will be in the spanner, so  $d_{\text{SP}'}(p', q') = d(p', q')$ . It follows immediately that  $d_{\text{SP}'}(p, q) \leq d_{\text{SP}'}(p', q') + \frac{5}{2}5^{j-2} = d(p', q') + \frac{5}{2}5^{j-2} \leq d(p, q) + \frac{5}{2}5^{j-2} + \frac{5}{2}5^{j-2} \leq d(p, q) + 5 \cdot 5^{j-2}$ . The stretch of the spanner is  $\frac{d_{\text{SP}'}(p, q)}{d(p, q)} < \frac{d(p, q) + 5 \cdot 5^{j-2}}{d(p, q)} = 1 + \frac{5 \cdot 5^{j-2}}{d(p, q)}$ . This term is maximized when  $d(p, q)$  takes on its minimum value, which is  $5^j$ . It follows that the spanner has stretch  $1 + \frac{5 \cdot 5^{j-2}}{5^j} = 1 + \frac{1}{5} = \frac{6}{5}$ .

It follows easily from the previous analysis that using  $c$ -level refinement edges reduces the stretch of the spanner. Let  $p''$  ( $q''$ ) be the highest descendant of  $p'$  ( $q'$ ) and ancestor of  $p$  ( $q$ ) in  $T_1$  at level  $j - c + 1$  or lower. Because  $c$ -level refinement edges are used, edge  $\langle p'', q'' \rangle$  must be in the spanner. We repeat the previous argument with  $p'$  and  $q'$  replaced respectively by  $p''$  and  $q''$ , and conclude that the resulting spanner has stretch less than  $1 + \frac{5 \cdot 5^{j-c+1}}{5^j} = 1 + \frac{1}{5^{c-2}} = 1 + \frac{25}{5^c}$ .

It remains only to demonstrate that  $p'$  and  $q'$  are indeed connected in the spanner. Let  $r \in Y_{5^n}$  ( $n \geq j - 1$ ) be the lowest node among the parents and replacements of  $p'$  and  $q'$ . We assume without loss of generality that  $r$  is the replacement of  $p'$ . Note that since  $r$  contains  $p'$ ,  $d(r, p') \leq 5^n$ . Also recall from above that  $d(p', q') < d(p, q) + \frac{5}{2} \cdot 5^{j-2} < 5^{j+1} + \frac{5}{2} \cdot 5^{j-2}$ . We can now prove that  $r$  possesses an intra-level connection to  $q'$ : Note that  $d(r, q') \leq d(r, p') + d(p', q') < 5^n + 5^{j+1} + \frac{5}{2} \cdot 5^{j-2}$ . Since  $5^{j+1} + \frac{5}{2} \cdot 5^{j-2} = 25 \cdot 5^{j-1} + \frac{1}{2}5^{j-1} = \frac{51}{2}5^{j-1} < 26 \cdot 5^{j-1}$ , the distance from  $r$  to  $q'$  is less than a factor of 27 times the radius of  $r$ . As the parent and replacement of  $q'$  are higher than  $r$ ,  $r$  must have an intra-level connection to  $q'$ . Finally, the refinement step implies that since  $r$  is connected to  $q'$ ,  $p'$  is connected to  $q'$  as well.  $\square$

**Dynamic updates.** In this section we discuss how to maintain the spanner dynamically under insertion and deletion of a point to the set.

*Insertions.* Recall that the insertion of a point into the point set results in the addition of a single leaf node to the spanning tree  $T$ , and possibly the addition of a large number of internal nodes into  $T$ . These changes are handled by the data structure of [6]. The changes to  $T_1$  are restricted to the addition of a single leaf node, and possibly a single internal node as well. Determining the parents of the new leaf node and internal node in  $T_1$  can be done easily in  $O(\log n)$  time by using the search structure of [6].

After the addition of the new nodes the edges of the points that corresponds to them are added as well, as follows. First, parent-child edges are added according to the parent child relations of the inserted nodes. To add the intra-level edges, we query  $T$  (again by using the data structure of [6]) and locate nodes that are near the newly inserted nodes. More specifically, we add intra-level edges to all nodes that are near the new nodes in their level, and also survived in  $T_1$ ; if there is a node in  $T$  but not in  $T_1$  that is in the level of the new node and near it, then an intra-level edge is added to this node's highest surviving child, as described earlier. We then locate the replacement nodes of the new nodes, again using the technique previously described.

Now that we have added edges from the new nodes, we must also add the edges onto the new nodes. For the internal node, we add an edge from its new child to it, and remove from the child any edges to or from nodes in levels that are above its new parent's level. Then for each new node, we must find the nodes that have intra-level edges to it – These are the nodes of  $T_1$  which lie in a level  $i$  that is between the levels of the new node and its parent, and have the ancestor of the new node at level  $Y_{5^i}$  (in the spanning tree  $T$ ) within a distance of 27 times their radius. It is easy to find these nodes using colored ancestor queries: It suffices for every  $\mathbf{r}$ -colored node in  $T$  to color  $\mathbf{b}(\text{lue})$  every node in  $T_1$  that is with 27 times the radius of the original node. Then we execute a colored ancestor query on the new node and  $\mathbf{b}$  to determine all red nodes of interest, of which there may only be a constant number.

A further complication may arise if the internal node added to  $T_1$  (say,  $v \in Y_{5^k}$ ) is itself a candidate to be a replacement node for another preexisting node. To discover if  $v$  should be a replacement node, we use  $T$  to enumerate the friends of  $v$ . For each friend of  $v$  that does not survive in  $T_1$ , we execute a colored descendant query on  $\mathbf{r}$  to find its highest surviving descendant. If  $v$  contains this descendant and is lower than the descendant's current replacement node, then  $v$  becomes the new replacement node of the descendant. The descendant's connections to the old replacement node, as well as to any nodes in levels above  $Y_{5^k}$ , are removed.

*Deletions.* Recall that the deletion of a point into the point set has no effect on  $T$ , but will cause the deletion of a single leaf node and perhaps a single internal node in  $T_1$ . All connections to these nodes are removed. The internal node served as a parent for its child node, and may have further served as a replacement for some other node. In both these cases the remaining nodes must be updated, in a manner identical to what was described for insertions.

## 4.2 Construction of a spanner for the original set $S$

In the previous section, we showed how to construct and dynamically maintain a spanner for the set  $S'$  that corresponds to the nodes of  $T_1$ . However,  $S'$  contains points that do not exist in  $S$ . In this section, we show that each point of  $S' - S$  can be assigned a unique nearby point in  $S$  which can take its place in the spanner.

Hence, we view  $P'$  (the spanner for  $S'$ ) as a blueprint for creating and maintaining the spanner  $P$  of  $S$ .  $P$  is created from  $P'$  via a small change in  $P'$  – the swapping of points previously mentioned. This perturbation of  $P'$  incurs only a small additional cost, and we will show that this cost can be offset by adding just one

more level of refinement edges. Since it was shown that the spanner  $P'$  can be maintained dynamically with only a constant number of changes to the spanner, the same will hold for  $P$ .

To show how points in  $S$  may replace points in  $S' - S$ , we revisit the spanning tree  $T_1$  that implied  $S'$ . It follows from the construction of  $T_1$  that all its leaves correspond to points of  $S$ . However, the internal nodes of  $T_1$  may correspond to points no longer in  $S$ . Further, even an internal node that corresponds to a point in  $S$  may be problematic, since a single point of  $S$  may correspond to many nodes of  $T_1$  and therefore have many edges in the spanner.

To solve these issues, we introduce the following point assignment scheme for internal nodes and their corresponding points in  $S'$ . Assume an arbitrary left-right ordering on the children of every node in  $T_1$ . We assign to each internal node the point that corresponds to the leftmost leaf descendant of its rightmost child. (Recall that each internal node must have at least two children.) Note that this assigns a point that corresponds to a leaf to at most one internal node; hence, every point in  $S'$  corresponding to an internal node is assigned a unique point in  $S$ . Now we alter the spanner  $P'$  by swapping out each point in  $S'$  for its assignment point in  $S$ , and rerouting the edges to the new point. The resulting spanner is  $P$ .

$P$  is a spanner for  $S$ , since it contains only points in  $S$ , and any route that existed in between leaf nodes of  $T_1$  still exists in a modified form in the new spanner. Also, because  $P'$  has constant degree, and a point of  $S$  corresponds to at most two points in  $S'$ ,  $P$  also has constant degree. Now recall that the close-containment property implies that the distance from a node at level  $i$  to any of its descendants is less than  $4 \cdot 5^{i-1}$ . It follows that replacing internal nodes by their assigned points (which are descendants of the replaced point) adds a small error per point, but it is not difficult to see that these cumulative errors may be offset by adding a single refinement level.

More rigorously, to prove the stretch of  $P$  we revisit the proof for the stretch of  $P'$  (the proof of Theorem 4). Recall that each point of  $S$  correspond to a leaf node in  $T_1$ . As in the proof, let  $p$  and  $q$  be two leaves in  $T_1$ , with  $5^j \leq d(p, q) < 5^{j+1}$ , and let  $p''$  ( $q''$ ) be the highest tree ancestor of  $p$  ( $q$ ) in level  $j - c + 1$  or lower. We have already demonstrated that if  $c$ -level refinement edges are used, edge  $\langle p'', q'' \rangle$  is found in  $P'$ . It is also easy to see that  $d(p'', q'') \leq d(p, q) + d(p'', p) + d(q'', q) < d(p, q) + \frac{5}{4}5^{j-c+1} + \frac{5}{4}5^{j-c+1} = d(p, q) + \frac{5}{2}5^{j-c+1}$ .

Let  $a(p'')$  and  $a(q'')$  be the assignment points for  $p''$  and  $q''$  respectively. Since edge  $\langle p'', q'' \rangle$  is found in  $P'$ ,  $\langle a(p''), a(q'') \rangle$  is found in  $P$ . As mentioned above, the close-containment property tells us that  $d(p'', a(p'')) \leq 4 \cdot 5^{j-c}$  and  $d(q'', a(q'')) \leq 4 \cdot 5^{j-c}$ . Hence,  $d_{\text{SP}}(a(p''), a(q'')) = d(a(p''), a(q'')) \leq d(p'', q'') + d(p'', a(p'')) + d(q'', a(q'')) < d(p'', q'') + 4 \cdot 5^{j-c} + 4 \cdot 5^{j-c} = d(p'', q'') + 8 \cdot 5^{j-c} < d(p, q) + \frac{5}{2}5^{j-c+1} + 8 \cdot 5^{j-c} = d(p, q) + \frac{41}{2}5^{j-c}$ .

Now, there is a path in  $P$  from  $a(p'')$  to  $p$  which mimics the path in  $P'$  from  $p''$  to  $p$ . Similar to what was done in the proof of Theorem 4, but taking into account the aforementioned error due to swapping assignment points (an additional  $\frac{4}{5}$  times the radius of the relevant point), we have that  $d_{\text{SP}}(a(p''), p) < 5^{j-c+1} \sum_{i=0}^{\infty} (1 + \frac{4}{5})(\frac{1}{5})^i = 5^{j-c+1}(1 + \frac{4}{5})(\frac{5}{4}) = \frac{9}{4}5^{j-c+1}$ . This quantity is an upper bound on  $d_{\text{SP}}(a(q''), q)$  as well. Finally, we can conclude that  $d_{\text{SP}}(p, q) \leq d_{\text{SP}}(a(p''), a(q'')) + d_{\text{SP}}(a(p''), p) + d_{\text{SP}}(a(q''), q) < d(p, q) + \frac{41}{2}5^{j-c} + \frac{9}{4}5^{j-c+1} + \frac{9}{4}5^{j-c+1} = d(p, q) + 43 \cdot 5^{j-c}$ . The stretch of the spanner is  $\frac{d_{\text{SP}}(p, q)}{d(p, q)} = \frac{d(p, q) + 43 \cdot 5^{j-c}}{d(p, q)} = 1 + \frac{43 \cdot 5^{j-c}}{d(p, q)}$ . This quantity is maximized when  $d(p, q)$  attains its minimum value of  $5^j$ , so the stretch of  $P$  is less than  $1 + \frac{43 \cdot 5^{j-c}}{5^j} = 1 + \frac{43}{5^c}$ . This is close to the stretch of  $P'$  of  $1 + \frac{25}{5^c}$ .

**Dynamic maintenance.** This assignment scheme can easily be maintained dynamically. Recall that an insertion or deletion of a point into  $S$  results in the insertion or deletion of at most two nodes in  $T_1$  (and two points in  $S'$ ). The effect of inserting one node into  $T_1$  is very minor: If an internal node is inserted we must find a new assignment for it. If the internal node is inserted as a rightmost child, then its parent's assignment must be updated as well. If the internal node or a leaf node is inserted as a leftmost child, then it is possible that an assignment of one of its ancestors must be updated as well. The deletion of a node may cause the deleted node's ancestor to require a new assignment. In any event, all these functions

can be done easily in  $O(\log n)$  time using standard dynamic tree structures, and we omit further details.

We now turn to maintaining the spanner for  $S$ . Recall that the spanner for  $S$  is formed from the spanner of  $S'$  by swapping out points of  $S' - S$  for points of  $S$ . Following an insertion or deletion into  $S$ , the spanner for  $S'$  may change in the following way: A constant number of points are added to the spanner, and a constant number of edges are inserted or deleted. It follows that the changes to the spanner of  $S$  are restricted to updating the assignment, and inserting or deleting a constant number of edges.

## References

- [1] S. Alstrup, T. Husfeldt, and T. Rauhe. Marked ancestor problems. *(IEEE) Symposium on Foundations of Computer Science*, pages 534–544, 1998.
- [2] S. Arya, D. M. Mount, S. Nathanyahu, R. Silverman, and A.Y. Yu. An optimal algorithm for approximate nearest neighbor searching in fixed dimension. *Journal of the ACM*, 45(6):891–923, 1998.
- [3] 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.
- [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] R. Cole and L. Gottlieb. Searching dynamic point sets in spaces with bounded doubling dimension. In *ACM Symposium on Theory of Computing*, 2006.
- [7] G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished Euclidean graphs. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 215–222, 22–24 1995.
- [8] J. Gao, L. Guibas, and A. Nguyen. Deformable spanners and applications. In *ACM Symposium on Computational Geometry*, 2004.
- [9] L. Gottlieb and L. Roditty. Improved algorithms for fully dynamic geometric spanners and geometric routing. In *ACM Symposium on Discrete Algorithms*, 2008.
- [10] A. Gupta, R. Krauthgamer, and J.R. Lee. Bounded geometries, fractals, and low-distortion embeddings. *(IEEE) Symposium on Foundations of Computer Science*, pages 534–543, 2003.
- [11] 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.
- [12] R. Krauthgamer and J. Lee. Navigating nets: Simple algorithms for proximity search. In *ACM-SIAM Symposium on Discrete Algorithms*, 2004.
- [13] L. Roditty. Fully dynamic geometric spanners. In *ACM Symposium on Computational Geometry*, 2007.
- [14] J. S. Salowe. Constructing multidimensional spanner graphs. *Int. J. Comput. Geometry Appl*, 1(2):99–107, 1991.
- [15] J. Soares. Approximating euclidean distances by small degree graphs. *Discrete & Computational Geometry*, 11:213–233, 1994.

- [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.