# SPANNERS FOR GEOMETRIC INTERSECTION GRAPHS WITH APPLICATIONS[*]

*Martin Fürer*[†] *Shiva Prasad Kasiviswanathan*[‡]

Abstract. A ball graph is an intersection graph of a set of balls with arbitrary radii. Given a real number $t > 1$, we say that a subgraph $G'$ of a graph $G$ is a $t$-spanner of $G$, if for every pair of vertices $u, v$ in $G$, there exists a path in $G'$ of length at most $t$ times the distance between $u$ and $v$ in $G$. In this paper, we consider the problem of efficiently constructing sparse spanners of ball graphs which supports fast shortest path distance queries.

We present the first algorithm for constructing spanners of ball graphs. For a disk graph in $\mathbb{R}^2$, we construct a $(1+\epsilon)$-spanner for any $\epsilon > 0$ with $O(n\epsilon^{-2})$ edges in $O(n^{4/3+\delta}\epsilon^{-4/3}\log^{2/3} S)$ time, using an efficient partitioning of the plane into squares and solving intersection problems. Here $\delta$ is any positive constant, and $S$ is the ratio between the largest and smallest radius. For the special case when the disks all have unit size, we show that the complexity of constructing a $(1+\epsilon)$-spanner is almost equal to the complexity of constructing a Euclidean minimum spanning tree. The algorithm extends naturally to other "disk-like" objects, also in higher dimensions.

The algorithm uses an efficient subdivision of space to construct a sparse graph having many of the same distance properties as the input ball graph. Additionally, the constructed spanners have a small vertex separator decomposition (hereditary). In dimension 2, the disk graph spanner has an $O(\sqrt{n}\epsilon^{-3/2} + \epsilon^{-3}\log S)$ separator. The presence of a small separator is then exploited to obtain very efficient data structures for approximate distance queries. The results on geometric graph separators might be of independent interest. For example, since complete Euclidean graphs are just a special case of (unit) ball graphs, our results also provide a new approach for constructing spanners with small separators in these graphs.

## 1 Introduction

Let $G = (V, E)$ be a weighted graph, and let $d_G(u, v)$ be the length of a shortest path between vertices $u$ and $v$ in $G$. For any fixed $\epsilon > 0$, a $(1+\epsilon)$-spanner of $G$ is a subgraph $G'$ such that for every pair of vertices $u$ and $v$, $d_{G'}(u, v)/d_G(u, v) \leq (1+\epsilon)$. Spanners are important data structures because they provide a way of approximating a graph in an economical way. Spanner constructions have been widely investigated for general graphs

[*]This article is primarily based on the abstract of [15], with some material from the earlier abstract of [14]. The work was supported by the National Science Foundation under Grants CCR-0209099, CCF-0728921, and CCF-0964655.

[†]*Department of Computer Science and Engineering, Pennsylvania State University*, furer@cse.psu.edu

[‡]*IBM T.J. Watson Research Center*, kasivisw@gmail.com

and complete Euclidean graphs, also with additional properties like weight, diameter, and degree [32, 8, 6, 10].

We also consider a related problem of preprocessing a graph such that subsequent distance queries can be answered quickly within a small error. This natural extension to the all pairs shortest path problem captures practical situations, where we often are interested in estimating the distance between two vertices quickly and accurately [40, 42]. In this framework, the goodness of an algorithm is typically measured in terms of the preprocessing time, query time, space complexity and approximation factor.

We present a new method for producing spanners of geometric graphs based on a hierarchical decomposition of the plane into tiles of various sizes. Our constructions are quite general, as they are not restricted to complete Euclidean graphs, but extend to geometric disk graphs, as well as their higher dimensional versions, the ball graphs. In all cases, edge lengths are given by Euclidean distances, but not all edges have to be present in our graphs. The difficulty in constructing a spanner for the disk graph metric when compared to the metric induced by a complete Euclidean graph is that two points that are close in space are not necessarily close under the graph metric. The constructed spanner also has a small vertex separator. The presence of a small separator is then exploited to obtain very efficient data structures for approximate distance queries.

Intersection graphs are graphs whose vertices are represented by sets such that two vertices are adjacent if and only if the corresponding sets have a non-empty intersection. A disk graph is an intersection graph of disks in the plane. We consider weighted disk graphs where the weight of an edge is the Euclidean distance between centers. In addition to their theoretical interest, such graphs have been used widely to model the communication between objects in VLSI [31] and in the context of wireless networks [33, 25]. For wireless networks they represent the fact that two wireless nodes can directly communicate with each other only if they are within a certain distance[1].

Spanners are important for disk graphs because restricting the size of a network reduces the amount of routing information. Spanners are used in topology control for maintaining network connectivity, improving throughput, and optimizing network lifetime [33, 25]. Distance queries are important in disk graphs as they are widely used to determine coverage in wireless sensor networks, and for routing protocols [27, 39, 17]. In most potential applications one would not only desire high accuracy of these estimates but also the actual path producing this estimate. Our approximation algorithms are designed keeping this in mind.

Geometric spanner constructions for disk-like graphs have been widely investigated in both theory and networking communities. Many constructions, both centralized and distributed, also with additional properties like planarity and power saving have been proposed [33, 26, 25, 16, 28]. However, all these constructions only work for some restricted cases of disk graphs (e.g., unit disk graphs).

[1]More realistic models for wireless networks like quasi unit-disk graphs [24], SINR [29] have also been widely studied in the literature and are known to model wireless networks better than undirected (unit) disk graphs.

## 1.1 Our Contributions

The main goal of this paper is to construct a sparse spanner for ball graphs. Our spanners are constructed using a hierarchical partitioning of the space, which produces small separators in a natural way. Therefore, we can support applications which profit from both, the sparseness of the spanner and the separator properties. Even though our construction is conceptually simple, its fast implementation requires carefully selected data structures. We now outline our contributions in more detail.

We present the first algorithm for constructing sparse spanners of general ball (disk) graphs. Let $S$ denote the ratio of the radii of the largest and smallest balls in the graph $G$. For ball graphs in $\mathbb{R}^k$, we construct spanners with $O(n\epsilon^{-k})$ edges. For the interesting case when $S$ is polynomially bounded the algorithm runs in $O(n^{(2/\ell)+\delta}\epsilon^{-k/\ell})$ time where $\ell = 1 + 1/(\lfloor k/2 \rfloor + 1)$ and $\delta$ is any positive constant. In general, the algorithm runs in $O(n^{(2/\ell)+\delta}\epsilon^{-k/\ell}\log^{1/\ell} S)$ time. Note that for $k \geq 2$, $\ell > 1$.

In the case when all the balls have the same radius (unit ball graphs), the algorithm has $\tilde{O}(n\epsilon^{-2})$[2] running time for $k = 2$, $\tilde{O}(n^{4/3}\epsilon^{-3})$ expected running time for $k = 3$, and $O(n^{2-2/(\lceil k/2 \rceil+1)+\delta}\epsilon^{-k})$ expected running time for $k \geq 4$. Additionally, for unit ball graphs, we show that constructing $(1+\epsilon)$-spanners has randomized complexity almost equivalent to the construction of a Euclidean minimum spanning tree. Therefore, we cannot hope to find a faster algorithm for constructing spanners of unit ball graphs, unless we improve on some other well-studied problems [12]. The previously best known constructions of spanners of unit ball graphs were primarily based on the Yao graph [41] construction and have $O(n^{2-a(k)})$ running time for $a(k) = 2^{-k+1}$ in dimensions $k$ greater than 3.

Our spanners also have a small separator decomposition. Note that the input graph might have no small separator, it could even be complete. The constructed spanners have an $O(n^{1-1/k}\epsilon^{-k+1/2} + \epsilon^{-2k+1}\log S)$ vertex separator, which can be found in $\tilde{O}(n)$ time. Since complete Euclidean graphs are just a special case of (unit) ball graphs, our results also provide a new approach for constructing spanners with small separators in these graphs. Our result can be seen as an extension of a result by Smith and Wormald [38], who show the existence of separators of size $O(n^{1-1/k}\epsilon^{-O(1)})$ in the Arya *et al.* [6] spanners of the complete Euclidean graphs. Separators for disk graphs with bounded $S$ were also investigated in [4, 13].

Using this, we obtain fast algorithms for approximately answering distance queries in ball graphs. An estimate $\delta(u,v)$ of the path length $d_G(u,v)$ is said to be a $c$-stretch if it satisfies $d_G(u,v) \leq \delta(u,v) \leq c d_G(u,v)$. We show that the spanner can be preprocessed in $O(nf(n,S,\epsilon)\log n)$ time and $O(nf(n,S,\epsilon))$ space, such that subsequent distance queries under the $d_G$ metric can be answered with $(1+\epsilon)$-stretch in $O(f(n,S,\epsilon))$ time, where $f(n,S,\epsilon) = n^{1-1/k}\epsilon^{-k+1/2} + \epsilon^{-2k+1}\log S$ is the size of the separator.

At the expense of a slightly higher preprocessing time we also obtain a better error guarantee on the queries. This better error guarantee is formalized by the notion of *strong*

[2]We use the notation $\tilde{O}(f) \equiv O(f\text{polylog}(f))$ (i.e., $\tilde{O}$ ignores logarithmic factors, not merely constants).

$(1+\epsilon)$-*approximation*. A $(1+\epsilon)$-approximate estimate $\delta(u,v)$ is said to be strong if

$$d_G(u,v) \leq \delta(u,v) \leq d_G(u,v) + \epsilon \cdot \zeta(u,v), \text{ where}$$

$$\zeta(u,v) = max\{\ell \mid \exists \text{ a shortest path in } G \text{ between } u \text{ and } v \text{ with maximal edge length } \ell\}.$$

Let $G$ be a ball graph with $m = \Omega(n \log n)$ edges. We show that after $O(mf(n,S,\epsilon))$ time and $O(nf(n,S,\epsilon))$ space preprocessing, a strong $(1+\epsilon)$-approximate estimate for the distance between any two vertices can be obtained in $O(f(n,S,\epsilon))$ time. In all our cases $\zeta(u,v)$ is strictly less than $d_G(u,v)$. Therefore, this approximation is strictly better than the standard $(1+\epsilon)$-approximation. In both the above results, we can also output a corresponding short path between the query vertices in $O(L)$ time, where $L$ is the number of edges of the reported path.

**Computational Model:** As is standard practice in computational geometry algorithms dealing with quadtrees (e.g., see [7, 22]), we assume in this paper that in addition to standard arithmetic operations certain operations on points in $\mathbb{R}^k$ can be done in constant time. Specifically, the operation needed involves finding the most significant binary digit at which two coordinates of two points differ. This can be done in $O(1)$ machine instructions if we have a most-significant-bit instruction, or by using floating-point or extended-precision normalization. If the coordinates are not in binary fixed or floating point, such operations may also involve computing integer floor and ceiling functions.

## 1.2 Related Work

Subsequent to our paper [15], Abam and Har-Peled [1] presented a new construction of semi-separated pair decomposition for a set of $n$ points in $\mathbb{R}^k$ and used that to obtain a $(1+\epsilon)$-spanner of a complete Euclidean graph. The spanner has $O(n\epsilon^{1-2k})$ edges, maximum degree of $O(\epsilon^{1-2k}\log^2 n)$, and a separator of size $O(n^{1-1/k}\epsilon^{-k})$. Their construction time is $O(n\epsilon^{-k}\log^2 n)$. For the (special) case of complete Euclidean graphs in $\mathbb{R}^k$ with $k > 3$, their construction has some advantages (in terms of running time, maximum degree) over our construction. For a complete Euclidean graph in $\mathbb{R}^2$, the algorithm presented in [1] yields a spanner with $O(n\epsilon^{-3})$ edges, maximum degree of $O(\epsilon^{-3}\log^2 n)$, separator of size $O(\sqrt{n}\epsilon^{-2})$, and takes $O(n\epsilon^{-2}\log^2 n)$ time to construct. Our algorithm, on the other hand, yields a spanner with $O(n\epsilon^{-2})$ edges, a separator of size $O(\sqrt{n}\epsilon^{-3/2})$, and takes $O(n\epsilon^{-2}\log n)$ time to construct (however, unlike the spanner of [1], our spanner could have maximum degree of $\Omega(n)$).

For general undirected graphs, Thorup and Zwick [40] show that for any $c \geq 1$, a graph with $n$ vertices and $m$ edges can be preprocessed in $O(cmn^{1/c})$ expected time, constructing a data structure of size $O(cn^{1+1/c})$, such that a $(2c-1)$-stretch answer to any distance query can be produced in $O(c)$ time. Many other time-space trade-off results are also known (see Zwick's [42] survey on this subject). Better results are available for some special classes of graphs. For example, if the graph $G$ is a geometric $O(1)$-spanner with $m$ edges, then Gudmundsson *et al.* [18, 20, 19] show that $G$ can be preprocessed in $O(m + n\log n)$ time, constructing a data structure of size $O(n\log n)$, such that a $(1+\epsilon)$-stretch answer to any distance query can be produced in $O(1)$ time. For unit disk graphs, Gao and Zhang [17] gave a construction that produces a data structure of size $O(n \log n)$ in $O(n\sqrt{n \log n}\epsilon^{-3})$ time, such that a $(1+\epsilon)$-stretch answer to any distance query can be produced in $O(1)$ time.

**Organization:** The rest of the paper is organized as follows. In Section 2, we introduce some terminology. In Section 3, we describe the basic ideas behind our construction. In Section 4, we present the spanner construction algorithm. In Section 5, we present and analyze the algorithm for finding a separator decomposition of the spanner. In Section 6, we present algorithms for fast answering of distance queries. We conclude in Section 7.

## 2 Preliminaries

Let $\mathcal{P} = \{v_1, \ldots, v_n\}$ be a set of points in $\mathbb{R}^k$ for any fixed dimension $k$. Let $\mathcal{D} = \{D_{v_1}, \ldots, D_{v_n}\}$ be a set of $n$ balls such that (1) $D_u$ is centered at $u \in \mathcal{P}$, and (2) $D_u$ has radius $r_u$. Balls $D_u$ and $D_v$ intersect if $d(u,v) \leq r_u + r_v$, where $d(\cdot,\cdot)$ denotes the Euclidean metric. The ball graph $G = (V = \mathcal{P}, E)$ is a weighted graph where an edge between $u$ and $v$ with weight $d(u,v)$ exists iff $D_u$ and $D_v$ intersect. Let $d_G$ denote the shortest path metric induced by the connected graph $G$ on its vertices.

We use $m$ to denote the number of edges in the graph $G$. We require that the input to the algorithms is the set of balls, not only the corresponding intersection graph. Without loss of generality we assume that for every ball $D_u \in \mathcal{D}$ there exists at least one ball center outside $D_u$[3]. We then re-scale the balls such that the largest radius equals one. The global scale factor (ratio of radii of largest and smallest balls) of $\mathcal{D}$ is then defined as

$$S(\mathcal{D}) = 1/\min\{r_u \mid D_u \in \mathcal{D}\}.$$

For a node $t$ in a tree, denote by $p(t)$ the parent of $t$ in the tree. We use $d_t$ to denote the level (depth) of node $t$ in the tree. In two dimensions, our algorithms recursively partitions the plane into squares and this generates a tree (akin to quadtrees). A point $(x,y)$ is *contained* in the node $t$ representing a square with center $(x_t, y_t)$ and length $l_t$ in the tree iff

$$x_t - l_t/2 \leq x < x_t + l_t/2 \quad \text{and} \quad y_t - l_t/2 \leq y < y_t + l_t/2.$$

For a set of squares $T$ in the tree, a point is contained in $T$ iff there exists $t \in T$, such that point is contained in $t$. The distance between two squares is the Euclidean distance between their centers.

Throughout the paper we refer to the vertices of a graph as *vertices* and vertices of a tree as *nodes*. We assume without loss of generality that $\epsilon^{-1}$ is a power of 2 to avoid many floors and ceilings. Note that for a given $\epsilon$ it is sufficient to construct a $(1+c\epsilon)$-spanner for a fixed constant $c$ as we could always start be scaling down $\epsilon$ by a factor of $c$. All logarithms (log) in this paper are base 2.

[3]This is without loss of generality because if there exists a ball that contains all other ball centers within it, then we can shrink the big ball safely till at least one ball center lies outside it and this shrinking does not change the graph. This scaling is not mandatory for our algorithms, but from a practical perspective this scaling helps in reducing the global scale factor which shows up in our results

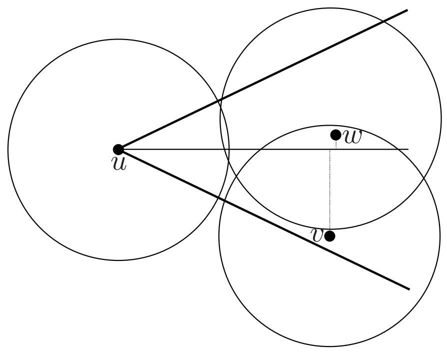

Figure 1: Figure illustrating the fact that Θ-graphs fail to be spanners of unit disk graphs. $u$, $v$, and $w$ are disk centers. The point $w$ is closer to $u$ than $v$. There is an edge $(u, w)$, but no edge $(u, v)$ in the unit disk graph. The distance between $u$ and the projection of $v$ (onto the angle bisector) is less than the distance between $u$ and the projection of $w$ (onto the angle bisector). If we add an edge $(u, v)$ to the Θ-graph then the path between $u$ and $v$ in the Θ-graph is shorter than the path between $u$ and $v$ in the original unit disk graph. If we don't add the edge then there may not be any path between $u$ and $w$ in the Θ-graph.

## 3 Yao Graphs and Well-Separated Pair Decompositions

In this section we present an overview of some of the basic concepts that we use to obtain the spanner results. Both the constructions of Yao graphs and well-separated pair decompositions have been widely used to construct geometric spanners (see for example [32]).

### 3.1 Modified Yao Graph

A Yao graph [41] construction involves partitioning the space around each point into cones with a fixed opening angle and connecting the point to its nearest neighbor in each cone. Even though constructing the original Yao graph is costly (see [41]), a variant of it called the Θ-graph can be constructed in $O(n \log^{d-1} n + n\epsilon^{-k})$ time [34]. In a Θ-graph points inside a cone are projected onto the angle bisector of the cone and the closest point is used to add an edge. Even though Θ-graphs are spanners of the complete Euclidean graphs, they fail to be spanners even for unit ball graphs (as these graphs are not complete graphs). See the example in Fig. 1.

It is known that, for unit disk (ball) graphs the Yao-graph with long edges (edges of length greater than 2) removed is a spanner (see, e.g., [32], Chapter 4). But no prior algorithm was known for constructing spanners for general disk graphs. We now define a *modified* Yao graph, which forms a sparse spanner of the general disk (ball) graph. We later show how a variant of this modified Yao graph can be constructed efficiently. Let $\mathcal{C}(p) = \{co_1(p), \ldots, co_{\epsilon^{-1}}(p)\}$ be a collection of $\epsilon^{-1}$ cones such that: (1) each cone has its apex at $p \in \mathcal{P}$, (2) each cone has an opening angle of $2\pi\epsilon$, and (3) the union of these cones cover $\mathbb{R}^2$.

**Definition 1** (Modified Yao Graph). *The vertices of a modified Yao graph $Y$ are the points*

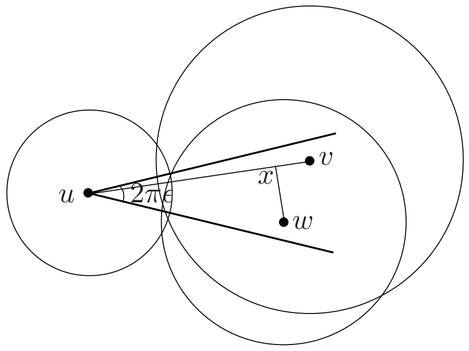

Figure 2: Figure illustrating the proof of Lemma 1.

*of $\mathcal{P}$. For each $p \in \mathcal{P}$ and integer $i$ ($1 \leq i \leq \epsilon^{-1}$), add an edge from $p$ to the point $q$ contained in $co_i(p)$ if*

1. *$r_q \geq r_p$,*
2. *the edge $(p, q)$ exists in the disk graph $G$, and*
3. *$q$ is the closest point in $co_i(p)$ to $p$ satisfying the previous two conditions.*

**Lemma 1.** *Let $(u, v)$ be an edge in the disk graph $G$. Then there exists a path in the modified Yao graph $Y$ of $G$ such that $d_Y(u, v) \leq (1 + c\epsilon)d(u, v)$ for some constant $c$ and all sufficiently small $\epsilon > 0$.*

*Proof.* Assume $r_u \leq r_v$. Consider the cone $co_i(u)$ containing $v$. Let $(u, w)$ be the edge added in $Y$ from $u$ to the point $w$ which is also contained in $co_i(u)$. Let $x$ be the projection of the point $w$ on the line connecting $u$ and $v$. See Fig. 2. Then $d(u, w) \leq d(u, x) + d(x, w)$ and $d(w, v) \leq d(x, v) + d(x, w)$. Adding these inequalities we get,

$$\begin{aligned} d(u,w) + d(w,v) &\leq d(u,x) + d(x,v) + 2d(x,w) \\ &= d(u,v) + 2d(x,w) \leq d(u,v) + 4\pi\epsilon d(u,w). \end{aligned}$$

The last inequality holds because $\frac{d(x,w)}{d(u,w)} \leq \sin(2\pi\epsilon) < 2\pi\epsilon$. Now for $\epsilon \leq 1/8$ (cones have opening angle $\leq \pi/4$), $d(w, v) < d(u, v)$. Since $r_u \leq r_w$ and there exists an edge $(u, v)$, it implies that there also exists an edge $(w, v)$ in $G$. We complete the proof by induction. We inductively assume that Lemma 1 holds for all the finitely many pairs of $(u', v')$ of vertices of $G$ with $d(u', v') < d(u, v)$, in particular for $u' = w$ and $v' = v$. Therefore,

$$\begin{aligned} d_Y(u,v) &\leq d(u,w) + d_Y(w,v) \leq d(u,w) + (1 + c\epsilon)d(w,v) \\ &\leq d(u,w) + (1 + c\epsilon)(d(u,v) - (1 - 4\pi\epsilon)d(u,w)) \leq (1 + c\epsilon)d(u,v). \end{aligned}$$

The last inequality is satisfied if $c \geq 78$ and $\epsilon \leq 1/15$. If $r_u \geq r_v$, then we swap $u$ and $v$ in the above proof. □

From the above lemma by summing over all edges of a path in $G$ we also get that graph $Y$ is a $(1+\epsilon)$-spanner of the disk graph $G$. For ball graphs in $\mathbb{R}^k$, the number of edges in $Y$ can be bounded by $O(n\epsilon^{-k+1})$ using the result of Lukovszki ([30], Theorem 2.22). We use the following ideas to *efficiently* construct a *variant* of the modified Yao graph which is still a spanner of the disk graph.

## 3.2 Modified Well-Separated Pair Decomposition

Let $(\mathcal{S}, \rho)$ be a metric space, where $\mathcal{S}$ is a set of elements and $\rho$ the distance function defined on $\mathcal{S} \times \mathcal{S}$. For any subset $T \subseteq \mathcal{S}$, the diameter $Dia_\rho(T)$ is defined to be $\max_{p,q \in T} \rho(p,q)$. The distance $\rho(T, T')$ between two sets $T, T' \subseteq \mathcal{S}$ is defined to be $\min_{p \in T, q \in T'} \rho(p,q)$.

**Definition 2** (Well-Separated Pair Decomposition)**.** *A $\gamma$-well-separated pair decomposition for $(\mathcal{S}, \rho)$ for a real number $\gamma > 0$, is a sequence $\{A_1, B_1\}, \ldots, \{A_m, B_m\}$ of pairs of nonempty subsets of $\mathcal{S}$, such that:*

① *the sets $A_i$ and $B_i$ are disjoint, for every $i$,*

② *for any two distinct points $p$ and $q$ of $\mathcal{S}$, there is an unique pair $\{A_i, B_i\}$ such that $p \in A_i$ and $q \in B_i$, or vice-versa,*

③ *for each pair $\{A_i, B_i\}$, $\rho(A_i, B_i) \geq \gamma \cdot \max\{Dia(A_i), Dia(B_i)\}$.*

For a point set in the $k$-dimensional Euclidean space it is known that an $\epsilon^{-1}$-well-separated pair decomposition with $O(\epsilon^{-k} n)$ pairs exists [9]. Given such a pair decomposition it can easily be converted into a $(1+\epsilon)$-spanner by picking a representative edge (an actual edge of the graph) from each pair into the spanner (the conversion process is explained in detail in [8, 37]).

However, the disk graph metric being more general doesn't have the same nice properties as the complete Euclidean graph metric. In a recent result, Gao and Zhang [17] gave a construction of an $\epsilon^{-1}$-well-separated pair decomposition with $O(\epsilon^{-4} n \log n)$ pairs, for unit disk graphs. They also show that for unit ball graphs in $\mathbb{R}^k$ $(k > 2)$ at least $\Omega(n^{2-2/k})$ pairs are needed. However, one cannot hope to extend these results to general disks graphs, as general disk graphs do not have a sub-quadratic well-separated pair-decomposition. One such example is the star graph, formed by a big disk and $n-1$ pairwise disjoint small disks intersecting the boundary of the big disk.

Our construction implicitly produces a *modified* well-separated pair decomposition. In our algorithm we construct set pairs that are disjoint (condition ①) and well-separated (condition ③). However (unlike Definition 2), we require that the vertices in each individual set form a clique in $G$. We show that for constructing the spanner when combined with ideas of the modified Yao graph, one requires only a linear number of such set pairs. The final challenge lies in minimizing the time for finding the representative edges between set pairs.

## 4 Spanners of Disk Graphs

We first describe a high-level idea of our algorithm. For simplicity, we describe all the algorithms for the dimension $k = 2$ and then state the generalizations to higher $k$. In order to construct a spanner $G'$ of a disk graph $G$, the disks are classified by their approximate radii. Recall that the radii are rescaled such that the largest radius equals one. A disk $D_u$ with radius $r_u$ is said to be of order $\lceil -\log r_u \rceil$. Now a first idea is that every disk is responsible for maintaining the connections to disks of lower or equal order only (i.e., to disks of similar or larger radius). This requirement is relaxed when several disks of equal order are located close together. Then all disks except one representative disk are treated like disks of smaller radii, and only the representative disk is responsible for establishing longer connections.

Occasionally, a small disk $D_u$ is isolated in the sense that there is no disk of lower order nearby. Far away large disks can still produce edges in $G$. Some of these long edges have to be retained in the spanner $G'$ to establish connections with these large disks. In order for $G'$ to have the spanner properties, we notice that it is sufficient to retain in each approximate direction just one edge to a far neighbor $D_v$.

In order to determine clusters of disks with a common approximate radius, the plane is partitioned by a hierarchy of grids (similar to a quadtree space partition). The largest disks of order 0 are classified according to the locations of their centers in the squares of a fixed grid of width $\epsilon$, where $\epsilon$ is an argument to the algorithm for determining the quality of the spanner constructed below. This grid is refined to a grid of width $\epsilon/2$ for the classification of disks of order 0 and 1, and so on. From each class (cluster of large disks centered in a grid box) an arbitrary representative is selected (except that any representative selected is also chosen as the representative of all smaller grid boxes containing the representative's disk center).

### 4.1 Spanner Construction Algorithm

As explained earlier, each disk is associated with an order, a disk $D_u$ is of order $l$ iff $2^{-l} \leq r_u < 2^{-l+1}$. Let $l_{max}$ denote the largest order among disks in $\mathcal{D}$, i.e., $l_{max} = \lceil \log S(\mathcal{D}) \rceil$. The spanner $G'$ is constructed in the following manner.

**Recursive Partitioning:** Our spanner construction involves recursively partitioning the plane using a simple variant of quadtrees. The input to the algorithm is a set of disks in $\mathbb{R}^2$. Let $\mathcal{P}$ be the set of their centers. Define the *bounding box* to be the smallest axis-parallel square enclosing $\mathcal{P}$. The left bottom corner of the bounding box is assumed to be the origin. An $\epsilon$-grid is defined by horizontal and vertical line segments drawn at $y \in \epsilon\mathbb{Z}$ and $x \in \epsilon\mathbb{Z}$ within the bounding box. We recursively partition the $\epsilon$-grid into smaller squares. We view the resulting structure as a 4-ary forest with the root nodes as the non-empty squares in the $\epsilon$-grid. Each square is partitioned into four equal squares, which form its children. We continue partitioning all the non-empty squares until each disk center is contained in a separate square of size $\epsilon 2^{-l_{max}}$ or smaller. See Fig. 3 and Fig. 4.

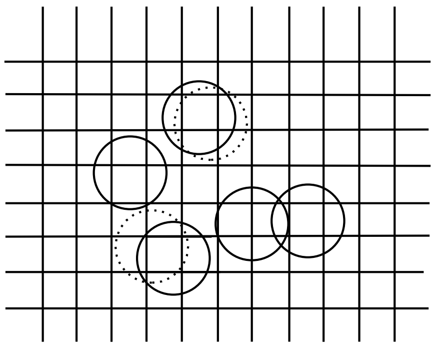

Figure 3: The $\epsilon$-grid with $\epsilon = 1$. Assume all disks have the same radius. The centers of solid disks are the representative vertices used in $G'$.

**Constructing the forest:** Let $\Gamma = (V_\Gamma, E_\Gamma)$ denote the forest from the above recursive partitioning of the $\epsilon$-grid. $\Gamma$ is a collection of disjoint trees. We define the $l$th level of a forest $\Gamma$ as the set of all nodes that have distance $l$ from the roots of the trees in $\Gamma$. The set of nodes at level $l$ in $\Gamma$ corresponds to the set of non-empty squares defined by the $\epsilon 2^{-l}$-grid. For a node $t \in V_\Gamma$ of level $d_t$, define $D(t)$ as the set of disks whose orders are $\leq d_t$ and whose centers are contained in $t$. Let $C(t)$ be the set of disk centers of disks in $D(t)$.

We define *Roots* as the set of nodes in $\Gamma$ where a node $t \in Roots$ if (1) $t$ is a root of a tree in $\Gamma$, or (2) $t \in V_\Gamma$ with $C(t) \neq \emptyset$, whereas $C(p(t)) = \emptyset$ (where $p(t)$ is the parent of $t$ in $\Gamma$).

A node $t$ of the forest is called *interesting* if $C(t) \neq C(p(t))$ and $C(t) \neq \emptyset$. It follows that all nodes $t \in Roots$ which have non-empty $C(t)$ are interesting (for a root node $t$ with no $p(t)$, we assume that $C(p(t)) = \emptyset$). Note that $\Gamma$ has at most $2n - 1$ interesting nodes. For efficiency, instead of $\Gamma$, we construct a compressed forest $\Gamma'$ in which we only introduce the interesting nodes of $\Gamma$ and shortcut the degree one internal nodes. The construction of the compressed forest follows from simple modifications of known algorithms (in [9, 7]) for generating compressed quadtrees. For the sake of completeness, in Appendix A, we describe a construction which is very similar to that of Bern *et al.* [7]. Readers familiar with the construction of compressed quadtrees can skip this construction.

For convenience, we will use $\Gamma$ to describe the high-level ideas and to prove the spanner property (Section 4.2). However, the actual algorithm uses $\Gamma'$ and in the running time analysis we will be using the fact that $\Gamma'$ can be constructed in $O(n \log n)$ time.

**Choosing the representatives:** For every leaf node $t \in V_\Gamma$ we choose the disk center in $C(t)$ as its representative $R_t$. Note that by the construction of $\Gamma$, each $C(t)$ is a singleton set when $t$ is a leaf. For an internal node $t$, pick as representative the disk with smallest order in $D(t)$ (ties are broken using a pre-defined ordering of disks).

**Neighborhood of nodes:** For every interesting node $t$, define its *close neighborhood*

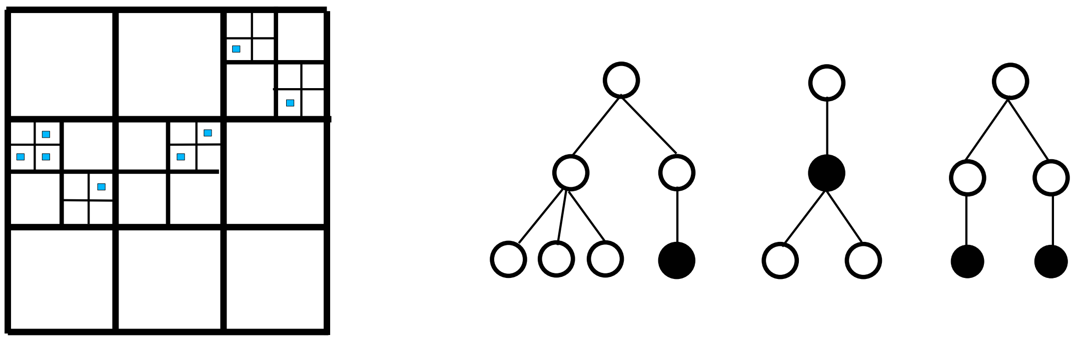

Figure 4: The recursive partitioning procedure for a set of disk centers and the corresponding forest Γ. Assume all the disks have the same radius. The white nodes are the interesting nodes in Γ.

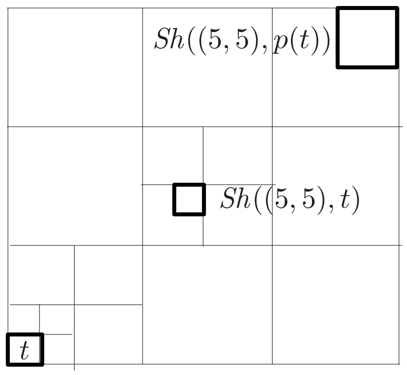

Figure 5: The result of $(\alpha = 5, \beta = 5)$-shift of node $t$ and $p(t)$.

$(N_c(t))$ as the set of all nodes at level $d_t$ which are within a distance of $2^{-d_t}$ from $t$.

We also define a *far neighborhood* $(N_f(t))$ for the nodes in *Roots*. To do so, we introduce some new definitions. For a node $t \in V_\Gamma$ and integers $\alpha, \beta$ define its $(\alpha, \beta)$-shift, $Sh((\alpha, \beta), t)$ with $-4\epsilon^{-1} \leq \alpha, \beta \leq 4\epsilon^{-1}$ and $\max\{|\alpha|, |\beta|\} \geq (2\epsilon)^{-1}$, as the square $t'$ obtained by shifting $t$ by $\epsilon\alpha 2^{-d_t}$ in the $x$-direction and by $\epsilon\beta 2^{-d_t}$ in the $y$-direction, respectively[4] (see Fig. 5). For $k > 1$, define $p^k(t) = p(p^{k-1}(t))$ (so $p^k(t)$ is the $k$th ancestor of $t$). For every $t \in Roots$, construct $O(\epsilon^{-2})$ ordered buckets, where $bucket((\alpha, \beta), t)$ is defined as

$$\begin{aligned} bucket((\alpha,\beta),t) &= \{Sh((\alpha,\beta),t), Sh((\alpha,\beta),p(t)), Sh((\alpha,\beta),p^2(t)), \dots\}, \\ Bucket(t) &= \bigcup_{\alpha,\beta} bucket((\alpha,\beta),t). \end{aligned}$$

Empty squares can be ignored in *Bucket*$(t)$. All nodes in a bucket are also nodes in Γ. Now for every $t \in Roots$ and every $(\alpha, \beta)$-pair do the following: scan through

[4]The max condition ensures that the squares $t$ and $t'$ are not very close to each other.

$bucket((\alpha, \beta), t)$ to find the first[5] node $t' \in bucket((\alpha, \beta), t)$ such that there exists $u \in C(t)$ and $v \in C(t')$ with edge $(u, v)$ in $G$. If no such $t'$ exists, then set $t' = \emptyset$. Add $t'$ to $N_f(t)$.

The idea behind creating the *bucket* is (1) to ensure that for every $(\alpha, \beta)$-pair there exists a straight line passing through the centers of all the nodes of $bucket((\alpha, \beta), t)$ and $t$, and (2) to ensure that disks that intersect any disk centered in $t$, have their centers either close to $t$ or inside a node of $Bucket(t)$. The first fact easily follows from the construction. The second fact is proved by the following lemma. Remember that a point is contained in a set of squares if there exists a square in the set which contains the point.

**Lemma 2.** *Let $u$ be a disk center contained in a node $t \in V_\Gamma$. Then for every edge $(u, v)$ in $G$, $v$ is contained in a node of $Bucket(t) \cup N_c(t)$.*

*Proof.* Let $d(\cdot, \cdot)$ denote the Euclidean metric. If $d(u, v) \leq 2^{-d_t}$ then $v$ is contained in $N_c(t)$. If $2^{-d_t} < d(u, v) \leq 2^{-d_t+1}$ then $v$ is contained in $\bigcup_{\alpha,\beta} Sh((\alpha, \beta), t)$. If $2^{-d_t+1} < d(u, v) \leq 2^{-d_t+2}$ then $v$ is contained in $\bigcup_{\alpha,\beta} Sh((\alpha, \beta), p(t))$. In general, if $2^{-d_t+k} < d(u, v) \leq 2^{-d_t+k+1}$ (for $k = 0, 1, \ldots, d_t$) then $v$ is contained in $\bigcup_{\alpha,\beta} Sh((\alpha, \beta), p^k(t))$. □

**Finding the close neighborhood:** Finding the close neighborhood for nodes at level 0 is straight-forward. Because we work with $\Gamma'$ we construct a *close pseudo-neighborhood* $(N'_c(t))$ for a node $t$. A node $r \in V_{\Gamma'}$ is in $N'_c(t)$ if (1) $r$ belongs to $N_c(t)$, or (2) in $\Gamma$ the node $r$ is the deepest level ancestor of a node $r' \in N_c(t)$ with $r' \notin V_{\Gamma'}$.

Now assuming we have found $N'_c(t)$, we describe the construction of the close pseudo-neighborhood for a child $s$ of $t$ in $\Gamma'$. We access all the nodes in $N'_c(t)$. We do a level-order traversal from these nodes with a modification that the subtree of any node $b$ is accessed only if $d_b \leq d_s$ and $R_b$ is at most a distance of $(1 + \sqrt{2}\epsilon)2^{-d_b}$ away from $R_s$. The nodes at which the traversal ends and whose representatives are at most $(1 + \sqrt{2}\epsilon)2^{-d_s}$ away from $R_s$ define $N'_c(s)$.

Any node $g \in V_{\Gamma'}$ accessed during the traversal has a node $f \in V_\Gamma$ such that (1) $f$ is an ancestor of $s$ in $\Gamma$, (2) $d_g = d_f$, and (3) the distance between $f$ and $g$ is $O(2^{-d_g})$. We charge the access of node $g$ to the node $f$ ($f$ need not be in $\Gamma'$). For this entire procedure we can charge all the accesses of node $g$ to different nodes which are at the same level as $g$ and only $O(2^{-d_g})$ away (note there are only $O(\epsilon^{-2})$ such nodes). This implies that the time for finding the close pseudo-neighborhoods for all nodes is $O(n\epsilon^{-2})$.

**Finding the far neighborhood for small global scale:** If $S(\mathcal{D})$ is *small* (say polynomially bounded, the exact bound on $S(\mathcal{D})$ will be specified later), we use the observation from Gupta *et al.* ([21], Section 2.1) that maps the problem of reporting the intersection of a collection of disks with a query disk into halfspace range searching in two dimensions higher. Agarwal and Matoušek ([3], Theorem 1.1) construct a data structure for $\mathbb{R}^k$ which for any parameter $n \leq m \leq n^{\lfloor k/2 \rfloor}$ and any positive constant $\delta$, after $O(m^{1+\delta})$ space and time preprocessing answers halfspace queries in $\tilde{O}(n/m^{1/\lfloor k/2 \rfloor})$ time, and has $O(m^{1+\delta}/n)$

[5]We view the $bucket((\alpha, \beta), t)$ as an ordered set, therefore the first node is the smallest square in $bucket((\alpha, \beta), t)$ satisfying the condition.

amortized update time[6] . We construct this data structure bottom-up. The data structure for any node $t \in V_{\Gamma'}$ stores the points in $C(t)$. Among the children of $t$, let $s$ be the node having the highest number of points in its data structure. We update the data structure of $s$ to construct the data structure for $t$. This is done by first deleting all points from the data structure that are not in $C(t)$ and then by using siblings of $s$ to insert the remaining points of $C(t)$. We then query the data structure for every disk in $C(t')$ with $t \in \mathit{Bucket}(t')$ to check for intersection.

If $S(\mathcal{D})$ is the global scale factor, then each $\mathit{bucket}((\alpha,\beta),t)$ is of size $O(\log S(\mathcal{D}))$. Therefore, $|\mathit{Bucket}(t)| = O(\epsilon^{-2}\log S(\mathcal{D}))$ and the representation of $\mathit{Bucket}(t)$ in $\Gamma'$ can be found the same way as for the close neighborhood. Each disk acts as a query disk $O(\epsilon^{-2}\log S(\mathcal{D}))$ many times, so the total number of queries is at most $O(n\epsilon^{-2}\log S(\mathcal{D}))$.

To balance the total time for setting up the data structure at every node and the total query time, we assume the parameter $m$ of [3] to be $n^c$ for some $c \geq 1$. As we work in two dimensions higher, each query can be answered in $\tilde{O}(n^{1-c/2})$ time. The total time for answering queries is $\tilde{O}(n^{2-c/2}\epsilon^{-2}\log S(\mathcal{D}))$. The total time for setting up all the data structures of [3] by the procedure described above is $O(n^{c+\delta})$, where $\delta$ is any positive constant. On eliminating $c$, by balancing the query and construction times, we get a space and time bound of $O(n^{4/3+\delta}\epsilon^{-4/3}\log^{2/3} S(\mathcal{D}))$ for finding the far neighborhood of all nodes.

**Finding the far neighborhood for large global scale:** In this case we don't explicitly construct the far neighborhood, but use the adjacency list of $G$ to directly add (to the spanner) the edges added through the far neighborhood. For a node $t \in \mathit{Roots}$, we only consider edges $(u,v)$ of $G$ that satisfy (1) $u \in C(t)$, $v \notin C(t)$, and (2) $r_u \leq r_v$. Let $E_t$ be the set of edges of $G$ that satisfy the above conditions for $t$. For an edge $(u,v) \in E_t$, we find points $v'$ and $v''$ on the line segment connecting $u$ to $v$ such that $2d(u,v') = d(u,v'')$, $v'$ is contained in $\bigcup_{\alpha,\beta} \mathit{Sh}((\alpha,\beta),t)$, and $v''$ is not contained in $\bigcup_{\alpha,\beta} \mathit{Sh}((\alpha,\beta),t)$. Let $\alpha',\beta'$ be such that $v'$ is contained in $\mathit{Sh}((\alpha',\beta'),t)$. We assign the edge $(u,v)$ to $\mathit{bucket}((\alpha',\beta'),t)$. Once all edges in $E_t$ have been assigned into their respective $\mathit{bucket}$, we pick for each $(\alpha,\beta)$-pair the shortest edge in $\mathit{bucket}((\alpha,\beta),t)$ and add it to the spanner. The entire procedure can be implemented in $O(|E|)$ time given the adjacency list of $G$.

**Edges in spanner $\mathbf{G'}$:** For every interesting node $t$ and every $t' \in N_c(t)$, we add an edge between $R_t$ and $R_{t'}$. Additionally, for every node $t \in \mathit{Roots}$, we put an edge of $G$ between a vertex in $C(t)$ and another in $C(t')$ into the spanner, where $t' \in N_f(t)$. The following lemma proves that $G'$ is indeed a sparse subgraph of $G$.

**Lemma 3.** *The graph $G'$ has $O(n\epsilon^{-2})$ edges and is a subgraph of the disk graph $G$.*

*Proof.* The nodes at level $d_t$ in $\Gamma$ are defined by the set of non-empty squares of the $\epsilon 2^{-d_t}$-grid. Thus, the set $N_c(t)$ is of size $O(\epsilon^{-2})$ and for every interesting node we add $O(\epsilon^{-2})$ edges from its representative $R_t$. There is an edge $(R_t, R_{t'})$ with $t' \in N_c(t)$ in $G$ because the Euclidean distance between these two points is at most $(1+\sqrt{2}\epsilon)2^{-d_t}$ and the radii of disks centered at $R_t$ and $R_{t'}$ are at least $2^{-d_t}$.

[6]The use of the dynamic halfspace range query algorithm in the above construction is just for simplicity. Similar results could also be obtained by using a static halfspace range query algorithm.

Also for every $t \in Roots$, we add $O(\epsilon^{-2})$ additional edges and these edges are added only if they are present in $G$. Since there are at most $2n-1$ interesting nodes in $\Gamma$, the result follows. □

## 4.2 Proof of the Spanner Property

Over the next three lemmas we prove that $G'$ is a $(1+\epsilon)$-spanner of the disk graph $G$. In the proofs, we treat $\epsilon$ as a small constant.

**Lemma 4.** *Let $u$ and $v$ be disk centers such that there exists a node $t \in V_\Gamma$ with $u, v \in C(t)$. Then there exists a path in the spanner $G'$ such that $d_{G'}(u,v) \leq (1+c_0\epsilon)d(u,v)$ for some constant $c_0$.*

*Proof.* Firstly note that, since $r_u, r_v \geq 2^{-d_t}$ and length of $t$ is $\epsilon 2^{-d_t}$, we have $d_G(u,v) = d(u,v)$ (i.e., edge $(u,v)$ exists in $G$). The proof is by induction. We inductively assume that Lemma 4 holds for all the finitely many pairs of $(u',v')$ of vertices of $G$ with $d(u',v') < d(u,v)$. In the subtree rooted at $t$, consider the deepest level such that there exist nodes $a$ and $b$ with $u \in C(a)$, $v \in C(b)$ and an edge between $R_a$ and $R_b$ in $G'$. There exists such $a$ and $b$ because interesting nodes with the same parent have an edge between their representatives.

In the subtree rooted at $a$, let $a'$ be the closest descendant of $a$ which is interesting with $u \in C(a')$. In the subtree rooted at $b$, let $b'$ be the closest descendant of $b$ which is interesting with $v \in C(b')$. If both $a'$ and $b'$ don't exist, then $u = R_a$ and $v = R_b$ and the edge $(R_a, R_b) = (u,v)$ exists in $G'$ (no interesting nodes in the subtrees rooted at $a$ and $b$ implies that $u$ and $v$ are the only disk centers contained in $a$ and $b$ respectively because by construction we continue partitioning the grid until each each disk center is contained in a separate square). Therefore, in this case $d_{G'}(u,v) = d_G(u,v) = d(u,v)$.

Lets now assume that both $a'$ and $b'$ exists with $d_{a'} \leq d_{b'}$ (the other two cases where only $a'$ or $b'$ exist could be handled in a similar way, and the case where $d_{a'} \geq d_{b'}$ is symmetric). Let $b''$ be the node at the same level as $a'$ with $v \in C(b'')$. There exists no edge between representatives of $a'$ and $b''$ (because $a$ and $b$ are the deepest level nodes with this property). This implies that $d(u,v) \geq 2^{-d_{a'}}$. Also if not $a'$, at least $p(a')$ contains both $u$ and $R_a$ (this is because $a'$ is closest descendant of $a$ which is interesting). Therefore for some constant $c_1 = 2\sqrt{2}$, we get $d(u,R_a) \leq c_1\epsilon 2^{-d_{a'}}$ and thus $d(u,R_a) \leq c_1\epsilon d(u,v)$. Similarly, $b''$ contains both $v$ and $R_b$ (because $b''$ is the closest descendant of $b$ which is interesting and $d_{b''} = d_{a'} \leq d_{b'}$), therefore $d(R_b,v) \leq \sqrt{2}\epsilon 2^{-d_{b''}} \leq c_1\epsilon 2^{-d_{b''}} = c_1\epsilon 2^{-d_{a'}} \leq c_1\epsilon d(u,v)$.

Since, $d(u,R_a) \leq c_1\epsilon d(u,v) \leq d(u,v)$ and $d(R_b,v) \leq c_1\epsilon d(u,v) \leq d(u,v)$, by the inductive hypothesis, we know $d_{G'}(u,R_a) \leq (1+c_0\epsilon)d(u,R_a)$ and $d_{G'}(R_b,v) \leq (1+c_0\epsilon)d(R_b,v)$. There exists a path in $G'$ from $u$ to $v$ of length $(1+c_0\epsilon)d(u,R_a) + d(R_a,R_b) +$

$(1+c_0\epsilon)d(R_b,v)$. Putting everything together we get

$$\begin{aligned}
d_{G'}(u,v) &\leq d_{G'}(u,R_a)+d_{G'}(R_a,R_b)+d_{G'}(R_b,v)\\
&\leq (1+c_0\epsilon)d(u,R_a)+d(R_a,R_b)+(1+c_0\epsilon)d(R_b,v)\\
&\leq (2+c_0\epsilon)d(u,R_a)+d(u,v)+(2+c_0\epsilon)d(R_b,v) \text{ (as } d(R_a,R_b)\leq d(u,R_a)+d(u,v)+d(R_b,v))\\
&\leq (1+c_0\epsilon)d(u,v).
\end{aligned}$$

The final step follows by substituting the bounds on $d(u,R_a)$ and $d(R_b,v)$ in terms of $d(u,v)$. The constants satisfy $c_1=2\sqrt{2}$ and $c_0\geq 4c_1/(1-2c_1\epsilon)$. □

**Lemma 5.** *Let $(u,v)$ be an edge in the disk graph such that there exist nodes $t,t'\in V_\Gamma$ with $t$ being an interesting node and (disk-centers) $u$ and $v$ contained in $t,t'$ respectively, $t'\in N_c(t)$, and $C(t)\neq\emptyset$ and $C(t')\neq\emptyset$. Then there exists a path in the spanner $G'$ such that $d_{G'}(u,v)\leq(1+c_0\epsilon)d(u,v)$ for some constant $c_0$.*

*Proof.* The proof is by induction. We inductively assume that Lemma 5 holds for all the finitely many pairs of $(u',v')$ of vertices of $G$ with $d(u',v')<d(u,v)$. Since $t'\in N_c(t)$ for interesting node $t$, we know that edge $(R_t,R_{t'})$ exists in $G'$.

The proof is very similar to that of Lemma 4. Consider the deepest level nodes $a,b$ in $\Gamma$, such that (1) $b\in N_c(a)$, (2) $u$ is contained in $a$ and $v$ is contained in $b$, and (3) edge between $R_a$ and $R_b$ is in $G'$. Note that by the assumptions of the lemma, we know that $t$ and $t'$ satisfy all the above three conditions. In the subtree rooted at $a$, let $a'$ be the closest descendant of $a$ which is interesting with $u$ contained in $C(a')$. In the subtree rooted at $b$, let $b'$ be the closest descendant of $b$ which is interesting with $v$ contained in $C(b')$. As in Lemma 4 if both $a'$ and $b'$ doesn't exist then $u=R_a$ and $v=R_b$, and edge $(u,v)$ exists in $G'$ implying that $d_{G'}(u,v)=d(u,v)$.

Lets now assume that both $a'$ and $b'$ exists with $d_{a'}\leq d_{b'}$ (the other two cases where only $a'$ or $b'$ exist could be handled in a similar way, and the case where $d_{a'}\geq d_{b'}$ is symmetric). Let $b''$ be the node at the same level as $a'$ with $v\in C(b'')$. There exists no edge between representatives of $a'$ and $b''$ (because $a$ and $b$ are the deepest level nodes with this property). This implies that $d(u,v)\geq 2^{-d_{a'}}$. Also if not $a'$, at least $p(a')$ contains both $u$ and $R_a$ (this is because $a'$ is closest descendant of $a$ which is interesting). Therefore for some constant $c_1=2\sqrt{2}$, we get $d(u,R_a)\leq c_1\epsilon 2^{-d_{a'}}$ and thus $d(u,R_a)\leq c_1\epsilon d(u,v)$. Similarly, $b''$ contains both $v$ and $R_b$ (because $b''$ is the closest descendant of $b$ which is interesting and $d_{b''}=d_{a'}\leq d_{b'}$), therefore $d(R_b,v)\leq\sqrt{2}\epsilon 2^{-d_{b''}}\leq\epsilon c_1 2^{-d_{b''}}=c_1\epsilon 2^{-d_{a'}}\leq c_1\epsilon d(u,v)$.

Since, $d(u,R_a)\leq c_1\epsilon d(u,v)\leq d(u,v)$ and $d(R_b,v)\leq c_1\epsilon d(u,v)\leq d(u,v)$, by the inductive hypothesis, we know $d_{G'}(u,R_a)\leq(1+c_0\epsilon)d(u,R_a)$ and $d_{G'}(R_b,v)\leq(1+c_0\epsilon)d(R_b,v)$. We get,

$$\begin{aligned}
d_{G'}(u,v) &\leq d_{G'}(u,R_a)+d_{G'}(R_a,R_b)+d_{G'}(R_b,v)\\
&\leq (1+c_0\epsilon)d(u,R_a)+d(R_a,R_b)+(1+c_0\epsilon)d(R_b,v)\\
&\leq (2+c_0\epsilon)d(u,R_a)+d(u,v)+(2+c_0\epsilon)d(R_b,v) \text{ (as } d(R_a,R_b)\leq d(u,R_a)+d(u,v)+d(R_b,v))\\
&\leq (1+c_0\epsilon)d(u,v).
\end{aligned}$$

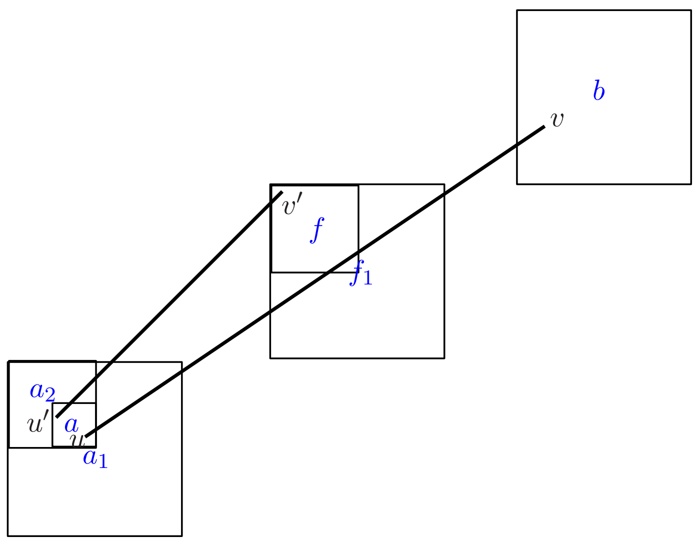

Figure 6: Figure illustrating the proof of Lemma 6. Here, $u, u', v, v'$ are vertices in the graph and $a, a_1, a_2, f, f_1, b$ are the centers of squares.

The final step follows by substituting the bounds on $d(u, R_a)$ and $d(R_b, v)$ in terms of $d(u, v)$ with constant $c_0 \geq 4c_1/(1 - 2c_1\epsilon)$. □

**Lemma 6.** *Let $(u, v)$ be an edge in the disk graph $G$. Then there exists a path in the spanner $G'$ such that $d_{G'}(u, v) \leq (1 + c_2\epsilon)d(u, v)$ for some constant $c_2$.*

*Proof.* The proof is by induction. We inductively assume that Lemma 6 holds for all the finitely many pairs of $(u', v')$ of vertices of $G$ with $d(u', v') < d(u, v)$.

Let $a$ be the smallest square in *Roots* containing $u$, and let $g$ the smallest square in *Roots* containing $v$. Without loss of generality, lets assume $d_a \geq d_g$ (the other case is symmetric). Let $t$ be the node containing $v$ with $d_a = d_t$, so $d_t \geq d_g$. We divide the analysis into two parts based on the position of $t$.

**Case 1:** First consider the case where $t \in N_c(a)$. Since, $a \in Roots$, we know that $C(a) \neq \emptyset$. Also, we know that $C(t) \neq \emptyset$ because if $C(t) = \emptyset$ then there is a square $h$ in *Roots* whose depth is greater than $d_t$ (and hence greater than $d_g$) that contains $v$ (a contradiction to the assumption that $g$ is the smallest square in *Roots* containing $v$). Therefore, $C(a)$ and $C(t)$ are both non-empty and we satisfy the conditions of Lemma 5. From, Lemma 5, we get that $d_{G'}(u, v) \leq (1 + c_0\epsilon)d(u, v)$.

**Case 2:** If $t \notin N_c(a)$, then we know from Lemma 2 that there exists at least one node in *Bucket*$(a)$ containing $v$ (as we are in the case where $v$ is not contained in any node of $N_c(a)$). Let $2^{-l+1} \leq d(u, v) < 2^{-l+2}$. Let $b \in bucket((\alpha', \beta'), a)$ be a node in $\Gamma$ at level (depth) $l$ containing $v$. Since, $a \in Roots$ means that there exists no ancestor $h$ of $a$ with $u \in C(h)$, which implies that $r_u \leq \epsilon 2^{-d_a}$. Now, since $l = d_b \leq d_a$, implies that $r_u \leq \epsilon 2^{-l} \leq 2^{-l}$. Since, $d(u, v) \geq 2^{-l+1}$ and $r_u \leq 2^{-l}$ implies that $r_v \geq 2^{-l}$ (remember that, $d(u, v) \leq r_u + r_v$). Therefore, $v \in C(b)$. See Fig. 6.

Let $f \in bucket((\alpha', \beta'), a) \cap N_f(a)$. Let $(u', v')$ with $u' \in C(a)$ and $v' \in C(f)$ be the edge added to the spanner. The edge $(u, u')$ also exists in $G$ because the radius of $u'$ is at least $\epsilon^{-1}$ times $a$'s length. Since, $d(u, u') \leq \sqrt{2}\epsilon 2^{-d_a} \leq \sqrt{2}\epsilon 2^{-l} \leq d(u, v)$. Therefore, by induction hypothesis we know that $d_{G'}(u, u') \leq (1 + c_2\epsilon)d(u, u')$.

Let $a_1$ and $a_2$ be the ancestors of $a$ at levels $d_b(= l)$ and $d_f$ in $\Gamma$, respectively. Let $f_1$ be the ancestor of $f$ in $\Gamma$ at level $d_b$. In the following description the distance between a point and a square is the distance of the point from the center of the square.

Since $f$ is obtained by shifting $x$- and $y$-coordinates of every point in $a_2$ by $\epsilon\alpha' 2^{-d_{a_2}}$ and $\epsilon\beta' 2^{-d_{a_2}}$ respectively, and since, $\max\{|\alpha'|, |\beta'|\} \geq (2\epsilon)^{-1}$, we have $d(a_2, f) \geq \frac{1}{2\epsilon}\epsilon 2^{-d_{a_2}}$. Now,

$$d(u, v') \geq d(a_2, f) - \sqrt{2}\epsilon 2^{-d_{a_2}} \geq 2^{-d_{a_2}}/2 - \sqrt{2}\epsilon 2^{-d_{a_2}}.$$

On the other hand, $d(u, u') \leq \sqrt{2}\epsilon 2^{-d_{a_2}}$. Therefore, for some constant $c_3$, we have

$$d(u, u') \leq \sqrt{2}\epsilon 2^{-d_{a_2}} \leq c_3\epsilon(2^{-d_{a_2}}/2 - \sqrt{2}\epsilon 2^{-d_{a_2}}) \leq c_3\epsilon d(u, v').$$

In $G'$ for some constant $c_4$ we have,

$$\begin{aligned} d_{G'}(u, v') &\leq d_{G'}(u, u') + d_{G'}(u', v') \\ &\leq (1 + c_2\epsilon)d(u, u') + d(u', u) + d(u, v') \\ &\leq (1 + c_4\epsilon)d(u, v'). \end{aligned}$$

Its enough to set $c_4 = 3c_3$.

The edge $(v', v)$ exists in $G$, as $r_{v'} \geq r_u$ (this follows because $v' \in C(f)$, but $u \notin C(a_2)$) and there is a straight line connecting the squares $a_1$,$f_1$, and $b$. Since $d(v', v) \leq d(u, v)$, by the inductive hypothesis we get $d_{G'}(v', v) \leq (1 + c_2\epsilon)d_{G'}(v', v)$ (as $d(v', v) \leq d(u, v)$). Hence (for $c_2 \geq c_4$),

$$\begin{aligned} d_{G'}(u, v) &\leq d_{G'}(u, v') + d_{G'}(v', v) \\ &\leq (1 + c_4\epsilon)d(u, v') + (1 + c_2\epsilon)d(v', v) \\ &= (1 + c_2\epsilon)(d(u, v') + d(v', v)) \\ &\leq (1 + c_2\epsilon)d(u, v). \end{aligned}$$

□

**Theorem 1.** *Let $G$ be a disk graph defined on $\mathcal{D}$. A $(1 + \epsilon)$-spanner of $G$ with $O(n\epsilon^{-2})$ edges can be constructed in $O(\min\{n^{4/3+\delta}\epsilon^{-4/3}\log^{2/3} S(\mathcal{D}), Adj(\mathcal{D}) + n\epsilon^{-2}\})$ time, where $\delta$ is any positive constant, and $Adj(\mathcal{D})$ is the time for constructing the adjacency list for $G$.*

### 4.3 Extension to Ball Graphs

The quadtrees become $2^k$-trees in $\mathbb{R}^k$. The close neighborhood and far neighborhood of any node are of size $O(\epsilon^{-k})$. Therefore, the total number of edges is $O(n\epsilon^{-k})$. In case of small global scale using the result of range-searching from [3] we get a running time of $O(n^{(2/\ell)+\delta}\epsilon^{-k/\ell}\log^{1/\ell} S(\mathcal{D}))$, where $\ell = 1 + 1/(\lfloor k/2 \rfloor + 1)$. In case of large global scale the running time is $O(|E| + n\epsilon^{-k})$ given the adjacency list of $G$. Using these observations, the following result is immediate:

**Theorem 2.** *Let $G$ be a $k$-dimensional ball graph defined on $\mathcal{D}$. A $(1+\epsilon)$-spanner of $G$ with $O(n\epsilon^{-k})$ edges can be constructed in $O(\min\{n^{(2/\ell)+\delta}\epsilon^{-k/\ell}\log^{1/\ell} S(\mathcal{D}), Adj(\mathcal{D})+n\epsilon^{-k}\})$ time, where $\delta$ is any positive constant, $\ell = 1+1/(\lfloor k/2\rfloor+1)$, and $Adj(\mathcal{D})$ is the time for constructing the adjacency list for $G$.*

### 4.4 Spanners for Unit Ball Graphs

If $G$ was defined on unit balls (disks) we can speed up the construction considerably. The algorithm remains the same until the part where we compute the far neighborhood. Here the set *Roots* contains only the non-empty squares of the $\epsilon$-grid. For finding the far neighborhood of nodes in *Roots*, we solve a collection of bichromatic closest pair problems. If the disks corresponding to the closest pair intersect, we add the corresponding edge into the spanner. See Fig. 7.

We now state a general result describing the upper bounds of the spanner construction in terms of the upper bounds for the bichromatic closest pair problem. We first define the bichromatic closest pair problem.

**Problem 1** (Bichromatic Closest Pair Problem)**.** *Given a set of $\lambda_1$ red points and $\lambda_2$ blue points in $\mathbb{R}^k$, find a red point $p$ and a blue point $q$ such that the Euclidean distance between $p$ and $q$ is minimum among all red-blue pairs.*

In $\mathbb{R}^k$ ($k > 2$), the currently best algorithm of Agarwal *et al.* [2] for finding a bichromatic closest pair between points sets $\mathcal{A}$ (with $\lambda_1 = |\mathcal{A}|$) and $\mathcal{B}$ (with $\lambda_2 = |\mathcal{B}|$) runs in $O((\lambda_1\lambda_2\log\lambda_1\log\lambda_2)^{2/3} + \lambda_1\log^2\lambda_2 + \lambda_2\log^2\lambda_1)$ expected time for $k = 3$ and $O((\lambda_1\lambda_2)^{1-1/(\lceil k/2\rceil+1)+\delta} + \lambda_1\log\lambda_2 + \lambda_2\log\lambda_1)$ expected time for $k \geq 4$, where $\delta$ is any positive constant. We assume in the rest of our discussion that for some $c > 1$, $n^c f(n)$ is an upper bound on the expected time for computing a bichromatic closest pair for a total of $n$ points in $\mathbb{R}^k$ ($k$ constant), where $\log f(n) = o(\log n)$ (i.e., $f$ grows slower than polynomial).

**Lemma 7.** *For unit ball graphs in $\mathbb{R}^k$ ($k > 2$), the worst case running time for finding the far neighborhood for all nodes in Roots is $O(n^c f(n)\epsilon^{-k})$.*

*Proof.* For every node $t$ in *Roots*, we solve the bichromatic closest pair problem between $C(t)$ and $C(t')$ for $t' \in$ *Roots* and $|C(t')| \leq |C(t)|$. Since $t$ participates in $O(\epsilon^{-k})$ bichromatic closest pair problems, the cost of solving these problems (charged to $t$) is $O(|C(t)|^c f(|C(t)|)\epsilon^{-k})$. Since every disk center has at most one node in *Roots* (say $t$) such that the disk center is in $C(t)$, the total cost of the procedure is at most:

$$O\left(\sum_{t\in Roots} |C(t)|^c f(|C(t)|)\epsilon^{-k}\right) \leq O(n^c f(n)\epsilon^{-k}).$$

□

**Theorem 3.** *Let $G$ be a unit ball graph in $\mathbb{R}^k$ ($k > 2$). A $(1+\epsilon)$-spanner of $G$ with $O(n\epsilon^{-k})$ edges can be constructed in $O(n^c f(n)\epsilon^{-k})$ expected time.*

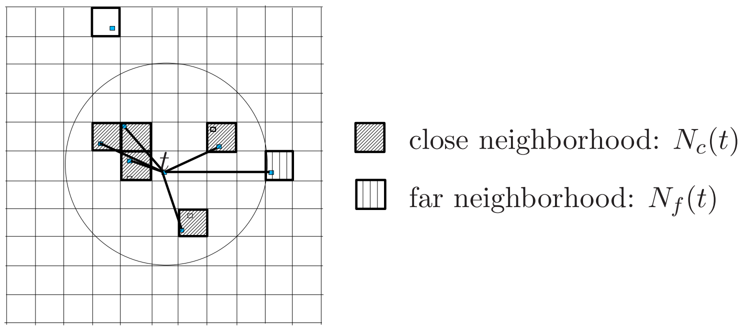

Figure 7: The close and far neighborhood of a node $t$ in *Roots* for a unit disk graph. All the squares within the circle are at most $2^{-d_t}$ distance away from $t$. Far neighborhood is determined by using bichromatic closest pair tests.

It is well known that the bichromatic closest pair problem in $\mathbb{R}^2$ can be solved in $O(n \log n)$ by using post-office queries [41]. Lemma 7 can also be proved for $k = 2$ by setting $c = 1$ and $f(n) = \log n$.

**Corollary 1.** *A $(1+\epsilon)$-spanner $G'$ of a unit ball graph $G$ can be constructed in $O(n\epsilon^{-2} \log n)$ time for $k = 2$, in $\tilde{O}(n^{4/3}\epsilon^{-3})$ expected time for $k = 3$, and in $O(n^{2-2/(\lceil k/2 \rceil+1)+\delta}\epsilon^{-k})$ expected time for $k \geq 4$, where $\delta$ is any positive constant.*

### 4.4.1 Complexity of Unit Ball Spanner $\equiv$ Complexity of Euclidean Minimum Spanning Tree

Theorem 3 shows that constructing a spanner of a unit ball graph is not harder than computing a bichromatic closest pair for $n$ points in $\mathbb{R}^k$. We now argue that the complexities of these two problems are in fact equivalent. To make this relation more precise, define the *time exponent* of a problem $P$ with respect to input size $n$ to be

$$\tau_P = \inf\{a \mid \exists \text{ a randomized algorithm solving } P \text{ with an expected running time of } O(n^a)\}.$$

Of similar interest could be the deterministic time exponent defined with the restriction to deterministic algorithms. We use the current definition because better randomized algorithms are known for some of the problems considered. Before we can state our equivalence we need to define two other problems.

**Problem 2** (Euclidean Minimum Spanning Tree Problem)**.** *Given a set $\mathcal{S}$ of $n$ points in $\mathbb{R}^k$, a Euclidean minimum spanning tree is a spanning tree whose edges have total minimum length among all spanning trees of $\mathcal{S}$, where the length of an edge is the Euclidean distance between its vertices.*

**Problem 3** (Spanning Forest of Unit Balls Problem)**.** *Given a set $\mathcal{S}$ of $n$ unit balls in $\mathbb{R}^k$, construct a spanning forest of the balls graph defined by $\mathcal{S}$, i.e., construct a spanning tree of each connected component.*

Let $\tau_S$ and $\tau_F$ be time exponents for constructing a $(1+\epsilon)$-spanner and a spanning forest for a set of $n$ unit balls in $\mathbb{R}^k$, respectively. Let $\tau_B$ and $\tau_E$ be the time exponents for the bichromatic closest pair and Euclidean minimum spanning tree problems for $n$ points in $\mathbb{R}^k$, respectively. Since the bichromatic closest pair and Euclidean minimum spanning tree problems have the same asymptotic complexities [2, 23], their time exponents are equal (i.e., $\tau_B = \tau_E$).

Equality of $\tau_F$ and $\tau_E$ has been shown by the following theorem.

**Theorem 4** (Eppstein [11], Theorem 4.2)**.** *The best possible randomized expected time bounds for constructing a spanning forest of a set of unit balls in $\mathbb{R}^k$ and a Euclidean minimum spanning tree in $\mathbb{R}^k$ differ by constant factors only.*

**Theorem 5.** *The time exponents $\tau_S$, $\tau_B$, $\tau_E$, and $\tau_F$ are all equal.*

*Proof.* We have just seen that $\tau_B = \tau_E = \tau_F$. From Theorem 3, we know that $\tau_S \le \tau_B$. To complete the proof we show that $\tau_F \le \tau_S$. Once we have a $(1+\epsilon)$-spanner any graph traversal (which can be done in linear time) can be used to construct the spanning forest of unit balls (i.e, $\tau_F \le \tau_S$). Putting everything together, we get that $\tau_S = \tau_B = \tau_E = \tau_F$ (all the four problems have the same time exponents). $\square$

The above theorem implies that the time exponent for deciding whether a unit ball graph is connected is the same as that of constructing a spanner for the unit ball graph.

## 5 Separators in Spanner Graph

Let $f$ be a function of $n$ and let $\alpha < 1$ be a positive real. A subset of vertices $T$ of a graph $G$ with $n$ vertices is an $f(n)$ vertex separator that $\alpha$-splits if $|T| \le f(n)$ and the vertices of $G \backslash T$ can be partitioned into two sets $V_1$ and $V_2$ such that there are no edges from $V_1$ to $V_2$, and $\max\{|V_1|, |V_2|\} \le \alpha n$. An $f(s)$-separator decomposition of $G$ is a recursive decomposition of $G$ using separators, where subgraphs of size $s$ have separators of size $O(f(s))$. In this section we show that the spanner graph $G'$ in $\mathbb{R}^k$ has an $O(n^{1-1/k}\epsilon^{-k+1/2} + \epsilon^{-2k+1}\log S(\mathcal{D}))$ vertex separator, where $S(\mathcal{D})$ is the global scale factor (ratio of radii of largest and smallest balls). Furthermore, the entire separator decomposition can be constructed in $\tilde{O}(n)$ time. As usual we describe our procedure for the 2-dimensional case.

We use recursive partitions of rectangles. As before the left bottom corner of the bounding box is assumed to be the origin. Let $\mathcal{X}$ be the sorted list of $x$-coordinates of vertices. Let $\mathcal{X}(\mathcal{R})$ be the sorted list of $x$-coordinates of vertices contained in rectangle $\mathcal{R}$. Similarly define $\mathcal{Y}$ and $\mathcal{Y}(\mathcal{R})$ for $y$-coordinates. We say a vertex crosses a line segment if any edge incident on it in $G'$ crosses the line segment. During each step, the algorithm focuses on one rectangle, called the *active rectangle*. An active rectangle $\mathcal{R}$ has at least $2/3$ of all vertices inside it and there exists a set of $O(\sqrt{n}\epsilon^{-3/2} + \epsilon^{-3}\log S(\mathcal{D}))$ vertices which when removed ensures that no remaining vertex has an edge in $G'$ that crosses the boundary of $\mathcal{R}$.

At every step, the algorithm uses two line segments (called *double line separator*) to divide the currently active rectangle. A *horizontal double line separator* of an active rectangle is a set of at most two horizontal line segments that partitions the active rectangle such that there exists a set of $O(\sqrt{n}\epsilon^{-3/2} + \epsilon^{-3} \log S(\mathcal{D}))$ vertices which when removed ensures that no remaining vertex crosses either of the horizontal line segments (similarly define *vertical double line separator*). Our algorithm recursively partitions an active rectangle alternatively with a horizontal or a vertical double line separator and stops when none of the new rectangles created contain enough vertices to become active. The initial active rectangle is the bounding box.

We maintain sorted doubly linked lists for both the $x$- and $y$-coordinates with pointers between elements representing the same point in the two lists. Using this when moving to an active rectangle $\mathcal{R}$ from the previous active rectangle $\mathcal{R}_p$, the lists $\mathcal{X}(\mathcal{R})$ and $\mathcal{Y}(\mathcal{R})$ can be constructed in time proportional to the number of disk centers contained in $\mathcal{R}_p$ but not contained in $\mathcal{R}$.

**Data Structure for double line separators:** We use the following data structure to efficiently construct double line separators. Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$. We partition the $x$-axis between $x_1$ and $x_n$ into intervals where every interval has at least $\sqrt{n}\epsilon^{-3/2}$ and at most $3\sqrt{n}\epsilon^{-3/2}$ consecutive points in $\mathcal{X}$. The length of the interval $[x_a, x_b]$ is the difference between the coordinates of its first and last element, i.e., $x_b - x_a$. We maintain a balanced binary search tree $Int_{\mathrm{x}}$ where leaves represent these intervals and each internal node stores a splitting value between its left and right subtree. Additionally, with each node we store the length of the longest interval in its subtree. Similarly, we construct a tree for $\mathcal{Y}$.

For updating the intervals when we go from $\mathcal{R}_p$ to $\mathcal{R}$ we use two operations: *merge* and *split plus merge*. When an interval becomes smaller than $\sqrt{n}\epsilon^{-3/2}$ we merge it with its left or right interval whichever is smaller than $2\sqrt{n}\epsilon^{-3/2}$. If neither satisfies the condition we split one of them into two halves and then do the merge. Both of these operations and the subsequent updates to the tree can be performed in $O(\log n)$ time and are only done at most $n$ times.

**Constructing double line separator for level $l$:** Consider the compressed forest $\Gamma'$ constructed in Section 4.1. Let $\mathcal{R}$ be the active rectangle when level $l$ of $\Gamma'$ is considered. We divide $\mathcal{R}$ either using a vertical or a horizontal double line separator. Assume without loss of generality that we are dividing $\mathcal{R}$ using a vertical double line separator. We construct the vertical double line separator (line segments $L_1$ and $L_2$) for $\mathcal{R}$ as follows. Let $x_m$ be the median of the elements in $\mathcal{X}(\mathcal{R})$. We first find the leaf node $p$ in the tree $Int_{\mathrm{x}}$ whose interval contains the median. If that leaf node represents an interval with length greater than $2^{-l+3}$, then define $L_1 = L_2$ as the segment of the vertical line contained in $\mathcal{R}$ and passing through $x_m$. Otherwise, to construct $L_1$ we walk up $Int_{\mathrm{x}}$ starting from node $p$ until we find the first left ancestor[7] ($anc_l$) having an interval of length greater than $2^{-l+3}$ in its subtree. We access the rightmost leaf in the left subtree of $anc_l$ corresponding to an interval $[x_a, x_b]$ with $x_b - x_a \geq 2^{-l+3}$. Consider a vertical line ($x = x_b - 2^{-l+2}$) to the left of $x_b$ at a distance of $2^{-l+2}$. Let $L_1$ be the segment of the vertical line contained in $\mathcal{R}$. If $anc_l$ doesn't exist then we define $L_1$ as the left vertical boundary of $\mathcal{R}$. To construct $L_2$, we walk up

[7] A left (right) ancestor of a node $a$ is any node $b$ in the tree such that $a$ is in the right (left) subtree of $b$.

$Int_{\mathrm{x}}$ starting from node $p$ until we find the first right ancestor ($anc_r$) having an interval of length greater than $2^{-l+3}$ in its subtree. We access the leftmost leaf in the right subtree of $anc_r$ corresponding to an interval $[x_c, x_d]$ with $x_d - x_c \geq 2^{-l+3}$. Consider a vertical line ($x = x_d - 2^{-l+2}$) to the left of $x_d$ at a distance of $2^{-l+2}$. Let $L_2$ be the segment of the vertical line contained in $\mathcal{R}$. If $anc_r$ doesn't exist then we define $L_2$ as the right vertical boundary of $\mathcal{R}$.

Over the next three lemmas we show that $L_1$ and $L_2$ satisfy the properties of vertical double line separator. For the proof, we partition the edges in $G'$ into two sets: let $E_o$ be the set of edges which were added to $G'$ when the nodes in *Roots* were considered, and $E_i$ be the remaining edges in $G'$. The edges in $E_o$ are typically the long edges, whereas the edges in $E_i$ are the short ones. Let $\mathcal{D}(l)$ be the set of disk centers (vertices) to which edges are added when nodes of level $l$ are considered in $\Gamma'$. Every edge in $G'$ is *attributed* to the disk of larger radius, i.e., an edge $(u, v)$ in $G'$ is attributed to $u$ if $r_u \geq r_v$. The general idea is that edges from the set $E_i$ are separated from inside of $\mathcal{R}$ and edges from the set $E_o$ are separated from outside of $\mathcal{R}$.

**Lemma 8.** *Let $\mathcal{R}$ be an active rectangle of level $l$. Then there exists a set of $O(\sqrt{n}\epsilon^{-3/2})$ vertices which when removed ensures that no remaining edge attributed to a vertex in $\bigcup_{l' \geq l} \mathcal{D}(l')$ cross either $L_1$ or $L_2$.*

*Proof.* We will prove the lemma for $L_1$, the case for $L_2$ is symmetric. The edges attributed to vertices of $\bigcup_{l' \geq l} \mathcal{D}(l')$ have lengths at most $2^{-l+2}$. Therefore, any edge attributed to $\bigcup_{l' \geq l} \mathcal{D}(l')$ and crossing $L_1$ must have both its endpoints within a distance of $2^{-l+2}$ from $L_1$. By construction, we guarantee the existence of such a set of $O(\sqrt{n}\epsilon^{-3/2})$ vertices. $\square$

**Lemma 9.** *The distance between $L_1$ and $L_2$ is less than $2^{-l+3}\sqrt{n}\epsilon^{3/2}$. The lengths of $L_1$ and $L_2$ for $l \geq 1$ are at most $2^{-l_p+3}\epsilon^{3/2}\sqrt{n}$, where $l_p$ is the level preceding $l$ in $\Gamma'$.*

*Proof.* Let $I_1, I_2, \ldots, I_z$ be the intervals represented by the leafs of the interval tree. Let $I_y$ be the interval ($1 \leq y \leq z$) containing the median $x_m$. If the length of interval $I_y$ is greater than $2^{-l+3}$ then $L_1 = L_2$ and the distance between $L_1$ and $L_2$ is 0. So we now assume that length of interval $I_y$ is less than $2^{-l+3}$. Let us assume that $L_1$ passes through interval $I_x$ and $L_2$ passes though interval $I_{x'}$ ($x < y < x'$). All the intervals $I_{x+1}, \ldots, I_{x'-1}$ have length at most $2^{-l+3}$. Also all the intervals $I_1, I_2, \ldots, I_z$ contain at least $\sqrt{n}\epsilon^{-3/2}$ disk centers. As there are only $n$ disks, $x' - x - 2 \leq (n - 2\sqrt{n}\epsilon^{-3/2})/(\sqrt{n}\epsilon^{-3/2})$. Hence, the distance between $L_1$ and $L_2$ is at most $(\sqrt{n}\epsilon^{3/2} - 2)2^{-l+3} + 2^{-l+3} \leq 2^{-l+3}\sqrt{n}\epsilon^{3/2}$.

Note that the above proof also holds if $L_1$ and/or $L_2$ are the boundaries of $\mathcal{R}$ (the cases where $anc_l$ and/or $anc_r$ do not exist). For example, if $L_1$ and $L_2$ are the left and right boundaries of $\mathcal{R}$ then all intervals $I_1, \ldots, I_z$ are of lengths at most $2^{-l+3}$ and since each interval contains at least $\sqrt{n}\epsilon^{-3/2}$ disk centers, therefore the distance between $L_1$ and $L_2$ is at most $n/(\sqrt{n}\epsilon^{-3/2}) \cdot 2^{-l+3} \leq 2^{-l+3}\sqrt{n}\epsilon^{3/2}$. Similarly, one can argue the cases where only one of $L_1$ or $L_2$ is the boundary of $\mathcal{R}$.

Let $l_p$ be the level preceding $l$ in $\Gamma'$. By using the same arguments, we can show that the distance between the two horizontal line segments forming the horizontal double line separator level $l_p$ is at most $2^{-l_p+3}\epsilon^{3/2}\sqrt{n}$. This implies that the lengths of $L_1$ and $L_2$ are at most $2^{-l_p+3}\epsilon^{3/2}\sqrt{n}$. □

**Lemma 10.** *Let $\mathcal{R}$ be an active rectangle at level $l$. Then the line segments $L_1$ and $L_2$ define a vertical double line separator for $\mathcal{R}$.*

*Proof.* We work with $L_1$. The case for $L_2$ is similar. From Lemma 8, we know that we are done if $l = 0$. Fix any level $l' < l$. Each node (square) in level $l'$ has an area of $(\epsilon 2^{-l'})^2$. Any edge attributed to a vertex in $\mathcal{D}(l')$ has length at most $2^{-l'+2}$. From Lemma 9, we know that the length of $L_1$ is less than $2^{-l_p+3}\sqrt{n}\epsilon^{3/2}$.

In $\mathcal{R}$ consider two vertical line segments drawn to the left and right of $L_1$ at a distance of $2^{-l'+2}$. Consider the rectangular region within $\mathcal{R}$, between these two vertical line segments. The area of this region is at most $2^{-l_p+3}2^{-l'+3}\epsilon^{3/2}\sqrt{n}$. Each edge in $E_i$ that crosses $L_1$ and is attributed to a vertex in $\mathcal{D}(l')$ will have at least one endpoint in this rectangular region. The number of nodes of level $l'$ in this region is at most $2^{l'-l_p+6}\sqrt{n}\epsilon^{-1/2}$. Since each node contributes at most $\epsilon^{-1}$ vertices (disk centers) to $\mathcal{D}(l')$, we get that the number of vertices of $\mathcal{D}(l')$ present in this region is at most $2^{l'-l_p+6}\sqrt{n}\epsilon^{-3/2}$. This forms a decreasing geometric series in $l'$. For levels $l''$ which are more than $O(\log n)$ below $l_p$, only a constant number of disks of $\mathcal{D}(l'')$ cross $L_1$. Our method of picking representatives ensures that they are the same over all such $l''$. Summing over all $l' \leq l_p$, we get that there exist

$$\sum_{0\leq l'\leq l_p} 2^{l'-l_p+6}\sqrt{n}\epsilon^{-3/2} = \mathrm{O}(\sqrt{n}\epsilon^{-3/2})$$

vertices which when removed ensure that no edge in $E_i$ attributed to a vertex in $\bigcup_{l'<l}\mathcal{D}(l')$ crosses $L_1$.

To finish the proof we consider edges from the set $E_o$ that cross $L_1$ and are attributed to the vertices in $\mathcal{D}(l')$. Since nodes of *Roots* appear only in the first $\log S(\mathcal{D})$ levels of $\Gamma'$, we only consider the case when $l' \leq \log S(\mathcal{D})$. Consider four lines, two vertical lines drawn at distance $2^{-l'+2}$ from $L_1$ on both sides and two horizontal lines drawn above and below $L_1$ at a distance of $2^{-l'+2}$ from its end points. The rectangular region $\mathcal{R}_{l'}$ formed by these four lines has an area at most $2^{-l'+3}(2^{-l_p+3}\sqrt{n}\epsilon^{3/2} + 2^{-l'+3})$. Each edge in $E_o$ that crosses $L_1$ and is attributed to a vertex in $\mathcal{D}(l')$ will have both its end points in $\mathcal{R}_{l'}$. Since each node contributes at most $\epsilon^{-1}$ vertices to $\mathcal{D}(l')$, we get that the number of vertices of $\mathcal{D}(l')$ present in $\mathcal{R}_{l'}$ is at most $2^{l'-l_p+6}\sqrt{n}\epsilon^{-3/2} + 64\epsilon^{-3}$. Summing over all $l' \leq \min\{l_p, \log S(\mathcal{D})\}$, we get that the number of vertices in $\bigcup_{l'<l}\mathcal{D}(l')$ to which edges belonging to $E_o$ and crossing $L_1$ are attributed to, is at most

$$\sum_{0\leq l'\leq \min\{l_p,\log S(\mathcal{D})\}} (2^{l'-l_p+6}\sqrt{n}\epsilon^{-3/2} + 64\epsilon^{-3}) = O(\sqrt{n}\epsilon^{-3/2} + \epsilon^{-3}\log S(\mathcal{D})).$$

Therefore, adding this to the result of Lemma 8, we know that there exist $O(\sqrt{n}\epsilon^{-3/2} + \epsilon^{-3}\log S(\mathcal{D}))$ vertices such that every edge crossing $L_1$ is incident on one of them. □

**Final shape of the separator:** $L_1$ and $L_2$ divide $\mathcal{R}$ into at most 3 rectangles, of which only the rectangle between $L_1$ and $L_2$ could be active (by construction of double line separators).

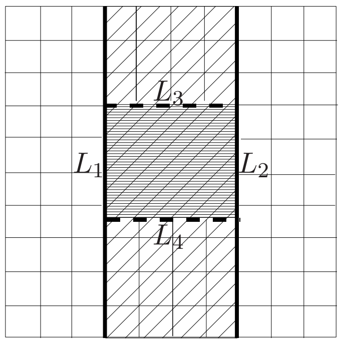

Figure 8: The line segments $L_1$, $L_2, L_3$, and $L_4$ are as defined by the algorithm. The shaded region between $L_1$ and $L_2$ represents the rectangle after the step in which $L_1$ and $L_2$ are defined. If needed in the next step we further divide this shaded region using horizontal double line separators $L_3$ and $L_4$.

If this rectangle $\mathcal{R}_b$ is active, we repeat the same procedure with $\mathcal{Y}(\mathcal{R}_b)$ to find horizontal line separators $L_3$ and $L_4$. The active rectangle thus formed is associated with the level following $l$ in $\Gamma'$ (some levels of $\Gamma$ might be skipped in $\Gamma'$). See Fig. 8.

The algorithm terminates when the partition of an active rectangle produces rectangles none of which are active. Consider the deepest level $l_d$ in $\Gamma'$. Let us assume that the algorithm has not terminated before reaching level $l_d$. Let $\mathcal{R}_d$ be the active rectangle at level $l_d$. Let us assume without loss of generality that a vertical double line separator is used for partitioning $\mathcal{R}_d$. At this level, each disk center is in a separate square of length $\epsilon 2^{-l_d}$. Let $x_m$ be the median of elements in $\mathcal{X}(\mathcal{R}_d)$. Its quite easy to see that in this case, $L_1 = L_2$ as the interval in $Int_{\text{x}}$ containing $x_m$ has length at least $\sqrt{n}\epsilon^{-3/2} \times \epsilon 2^{-l_d} \geq 2^{-l_d+3}$. Therefore, $\mathcal{R}_d$ is partitioned into two rectangles none of which could have more than $n/2$ vertices, hence neither of them could be active (remember that an active rectangle needs to have at least $2n/3$ vertices). Therefore, the algorithm always terminates.

Among the rectangles created by partitioning the last active rectangle, the rectangle $\mathcal{R}_f$ containing the largest number of disk centers forms one separated component. Since the last active rectangle was partitioned by double line separators, we know there exists a set of $O(\sqrt{n}\epsilon^{-3/2} + \epsilon^{-3} \log S(\mathcal{D}))$ vertices to which all edges crossing $\mathcal{R}_f$ are incident on. This set of vertices forms the vertex separator. Also the last active rectangle had at least $2/3$ of the vertices and it gets divided into at most 3 rectangles, implying that $\mathcal{R}_f$ definitely contains at least $2/9$ of the vertices of $G'$.

The algorithm also extends to $k$ dimensions and we get an $O(n^{1-1/k}\epsilon^{-k+1/2} + \epsilon^{-2k+1} \log S(\mathcal{D}))$ separator. The rectangle gets replaced by a $k$-dimensional box and line segments are $(k-1)$-dimensional hyperplanes. The algorithm recursively partitions along every dimension.

We now summarize the main result of this section in the following theorem.

**Theorem 6.** *Let $G$ be a $k$-dimensional ball graph defined on $\mathcal{D}$ and $G'$ be a $(1+\epsilon)$-spanner of $G$ constructed as described above. An $O(n^{1-1/k}\epsilon^{-k+1/2} + \epsilon^{-2k+1} \log S(\mathcal{D}))$ vertex separator* *decomposition of $G'$ with 7/9-split can be found in $\tilde{O}(n)$ time.*

**Complete Euclidean graphs in $\mathbb{R}^2$.** As mentioned earlier, for a complete Euclidean graph in $\mathbb{R}^2$, Abam and Har-Peled [1] presented a construction of a $(1+\epsilon)$-spanner with $O(n\epsilon^{-3})$ edges, maximum degree of $O(\epsilon^{-3}\log^2 n)$, and a separator of size $O(\sqrt{n}\epsilon^{-2})$. Their construction takes $O(n\epsilon^{-2}\log^2 n)$ time. Since complete Euclidean graphs are just a special case of (unit) ball graphs, using Corollary 1 and Theorem 6, we get the following result.

**Corollary 2.** *For any $\epsilon > 0$ and any set of $n$ points in $\mathbb{R}^2$, there is a $(1+\epsilon)$-spanner $G'$ with $O(n\epsilon^{-2})$ edges and a separator of size $O(\sqrt{n}\epsilon^{-3/2})$. The $(1+\epsilon)$-spanner $G'$ can be constructed in $O(n\epsilon^{-2}\log n)$ time.*

## 6 Approximate Distance Queries

Gao and Zhang [17] discuss many approximate proximity problems for unit disk graphs. Using a well-separated pair decomposition, they show that a unit disk graph can be preprocessed in $O(n\sqrt{n\log n}\epsilon^{-3})$ time, such that subsequent distance queries can be answered with $(1+\epsilon)$-stretch in constant time.

We note that other than the standard advantages of using a sparse spanner for solving approximate proximity problems, the separator helps us to also support fast answering of distance queries in ball graphs. First, we define some notations that will be used throughout this section. For a set $U$ of vertices in $G$, we use $Ne(U)$ to denote the neighborhood of $U$, i.e.,

$$Ne(U) = \{v \in V \mid \exists u \in U \text{ such that } (u,v) \in E\}.$$

We use a rooted binary tree $T_G$ to represent a separator decomposition of a graph $G = (V,E)$. Each node $s \in T_G$ is labeled by two subsets of vertices $V(s) \subseteq V$ and $Sep(s) \subseteq V(s)$. Let $G(s) = (V(s), E(s))$ denote the subgraph induced by $V(s)$. Then $Sep(s)$ is a separator in $G(s)$. The root $r \in T_G$ has $V(r) = V$ and $Sep(r)$ is a separator in $G$. For any $s \in T_G$, the labels of its children $s_0, s_1$ are defined as follows: let $V_1 \subset V(s)$ and $V_2 \subset V(s)$ be the components separated by $Sep(s)$ in $G(s)$. Then

$$V(s_0) = V_1 \cup (Sep(s) \cap Ne(V_1)) \quad \text{and} \quad V(s_1) = V_2 \cup (Sep(s) \cap Ne(V_2)).$$

Following previous notation, let $G'$ represent the spanner graph and $\Gamma$ represent the forest from the recursive partitioning of $\epsilon$-grid. We use $m$ to denote the number of edges in $G$. For a node $t$, denote by $p_s(t)$ the parent of $t$ in $T_{G'}$.

### 6.1 Distance Query Algorithm

The algorithm Dist-query (Fig. 9) is similar to the one used by Arikati *et al.* [5] for answering distance queries in planar graphs. The algorithm produces an estimate $\delta(u,v)$ for the distance between vertices $u$ and $v$.

Let $\textsc{Lca}(t_1, t_2)$ denote the least common ancestor of nodes $t_1, t_2$ in $T_{G'}$. The shortest path need not necessarily stay inside $V(\textsc{Lca}(t_1, t_2))$. For this reason we also look at the

```
Algorithm Dist-query($G, \widehat{G}, u, v$)

Preprocessing: Let $G'$ be a $(1+\epsilon)$-spanner of $G$
1. Compute $T_{G'}$ the separator decomposition tree for $G'$
2. For each node $s \in T_{G'}$ do
    (a) $H(s) \leftarrow$ graph induced by $V(s)$ on $\widehat{G}$
    (b) From each node in $Sep(s)$ do a single-source shortest path (Sssp) computation on $\widehat{G}$
Reporting distance between $\mathbf{u}, \mathbf{v} \in \mathbf{V}$:
3. If edge $(u, v)$ exists, $\delta(u, v) \leftarrow d_G(u, v)$
4. Else
    (a) $\delta(u, v) \leftarrow \infty$
    (b) Let $t_1$ and $t_2$ be the leaves in $T_{G'}$ such that $u \in V(t_1)$ and $v \in V(t_2)$
    (c) $t \leftarrow$Lca$(t_1, t_2)$, i.e., least common ancestor of $t_1, t_2$ in $T_{G'}$
    (d) While $t$ is not the root of $T_{G'}$ do
        (i) $\delta(u, v) \leftarrow \min\{\delta(u, v), \min_{z \in Sep(t)}\{d_G(u, z) + d_G(z, v)\}\}$
        (ii) set $t = p_s(t)$
5. Report $\delta(u, v)$
```

Figure 9: Algorithm for answering distance query between $u$ and $v$ in $G$.

ancestor subgraph of $V(\text{Lca}(t_1, t_2))$. The observation is that in such a case the shortest path has to pass through some separator vertex of these subgraphs. Using the algorithm of Schieber and Vishkin [35], the least common ancestor (Lca) queries can be answered in $O(1)$ time after linear time preprocessing. For the Step 4d, the distances used are available as they are already precomputed in the preprocessing Step 2b.

In the next two subsections, we analyze two variants of this query algorithm, which mainly differ only in the argument $\widehat{G}$ passed to the algorithm Dist-query. For answering distance queries with a $(1+\epsilon)$-approximation we use the spanner graph $G'$ for $\widehat{G}$. For answering distance queries with a strong $(1+\epsilon)$-approximation we use the original graph $G$ for $\widehat{G}$. Let us remind that a $(1+\epsilon)$-approximate estimate $\delta(u, v)$ is said to be strong if $d_G(u, v) \leq \delta(u, v) \leq d_G(u, v) + \epsilon \cdot \zeta(u, v)$, where

$$\zeta(u, v) = max\{\ell \mid \exists \text{ a shortest path in } G \text{ between } u \text{ and } v \text{ with maximal edge length } \ell\}.$$

### 6.1.1 $(1+\epsilon)$-approximate Distance Query Algorithm

The algorithm (Fig. 10) for answering $(1+\epsilon)$-approximate distance queries relies on the fact that $G'$ is a $(1+\epsilon)$-spanner. The following theorem analyzes the algorithm Dist-query$_{(1+\epsilon)}$.

**Theorem 7.** *Let $G$ be a $k$-dimensional ball graph defined on $\mathcal{D}$ and $G'$ be a $(1+\epsilon)$-spanner of $G$ constructed as described above. The graph $G'$ can be preprocessed in $O(nf(n, S(\mathcal{D}), \epsilon) \log n)$ time and $O(nf(n, S(\mathcal{D}), \epsilon))$ space such that subsequent distance queries under the $d_G$ metric can be answered with $(1+\epsilon)$-stretch in $O(f(n, S(\mathcal{D}), \epsilon))$ time, where $f(n, S(\mathcal{D}), \epsilon) = n^{1-1/k}\epsilon^{-k+1/2} + \epsilon^{-2k+1} \log S(\mathcal{D})$.*

| Algorithm Dist-query$_{(1+\epsilon)}$ |
|---|
| **Input:** ball graph $G = (V, E)$, vertices $u, v \in V$ |
| **Output:** Dist-query$(G, G', u, v)$ (where $G'$ is the spanner constructed as above) |

Figure 10: Algorithm for $(1 + \epsilon)$-approximate answering of distance queries.

*Proof.* For every pair of query vertices $u, v$, the algorithm reports the exact distance between $u$ and $v$ in $G'$. Since $G'$ is a $(1 + \epsilon)$-spanner,

$$d_G(u, v) \leq d_{G'}(u, v) \leq (1 + \epsilon) d_G(u, v).$$

Therefore, the answers are $(1 + \epsilon)$-approximate.

The running time of the preprocessing is dominated by the time for running Sssp from all the separator nodes. Using Dijkstra's algorithm for Sssp gives a running time of $O(nf(n, S(\mathcal{D}), \epsilon) \log n)$ for the preprocessing phase. The space needed for storing all the results of Sssp computations is $O(nf(n, S(\mathcal{D}), \epsilon))$. The query time is dominated by the Step 4d of Dist-query and can be bounded by $O(f(n, S(\mathcal{D}), \epsilon))$. □

### 6.1.2 Strong $(1 + \epsilon)$-approximate Distance Query Algorithm

The algorithm (Fig. 11) for answering strong $(1 + \epsilon)$-approximate distance queries uses the original graph $G$ for $\widehat{G}$. Thus, the shortest path computations (Step 2b of algorithm Dist-query) are performed on $G$ itself. Since we compute distances using $G$, the distance queries can be answered more accurately. But since $G$ has more edges than $G'$, requirements for the preprocessing time and space are now higher. To illustrate the main ideas in our algorithm

| Algorithm Dist-query$_{\text{strong}(1+\epsilon)}$ |
|---|
| **Input:** ball graph $G = (V, E)$, vertices $u, v \in V$ |
| **Output:** Dist-query$(G, G, u, v)$ |

Figure 11: Algorithm for strong $(1 + \epsilon)$-approximate answering of distance queries.

we start by considering the easier case where all the disks have the same radius.

**Theorem 8.** *Let $G$ be a unit disk graph with $m = \Omega(n \log n)$ edges. The graph $G$ can be preprocessed in $O(m\sqrt{n}\epsilon^{-1})$ time, producing a data structure of size $O(n^{3/2}\epsilon^{-1})$, such that subsequent distance queries can be answered approximately, in $O(\sqrt{n}\epsilon^{-1})$ time. The outputs produced are strong $(1 + \epsilon)$-approximate distance estimates.*

*Proof.* If there is an edge between $u$ and $v$ in $G$, then the actual distance is the estimate. Otherwise, we know $d_G(u, v) > 2$. Consider a shortest path $P_{sh} = (u_1, u_2, \ldots, u_z)$ between $u_1 = u$ and $u_z = v$ with $z > 2$ and no shortcuts, i.e., no edge from $u_{j-1}$ to $u_{j+1}$ for any $j$ between 2 and $z - 1$. Such a path $P_{sh}$ exists due to the triangle inequality. Since $d(u_{j-1}, u_{j+1}) > 2$, $\max\{d(u_{j-1}, u_j), d(u_j, u_{j+1})\} \geq 1$. Let $(sq(u_1), \ldots, \ldots, sq(u_z))$ be the sequence of squares in *Roots* such that $sq(u_i)$ contains $u_i$. Remember that a square contains a point if the point lies within the boundary of the squares. We know the path $(R_{sq(u_1)}, \ldots, R_{sq(u_z)})$ exists in $G'$ (by construction). Let $s$ be the deepest level node in $T_{G'}$ where this path gets separated. This implies that among the vertices $\{R_{sq(u_1)}, \ldots, R_{sq(u_z)}\}$ at least one is in $Sep(s)$, say $R_{sq(u_i)}$.

We compute single source shortest paths from the vertex $R_{sq(u_i)}$, which is at most $\sqrt{2}\epsilon$ distance away from $u_i$. There is also an edge between $u_i$ and $R_{sq(u_i)}$ in $G$. This proves that the estimate is only $O(\epsilon)$ greater than the length of an actual shortest path. It is also a strong $(1+\epsilon)$-approximate estimate, because there exists an edge in the path of $P_{sh}$ with length at least 1.

The running time of the preprocessing is dominated by the time for running Sssp from all the separator nodes. Using Dijkstra's algorithm for Sssp gives a running time of $O(m\sqrt{n}\epsilon^{-1})$ for the preprocessing phase. Note that since $m = \Omega(n \log n)$, every run of Dijkstra's algorithm can be performed in $O(m)$ time using Fibonacci heaps. The space needed for storing the results of all Sssp computations is $O(n^{3/2}\epsilon^{-1})$. The query time is dominated by Step 4d and can be bounded by $O(\sqrt{n}\epsilon^{-1})$. □

**Strong $(1+\epsilon)$-approximate Distance Queries in Ball Graphs:** Again we describe the algorithm for the case of disk graphs and then state the extensions to higher dimensions. For two squares $s$ and $s'$ in $\Gamma$, the distance $dist(s, s')$ is the Euclidean distance between their centers. Extending the notion of representative vertices, we define representative paths.

We define a *representative path* in $G'$ for every edge of $G$. For a vertex $w$ in $G$, let $sq(w)$ denote the deepest level node in $\Gamma$ containing $w$. Consider an edge $(u, v)$ of the graph $G$. In the rest of the discussion we assume without loss of generality that $r_u \leq r_v$. The representative path $P(u, v)$ starts at $R_{sq(u)}$ and ends at $R_{sq(v)}$.

Let $a$ be the deepest level node in *Roots* containing $u$. Let $b$ be a node in $\Gamma$ containing $v$ with $d_a = d_b$. Since $u$ and $R_{sq(u)}$ (similarly $v$ and $R_{sq(v)}$) are always contained in the same node in $\Gamma$, we get that $R_{sq(u)}$ is contained in $a$ and $R_{sq(v)}$ is contained in $b$. The representative path $P(u, v)$ for an edge $(u, v)$ is defined using the following case distinction:

**Case 1:** If the distance between $a$ and $b$ is greater than $2^{-d_a}$, then by Lemma 2, we know that there exists some $c \in Bucket(a)$ containing $v$ (and thus also $R_{sq(v)}$). Since $dist(a, c) \geq 2^{-d_c}$, we also know that $r_v \geq 2^{-d_c-1}$ and $v \in C(c)$. In $G'$ there is an edge between $R_a$ and $R_c$. From Lemma 4, we know there exists a path in $G'$ between $R_{sq(u)}$ and $R_a$ and between $R_{sq(v)}$ and $R_c$. Define the representative path $P(u, v)$ as $(R_{sq(u)}, \ldots, R_a, R_c, \ldots, R_{sq(v)})$.
**Case 2:** Otherwise, consider the deepest level nodes $f, g$ in $\Gamma$, such that (1) $R_{sq(u)}$ is contained in $f$ and $R_{sq(v)}$ is contained in $g$, and (2) there exists an edge $(R_f, R_g)$ in $G'$. From Lemma 4, we know there exists a path between $R_{sq(u)}$ to $R_f$ and between $R_{sq(v)}$ and $R_g$ in $G'$. Define the representative path $P(u, v)$ as $(R_{sq(u)}, \ldots, R_f, R_g, \ldots, R_{sq(v)})$.

For every representative path, we also define a pair of nodes in $\Gamma$ as its *covering nodes*. If $P(u, v)$ is defined using Case 1, then the nodes $a$ and $c$ are the covering nodes.

If $P(u, v)$ is defined using Case 2, then $f$ and $g$ are the covering nodes. In both cases, all vertices in $P(u, v)$ are contained in one of the covering nodes with $u$ and $v$ contained in different covering nodes.

**Lemma 11.** *Let $(u, v)$ be an edge in $G$ of length greater than $2^{-l_{max}}$. Let $P(u, v)$ be its representative path in $G'$ with $p$ and $q$ as the covering nodes. Then $dist(p, q) \geq \max\{c_5 2^{-d_p}, c_5 2^{-d_q}\}$, for some constant $c_5$.*

*Proof.* We again use the same case distinction as in defining the path $P(u, v)$. Assume $p$ contains $u$, and $q$ contains $v$. If $P(u, v)$ is defined using Case 1, then $d_p \geq d_q$. Let $r$ be the ancestor of $p$ which is at the same level as $q$. Since $q$ is at least $2^{-d_q}$ away from $r$, we get that $dist(p, q) \geq c_5 2^{-d_q}$ ($p$ is a square inside $r$).

If $P(u, v)$ is defined using Case 2, then $d_p = d_q$. Let $p'$ be the child of $p$ containing $u$. Let $q'$ be the child of $q$ containing $v$. Since there is no edge between $R_{p'}$ and $R_{q'}$, we conclude $dist(p', q') \geq 2^{-d_q - 1}$. This implies that $dist(p, q) \geq c_5 2^{-d_q}$. The existence of nodes $p'$ and $q'$ is guaranteed if $d(u, v) \geq (1+\epsilon/\sqrt{2})2^{-l_{max}}$ (remember that $l_{max}$ is the deepest level in $\Gamma$). Otherwise, $dist(p, q) = \Omega(2^{l_{max}})$ and again $dist(p, q) \geq c_5 2^{-d_q}$, with $c_5 \geq 1/4$. □

The following theorem shows that we get a strong $(1 + \epsilon)$-approximation.

**Theorem 9.** *Let $G$ be a disk graph on $\mathcal{D}$ with $m = \Omega(n \log n)$ edges. The graph $G$ can be preprocessed in $O(m\sqrt{n}\epsilon^{-1} + m\epsilon^{-2} \log S(\mathcal{D}))$ time, producing a data structure of size $O(n^{3/2}\epsilon^{-1} + n\epsilon^{-2} \log S(\mathcal{D}))$, such that subsequent distance queries can be answered approximately, in $O(\sqrt{n}\epsilon^{-1} + \epsilon^{-2} \log S(\mathcal{D}))$ time. The outputs produced are strong $(1 + \epsilon)$-approximate distance estimates.*

*Proof.* If there is an edge between $u$ and $v$ in $G$, then the actual distance is the estimate. As in Theorem 8, consider a shortest path $P_{sh} = (u_1, u_2 \ldots, u_z)$ between $u_1(= u)$ and $u_z(= v)$ with $z > 2$ and no edge shortcuts. Since there is always an edge if the distance between two vertices is less than $2^{-l_{max}+1}$, we get $d(u_{j-1}, u_{j+1}) \geq 2^{-l_{max}+1}$ and $\max\{d(u_{j-1}, u_j), d(u_j, u_{j+1})\} \geq 2^{-l_{max}}$ for any $j$ between 2 and $z-1$. Let $(sq(u_1), \ldots, sq(u_z))$ be the sequence of squares in $\Gamma$ such that $sq(u_i)$ contains $u_i$. Let $s$ be the deepest level node in $T_{G'}$ with $R_{sq(u_1)}, R_{sq(u_z)} \in V(s)$.

If any vertex $R_{sq(u_i)}$ from $\{R_{sq(u_1)}, \ldots, R_{sq(u_z)}\}$ is in the separator $Sep(s)$, then we do a single source shortest paths search from $R_{sq(u_i)}$ on $H(s)$ (graph constructed in Step 2a) of the algorithm Dist-query). Now $R_{sq(u_i)}$ is at most a distance $\epsilon 2^{-l_{max}+1/2}$ away from $u_i$ and there exists an edge between $R_{sq(u_i)}$ and $u_i$ in $G$. This proves the estimate produced is only $O(\epsilon)$ greater than the shortest path. The strong $(1 + \epsilon)$-approximation follows as there is an edge in $P_{sh}$ which is of length at least $2^{-l_{max}}$.

Otherwise, consider two vertices $R_{sq(u_a)}$ and $R_{sq(u_b)}$ from $\{R_{sq(u_1)}, \ldots, R_{sq(u_z)}\}$ which are separated into different components by $Sep(s)$ and for which there exists an edge $(u_a, u_b)$ in $G$ (by the assumption about the path $P_{sh}$ we get $|a - b| = 1$). Since there is no edge between $R_{sq(u_a)}$ and $R_{sq(u_b)}$, we conclude $d(u_a, u_b) \geq 2^{-l_{max}}$. Consider the representative path $P(u_a, u_b)$ in $G'$. There exists at least one vertex (say $z$) on the path $P(u_a, u_b)$ in the separator $Sep(s)$. Let $p$ and $q$ be the two covering nodes for $P(u_a, u_b)$. By definition all the vertices in $P(u_a, u_b)$ are contained in either $p$ or $q$. Assume without loss of generality that $z$ and $u_a$ are contained in $p$.

In $G$ there exist edges $(z, R_p)$ and $(R_p, u_a)$. Since $z, R_p$ and $u_a$ are all contained in $p$ means $d(z, R_p) \leq \epsilon 2^{-d_p+1/2}$ and $d(R_p, u_a) \leq \epsilon 2^{-d_p+1/2}$. On the other hand, $dist(p, q) \geq \max\{c_5 2^{-d_p}, c_5 2^{-d_q}\}$ by Lemma 11. This also implies that $d(u_a, u_b) \geq c_6 2^{-d_p}$ for some constant $c_6$. Putting all together we get $d(z, R_p) \leq c_7 \epsilon d(u_a, u_b)$ and $d(R_p, u_a) \leq c_7 \epsilon d(u_a, u_b)$ for some constant $c_7$, implying in $G$ there is a path from $z$ to $u_a$ of length at most $2c_7 \epsilon d(u_a, u_b)$. Therefore, when we do the single source shortest path computation from $z$ on $H(s)$, the error we make in taking the detour can be bounded by some constant times $\epsilon d(u_a, u_b)$. Implying that our estimate is a strong $(1+\epsilon)$-approximation.

The running time analysis follows as in Theorem 8. The preprocessing time and space are related to the size of the separator in $G$. □

In $\mathbb{R}^k$, the spanner of a ball graph $G$ can be constructed in $O(n^{(2/\ell)+\delta} \epsilon^{-k/\ell} \log^{1/\ell} S(\mathcal{D}))$ time, where $\ell = 1 + 1/(\lfloor k/2 \rfloor + 1)$ (Theorem 2). By repeating the same analysis as in the case of disk graphs (Theorem 9), we have the following result.

**Theorem 10.** *Let $G$ be a $k$-dimensional ball graph on $\mathcal{D}$ with $m = \Omega(n \log n)$ edges. The graph $G$ can be preprocessed in $O(mn^{1-1/k} \epsilon^{-k+1} + m\epsilon^{-k} \log S(\mathcal{D}))$ time, producing a data structure of size $O(n^{2-1/k} \epsilon^{-k+1} + n\epsilon^{-k} \log S(\mathcal{D}))$, such that subsequent distance queries can be answered approximately, in $O(n^{1-1/k} \epsilon^{-k+1} + \epsilon^{-k} \log S(\mathcal{D}))$ time. The outputs produced are strong $(1+\epsilon)$-approximate distance estimates.*

## 7 Concluding Remarks

The bottleneck in the running time of our spanner algorithms is the time for finding the intersections between sets of balls. Faster algorithms for finding the intersections would improve the running time of the spanner construction. For disk graphs, where all disks have almost the same radius (between $[1-\epsilon, 1]$), our spanner construction for unit disk graphs still works by replacing the bichromatic closest pair algorithm with Sharir's [36] bichromatic disk intersection algorithm. Therefore, we get a running time of $O(n\epsilon^{-2} \log^2 n)$.

An obvious open problem is to obtain algorithms with improved running times. We believe that the $n^\delta$ factors appearing in the Theorems 1 and 2 could be reduced to poly$(\log n)$ by a more careful analysis of the data structure used for halfspace range searching. Another interesting problem would be to investigate faster algorithms for constructing the modified Yao graph.

All the results presented in this paper can also be extended to cases when intersections are between squares, or regular polygons, or other disk-like objects, as well as their higher dimensional versions. This follows as our algorithms don't use any special properties of balls (or disks). However, the partitioning scheme only works if all the objects have almost the same aspect ratio. An open problem is to extend the results to intersection graphs between objects not having this property.

Also, a more general network model, the quasi unit-disk graph (quasi-UDG) model, captures much better the characteristics of wireless networks [24]. A quasi-UDG with parameters $r$ and $R$ ($r \geq 0$, $R > 0$) over a set of points in the plane is defined as follows. The points in the set are the vertices of the graph. For any two points $u$ and $v$ in the set with Euclidean distance $d(u, v)$: if $d(u, v) \leq r$ then $(u, v)$ is an edge in the graph; if $d(u, v) > R$ then $(u, v)$ is not an edge in the graph; and if $r < d(u, v) \leq R$ then $(u, v)$ may or may not be an edge in the graph. Constructing spanners for quasi unit-disk graphs is an interesting open problem.

**Acknowledgment** We thank the reviewers for their insightful suggestions that immensely improved the paper.

## References

[1] Mohammad Ali Abam and Sariel Har-Peled. New Constructions of SSPDs and their Applications. *Computational Geometry*, 45(5-6):200–214, 2012.

[2] Pankaj K. Agarwal, Herbert Edelsbrunner, Orfried Schwarzkopf, and Emo Welzl. Euclidean Minimum Spanning Trees and Bichromatic Closest Pairs. *Discrete & Computational Geometry*, 6:407–422, 1991.

[3] Pankaj K. Agarwal and Jirí Matoušek. Dynamic Half-Space Range Reporting and Its Applications. *Algorithmica*, 13(4):325–345, 1995.

[4] Jochen Alber and Jiří Fiala. Geometric Separation and Exact Solutions for the Parameterized Independent Set Problem on Disk Graphs. *Journal of Algorithms*, 52(2):134–151, 2004.

[5] Srinivasa R. Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid, and Christos D. Zaroliagis. Planar Spanners and Approximate Shortest Path Queries among Obstacles in the Plane. In *ESA '96*, volume 1136, pages 514–528. Springer, 1996.

[6] Sunil Arya, Gautam Das, David M. Mount, Jeffrey S. Salowe, and Michiel Smid. Euclidean Spanners: Short, Thin, and Lanky. In *STOC '95*, pages 489–498. ACM, 1995.

[7] Marshall W. Bern, David Eppstein, and Shang-Hua Teng. Parallel Construction of Quadtrees and Quality Triangulations. *International Journal of Computational Geometry & Applications*, 9(6):517–532, 1999.

[8] Paul B. Callahan and S. Rao Kosaraju. Faster Algorithms for Some Geometric Graph Problems in Higher Dimensions. In *SODA '93*, pages 291–300. SIAM, 1993.

[9] Paul B. Callahan and S. Rao Kosaraju. A Decomposition of Multidimensional Point Sets with Applications to $K$-nearest-neighbors and $N$-body Potential Fields. *Journal of ACM*, 42(1):67–90, 1995.

[10] David Eppstein. Spanning Trees and Spanners. In Jörg-Rudiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier, 2000.

[11] David Eppstein. Testing Bipartiteness of Geometric Intersection Graphs. In *SODA '04*, pages 860–868. SIAM, 2004.

[12] Jeff Erickson. On the Relative Complexities of Some Geometric Problems. In *CCCG '95*, pages 85–90, 1995.

[13] Bin Fu. Theory and Application of Width Bounded Geometric Separator. In *STACS '06*, volume 3884, pages 277–288. Springer, 2006.

[14] Martin Fürer and Shiva Prasad Kasiviswanathan. Approximate Distance Queries in Disk Graphs. In *WAOA '06*, volume 4368, pages 174–187. Springer, 2006.

[15] Martin Fürer and Shiva Prasad Kasiviswanathan. Spanners for Geometric Intersection Graphs. In *WADS '07*, volume 4619, pages 312–324. Springer, 2007.

[16] Jie Gao, Leonidas J. Guibas, John Hershberger, Li Zhang, and An Zhu. Geometric Spanner for Routing in Mobile Networks. In *MobiHoc '01*, pages 45–55, 2001.

[17] Jie Gao and Li Zhang. Well-separated Pair Decomposition for the Unit-Disk Graph Metric and its Applications. *SIAM Journal on Computing*, 35(1):151–169, 2005.

[18] Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Approximate Distance Oracles for Geometric Graphs. In *SODA '02*, pages 828–837. ACM, 2002.

[19] Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Approximate Distance Oracles for Geometric Spanners. *ACM Trans. Algorithms*, 4(1):1–34, 2008.

[20] Joachim Gudmundsson, Christos Levcopoulos, Giri Narasimhan, and Michiel H. M. Smid. Approximate Distance Oracles Revisited. In *ISAAC '02*, volume 2518, pages 357–368. Springer, 2002.

[21] Prosenjit Gupta, Ravi Janardan, and Michiel Smid. Algorithms for Some Intersection Searching Problems Involving Circular Objects. *International Journal of Mathematical Algorithms*, 1:35–52, 1999.

[22] Sariel Har-Peled. *Geometric Approximation Algorithms*, volume 173. AMS Mathematical Surveys and Monographs, 2011.

[23] Drago Krznaric, Christos Levcopoulos, and Bengt J. Nilsson. Minimum Spanning Trees in d Dimensions. *Nordic Journal of Computing*, 6(4):446–461, 1999.

[24] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Ad hoc Networks Beyond Unit Disk Graphs. *Wireless Networks*, 14(5):715–729, 2008.

[25] Xiang-Yang Li. Algorithmic, Geometric and Graph Issues in Wireless Networks. *Wireless Communications and Mobile Computing*, 3(2):119–140, 2003.

[26] Xiang-Yang Li, Gruia Calinescu, and Peng-Jun Wan. Distributed Construction of Planar Spanner and Routing for Ad Hoc Wireless Networks. In *INFOCOM '02*, volume 3, pages 1268–1277. IEEE, 2002.

[27] Xiang-Yang Li, Peng-Jun Wan, and Ophir Frieder. Coverage in Wireless Ad Hoc Sensor Networks. *IEEE Transactions on Computers*, 52(6):753–763, 2003.

[28] Xiang-Yang Li and Yu Wang. Efficient Construction of Low Weight Bounded Degree Planar Spanner. In *COCOON '03*, volume 2697, pages 374–384. Springer, 2003.

[29] Zvi Lotker and David Peleg. Structure and Algorithms in the SINR Wireless Model. *SIGACT News*, 41(2):74–84, 2010.

[30] Thomas Lukovszki. New Results on Geometric Spanners and Their Applications. PhD thesis, University of Paderborn, 1999.

[31] Carver Mead and Lynn Conway. *Introduction to VLSI System*. Addison-Wesley, Reading, 1980.

[32] Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

[33] Rajmohan Rajaraman. Topology Control and Routing in Ad hoc Networks: A Survey. *SIGACT News*, 33:60–73, 2002.

[34] Jim Ruppert and Raimund Seidel. Approximating the $d$-dimensional Complete Euclidean Graph. In *CCCG '91*, pages 207–210, 1991.

[35] Baruch Schieber and Uzi Vishkin. On Finding Lowest Common Ancestors: Simplification and Parallelization. *SIAM Journal on Computing*, 17(6):1253–1262, 1988.

[36] Micha Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM Journal on Computing*, 14:448–468, 1985.

[37] Michiel Smid. The Well-separated Pair Decomposition and Its Applications. In Teofilo Gonzalez, editor, *Handbook on Approximation Algorithms and Metaheuristics*, pages 53–1 – 53–12. Chapman & Hall/CRC, 2007.

[38] Warren D. Smith and Nicholas C. Wormald. Geometric Separator Theorems & Applications. In *FOCS '98*, pages 232–243. IEEE, 1998.

[39] Anand Srinivas and Eytan Modiano. Minimum Energy Disjoint Path Routing in Wireless Ad-hoc Networks. In *MOBICOM '03*, pages 122–133. ACM, 2003.

[40] Mikkel Thorup and Uri Zwick. Approximate Distance Oracles. *Journal of ACM*, 52(1):1–24, 2005.

[41] Andrew Chi-Chih Yao. On Constructing Minimum Spanning Trees in $k$-Dimensional Spaces and Related Problems. *SIAM Journal on Computing*, 11(4):721–736.

[42] Uri Zwick. Exact and Approximate Distances in Graphs - A Survey. In *ESA '01*, volume 2161, pages 33–48. Springer, 2001.

## A Compressed Quadtree Construction

In this section, we discuss the construction of the compressed forest $\Gamma' = (V_{\Gamma'}, E_{\Gamma'})$. We start by introducing some terminology. Given two fixed-point real numbers $X = \sum x_i 2^i$ and $Y = \sum y_i 2^i$, define a bitwise shuffle operation of $X$ and $Y$ as: $X|Y = \sum_i (2x_i + y_i)4^i$. Define $agree(X, Y)$ with $X \neq Y$ as the first location after the binary point in which $X$ and $Y$ differ (equivalently it is the smallest $l \geq 0$ such that we have $\lfloor X2^{l+1} \rfloor \neq \lfloor Y2^{l+1} \rfloor$).

For a point $p$, let $x(p)$ and $y(p)$ denote its $x$- and $y$-coordinates. Sort the points in $\mathcal{P}$ in increasing order of their shuffled representations. Let $\mathcal{P}_s = (p_1, p_2, \ldots, p_n)$ denote this sorted sequence (i.e., $x(p_{i-1})|y(p_{i-1}) \leq x(p_i)|y(p_i)$ for $2 \leq i \leq n$). Note that the (bit-fiddling) shuffle operation need not be performed explicitly, as *agree* can be used to compare shuffled numbers. Because if $X|Y < X'|Y'$, then either $agree(X, X') \leq agree(Y, Y')$ and $X < X'$ or $agree(X, X') > agree(Y, Y')$ and $Y < Y'$. Construct a list $A_s$ whose $i$th entry $a_i$ is the minimum of $agree(x(p_i)\epsilon^{-1}, x(p_{i-1})\epsilon^{-1})$ and $agree(y(p_i)\epsilon^{-1}, y(p_{i-1})\epsilon^{-1})$. Set $a_1 = 0$. Start the forest with a different leaf node $t_j$ for every disk ($1 \leq j \leq n$). We maintain a stack with *top* representing the last inserted entry and a special bottom of stack entry $(-\infty, Null)$. When we scan a new element $a_i$ in the list $A_s$ we do:

i. If $a_i \geq a_{top}$, push $(a_i, t_i)$ onto the top of the stack and increment $i$.

ii. Else, let $a_{top} = a_{top-1} = \cdots = a_{top-r+2} > a_{top-r+1}$. Note that $2 \leq r \leq 4$. Pop $(a_{top}, t_{top}), \ldots, (a_{top-r+1}, t_{top-r+1})$. Create a new node $t$ in $\Gamma'$, and let its children be $t_{top}, \ldots, t_{top-r+2}, t_{top-r+1}$. Label $t$ with $a_{top}$, and push $(a_{top-r+1}, t)$ on the stack.

The level of a node $t$ ($d_t$) is the label of $t$. At every node in $\Gamma'$, we also maintain a disk with a largest radius in it. The representative $R_t$ of $t \in V_{\Gamma'}$ is chosen to be a disk with a largest radius which was also the representative in one of the children of $t$. Then we visit each node $t$ and add the children of $t$ to *Roots* if among the disks with centers contained in $t$, the largest radius is less than $2^{-d_t}$.

The time for constructing the forest is $O(n \log n)$, assuming the *agree*[8] operation can be implemented in constant time.

[8]Our model of computation is the real RAM model augmented with the *agree* operation. This assumption is present in other constructions also [7, 22].