Faster Algorithms for Growing Prioritized Disks and Rectangles

Hee-Kap Ahn, Sang Won Bae, Jongmin Choi, Matias Korman, Wolfgang Mulzer, Eunjin Oh, Ji-won Park, André van Renssen, and Antoine Vigneron

1Department of Computer Science and Engineering, POSTECH, Republic of Korea, {heekap, icothos, jin9082}@postech.ac.kr
2Department of Computer Science, Kyonggi University, Republic of Korea, subae@kgu.ac.kr
3Tohoku University, Sendai, Japan, mati@dais.is.tohoku.ac.jp
4Institut für Informatik, Freie Universität Berlin, Germany, mulzer@inf.fu-berlin.de
5School of Computing, KAIST, Republic of Korea, sldnjsi727@naver.com
6National Institute of Informatics (NII), Tokyo, Japan, andre@nii.ac.jp
7JST, ERATO, Kawarabayashi Large Graph Project
8School of Electrical and Computer Engineering, UNIST, Republic of Korea, antoine@unist.ac.kr

Abstract

Motivated by map labeling, we study the problem in which we are given a collection of \( n \) disks \( D_1, \ldots, D_n \) in the plane that grow at possibly different speeds. Whenever two disks meet, the one with the lower index disappears. This problem was introduced by Funke, Krumpe, and Storandt [IWOCA 2016]. We provide the first general subquadratic algorithm for computing the times and the order of disappearance. This algorithm also works for other shapes (such as rectangles) and in any fixed dimension.

Using quadtrees, we provide an alternative algorithm that runs in near linear time, although this second algorithm has a logarithmic dependence on either the ratio of the fastest speed to the slowest speed of disks or the spread of disk centers (the ratio of the maximum to the minimum distance between them). Our result improves the running times of previous algorithms by Funke, Krumpe, and Storandt [IWOCA 2016], Bahrdt et al. [ALENEX 2017] and Funke and Storandt [EWCG 2017]. Finally, we give an \( \Omega(n \log n) \) lower bound on the problem, showing that our quadtree algorithms are almost tight.

1 Introduction

Suppose we are given a sequence \( D_1, \ldots, D_n \) of growing disks. At time \( t = 0 \), each disk \( D_i \) starts out as a point \( p_i \in \mathbb{R}^2 \), and as time passes, it grows linearly with growth rate \( v_i > 0 \). Thus, at any time \( t \geq 0 \), the disk \( D_i \) is centered at \( p_i \) and has radius \( tv_i \). The position of a disk in the sequence corresponds to its priority (the smaller the index, the higher its priority). Whenever two disks meet, we eliminate the one with lower priority from the arrangement. More precisely, for any \( 1 \leq i < j \leq n \), let \( t(i,j) = |p_ip_j|/(v_i + v_j) \). Then, if neither of the two disks \( D_i \) and \( D_j \) has been removed before time \( t(i,j) \), we eliminate \( D_j \) at this time, while \( D_i \) keeps growing. Our goal is to determine the elimination order, that is, the instant of time and the order in which the disks are removed from the arrangement.

Motivated by map labeling, this problem was first considered by Funke, Krumpe and Storandt [6]. As one zooms out from a labeled map, labels grow in size. Clearly, we do not want the labels to overlap, so whenever this happens, one of the two is removed. This creates the need to determine when...
Table 1: Summary of our results. The $O(dn^2)$-time algorithm in the first row works for growing objects of any shape in $\mathbb{R}^d$ such that the touching time of any pair of them can be computed in $O(d)$ steps. $\mathcal{SA}_k$ stands for any semialgebraic shape that is described with $k$ parameters. $\Phi$ denotes the spread of the disk centers and $\Delta = \max_i v_i / \min_j v_j$ is the maximum ratio between two growth rates.

| Shape                       | Time                                      | Method           | Where       |
|-----------------------------|-------------------------------------------|------------------|-------------|
| Balls, Boxes in $\mathbb{R}^d$ | $O(dn^2)$                                | Priority sort    | Section 2   |
| Disks in $\mathbb{R}^2$     | randomized expected $O(n^{5/3+\varepsilon})$ | Bucketing        | Section 3   |
| Rectangles in $\mathbb{R}^2$ | $O(n^{11/6+\varepsilon})$                |                  |             |
| $\mathcal{SA}_k$            | $O(n^{(4k-5)/(2k-2)+\varepsilon})$       |                  |             |
| Cubes in $\mathbb{R}^d$     | $O(n \log^{d+2} n)$                       | Linearity of queries | Section 4 |
| Disks in $\mathbb{R}^2$     | $O(n \log \Phi \min\{\log \Delta, \log \Phi\})$ | Quocketing       | Section 5.1 |
| Disks in $\mathbb{R}^2$     | $O(n(\log n + \min\{\log \Delta, \log \Phi\}))$ | Compressed quadtree | Section 5.2 |

and in which order the labels need to be discarded. Funke, Krumpe and Storandt [9] observed that a straightforward simulation of the growth process with a priority queue solves the problem in time $O(n^2 \log n)$. They also gave an algorithm that runs in expected time $O(n(\log^6 n + \Delta^2 \log^2 n + \Delta^4 \log n))$, where $\Delta = \max_i v_i / \min_j v_j$ is the maximum ratio between two growth rates. Subsequently, Bahrdt et al. [2] improved this to an algorithm that runs in worst-case $O(\Delta^2 n(\log n + \Delta^2))$ time. This generalizes to growing balls in arbitrary fixed dimension $d$, with running time $O(\Delta^d n(\log n + \Delta^d))$. Recently, Funke and Storandt [7] presented two further parameterized algorithms for the problem. The first algorithm runs in time $O(n \log \Delta(\log n + \Delta^{-1}))$ for arbitrary dimension $d$, while the second algorithm is specialized for the plane and runs in time $O(Cn \log^O(1) n)$, where $C$ denotes the number of distinct growth rates.

If we are interested only in the first pair of touching disks, this problem is equivalent to the weighted closest pair of the disk centers. Formann showed how to compute it in optimal $O(n \log n)$ time [5].

**Our results.** We first present a simple algorithm that runs in time $O(dn^2)$ in any fixed dimension $d$ (Section 2). In Section 3 we speed it up by combining it with an advanced data structure for querying lower envelopes of algebraic surfaces [1][10] and bucketing. The running time depends on the exact shape and dimension of the objects. In particular, the algorithm runs in randomized expected time $O(n^{5/3+\varepsilon})$ for disks, and $O(n^{11/6+\varepsilon})$ time for rectangles, also in two dimensions. These are the first subquadratic-time algorithms for growing disks and rectangles in the plane. More generally, we show that the elimination sequence of a set of $n$ growing objects of any semi-algebraic shape described with $k \geq 4$ parameters can be computed in subquadratic time for any fixed $k$. In Section 4 we consider the case of growing squares. These objects are much simpler, hence we can use ray shooting techniques and similar properties to reduce the running time to $O(n \log^{d+2} n)$.

In Section 5 we consider a completely different approach based on quadtrees. The main difference is that the running time of these algorithms also depends on the spread $\Phi$ of the disk centers (that is, the ratio of the maximum to the minimum distance between disk centers) and the ratio $\Delta$ between the fastest and slowest speed of the disks. Table 4 provides a summary of our results. Finally, we give an $\Omega(n \log n)$ lower bound using a simple reduction from sorting. Our algorithm using compressed quadtrees is thus nearly optimal as well as it is an improvement over Bahrdt et al.’s algorithm in [2] that runs in $O(\Delta^2 n(\log n + \Delta^2))$ time.

**Notation.** For any $1 \leq i \leq n$, we denote by $t_i$ the time at which disk $D_i$ is eliminated. Since $D_i$ will never be eliminated, we set $t_1 = \infty$. We denote by $t(i,j) = |p_i p_j|/(v_i + v_j)$ the time at which disks the $D_i$ and $D_j$ would touch, supposing that no other disk has interfered. We assume general position, meaning that all times $t(i,j)$ for $i \neq j$ are pairwise distinct.
2 A simple quadratic algorithm

We provide a simple iterative way to determine the elimination times $t_i$. This method will be used for small groups of disks afterwards. As noted above, we have $t_1 = \infty$. For $i \geq 2$, the next lemma shows how to find $t_i$, provided that $t_1, \ldots, t_{i-1}$ are known.

**Lemma 1.** Let $i \in \{2, \ldots, n\}$, and let

$$j^* = \arg\min_{j=1,\ldots,i-1} \{t(i,j) \mid t(i,j) \leq t_j\}.$$

Then, $t_i = t(i,j^*)$, i.e., the disk $D_i$ is eliminated by the disk $D_{j^*}$.

**Proof.** On the one hand, we have $t_i \leq t(i,j^*)$, because at time $t(i,j^*)$, the disk $D_i$ would meet the disk $D_{j^*}$ that has higher priority and that has not been eliminated yet. On the other hand, we have $t_i \geq t(i,j^*)$, because every disk that $D_i$ could meet before time $t(i,j^*)$ either has lower priority or has been eliminated before the encounter. \hfill \square

Lemma 1 leads to a straightforward iterative algorithm, see Algorithm 1.

**Algorithm 1** A quadratic time algorithm

```
1: function EliminationOrder($p_1, \ldots, p_n, v_1, \ldots, v_n$)
2:     $t_1 \leftarrow \infty$
3:     for $i \leftarrow 2, n$ do
4:         $t_i \leftarrow t(i, 1)$
5:         for $j \leftarrow 2, i-1$ do
6:             if $t_j \geq t(i,j)$ and $t_i \geq t(i,j)$ then
7:                 $t_i \leftarrow t(i,j)$
8:     $S \leftarrow \{D_1, \ldots, D_n\}$
9:     Sort $S$ using key $t_i$ for each disk $D_i$
10:    return $S$
```

**Theorem 2.** Algorithm 1 computes the elimination order of a set of prioritized disks in $O(n^2)$ time. It generalizes to growing objects of any shape in $\mathbb{R}^d$ such that the touching time of any pair of them can be computed in $O(d)$ steps, with running time $O(dn^2)$.

**Proof.** The correctness follows directly from Lemma 1. The running time analysis is straightforward. Lemma 1 is purely combinatorial and requires only that the times $t(i,j)$ are well defined. Thus, Algorithm 1 can be generalized to balls and rectangles in $\mathbb{R}^d$ by using an appropriate subroutine for computing $t(i,j)$. This subroutine takes $O(d)$ steps. \hfill \square

3 A subquadratic algorithm using bucketing

We now improve Algorithm 1 by using a bucketing approach and lifting the problem to higher dimensions. For this purpose, we will use a data structure for querying lower envelopes in $\mathbb{R}^4$, which allows us to compute $t_i$ in increasing order of $i$.

Suppose that for a set $B \subset \{1, \ldots, n\}$ of indices, we know the elimination time $t_j$ of any $D_j$ with $j \in B$. In an elimination query, we are given a query index $q > \max B$, and we ask for the disk $D_j$ with $j^* \in B$, that eliminates the query disk $D_q$. The argument from Lemma 1 shows that we can find $j^*$ as follows:

$$j^* = \arg\min_{j \in B} \{t(q,j) \mid t(q,j) \leq t_j\}.$$ 

This leads to a natural interpretation of elimination queries: a query disk $D$ corresponds to a point $(x,y,v) \in \mathbb{R}^3$, where $(x,y)$ is the center of $D$ and $v$ is the growth rate. For each $j \in B$, consider the
function \( f_j : \mathbb{R}^3 \rightarrow \mathbb{R} \) defined by

\[
 f_j(x, y, v) = \begin{cases} 
 t(j, D(x, y, v)), & \text{if } t(j, D(x, y, v)) < t_j, \\
 \infty, & \text{otherwise},
\end{cases}
\]

where \( t(j, D(x, y, v)) \) denotes the time when \( D_j \) and the growing disk given by \((x, y, v)\) touch. For \( q > \max B \), let \((x_q, y_q, v_q) \in \mathbb{R}^3\) be the point that represents \( D_q \). Then, the elimination query \( q \) corresponds to finding the point vertically above \((x_q, y_q, v_q)\) in the lower envelope of the graphs of the functions \( f_j \) for all \( j \in B \). The following lemma is a direct consequence of a result by Agarwal et al. [1].

**Lemma 3.** Let \( B \subset \{1, \ldots, n\} \) with \(|B| = m\). Then, for any fixed \( \varepsilon > 0 \), elimination queries for \( B \) can be answered in \( O(\log^2 m) \) time, after randomized expected preprocessing time \( O(m^{3+\varepsilon}) \).

We now describe our subquadratic algorithm. Set \( m = \lfloor n^{1/3} \rfloor \). We group the disks into \( \lfloor m/n \rfloor \) buckets \( B_1, \ldots, B_{\lfloor m/n \rfloor} \) such that the \( k \)-th bucket \( B_k \) contains the disks \( D_{(k-1)m+1}, \ldots, D_{km} \). There are \( O(n^{2/3}) \) buckets, and each bucket contains at most \( m \) disks. As before, we compute the elimination times \( t_1, \ldots, t_n \) in this order. As soon as the elimination times of all the disks in a bucket \( B_k \) have been determined, we construct the elimination query data structure for \( B_k \). For each bucket, this takes \( O(n^{1+\varepsilon}) \) time, for a total time of \( O(n^{5/3+\varepsilon}) \).

Now, in order to determine the elimination time \( t_i \) of a disk \( D_i \), note that we must check the previous buckets (as well as the bucket containing \( D_i \)). We first perform elimination queries for the previous buckets, that is, buckets \( B_k \) with \( 1 \leq k \leq \lfloor (i-1)/m \rfloor \). There are \( O(n^{2/3}) \) such queries, so this takes \( O(n^{2/3} \log^2 n) \) time. Then, we handle the disks \( D_j, 1 \leq j < i, \) that are in the same bucket as \( D_i \) by brute force, which takes \( O(n^{1/3}) \) time. Overall, the running time is dominated by the time spent in preprocessing the buckets for elimination queries, which takes \( O(n^{5/3+\varepsilon}) \) time.

**Theorem 4.** The elimination sequence of a set of \( n \) growing disks can be computed in randomized expected time \( O(n^{5/3+\varepsilon}) \) for any fixed \( \varepsilon > 0 \).

As before, our algorithm generalizes to other types of shapes. Consider for example the problem of growing rectangles in \( \mathbb{R}^2 \). Each rectangle is given by 4 parameters: the \( x \)- and \( y \)-coordinates of two opposite corners after one unit of time (these values allow us to also obtain the center and the speed of the rectangle). Thus, the data structure for elimination queries is obtained by computing a lower envelope in \( \mathbb{R}^2 \). Given \( m \) growing rectangles, such a data structure with query time \( O(\log m) \) can be constructed in \( O(m^{k+\varepsilon}) \) time for any fixed \( \varepsilon > 0 \) [10]. We apply the same approach as for growing disks, but using buckets of size \( m = \lfloor n^{1/6} \rfloor \). This gives the following theorem.

**Theorem 5.** The elimination sequence of a set of \( n \) growing rectangles can be computed in time \( O(n^{11/6+\varepsilon}) \) for any \( \varepsilon > 0 \).

More generally, we can use regions defined by any semi-algebraic shape of constant complexity. If the shape of the object is described with \( k \geq 4 \) parameters, we need to construct the lower envelope of \( n \) surfaces in \( \mathbb{R}^{k+1} \) to answer elimination queries. After \( O(n^{2k-2+\varepsilon}) \)-time preprocessing, we can answer queries in logarithmic time [10] (again, for any fixed \( \varepsilon > 0 \)). The optimal size of the buckets is \( n^{1/(2k-2)} \), which gives an overall running time of \( O \left( n^{\frac{2k-2}{k-1}+\varepsilon} \right) \), which is subquadratic for any fixed \( k \geq 4 \).

**Theorem 6.** The elimination sequence of a set of \( n \) growing objects of any semi-algebraic shape described with \( k \geq 4 \) parameters can be computed in \( O \left( n^{\frac{2k-2}{k-1}+\varepsilon} \right) \) time for any \( \varepsilon > 0 \).

## 4 Growing Cubes

Axis-aligned cubes in \( \mathbb{R}^d \) are described with \( d + 1 \) parameters. Thus, we can use the approach of the previous section to find the elimination order. However, elimination queries become much easier, since
they are linear functions on the input. In this section, we combine the bucketing approach with ray shooting techniques for lines to reduce the running time to a slightly superlinear bound.

To simplify the presentation, we first assume that \( d = 2 \). Now, a sequence of \( n \) growing squares is given by the centers \( p_1, \ldots, p_n \) and the growth rates \( v_1, \ldots, v_n \). At time \( t \geq 0 \), each square \( D_i \) has edge length \( 2v_i t \).

We consider the four quadrants around each center \( p_i = (x_i, y_i) \). The north, east, south, and west quadrants are, respectively, \( \{(x, y) \in \mathbb{R}^2 \mid y - y_i \geq |x - x_i|\} \), \( \{(x, y) \in \mathbb{R}^2 \mid x - x_i \geq |y - y_i|\} \), \( \{(x, y) \in \mathbb{R}^2 \mid -(y - y_i) \geq |x - x_i|\} \), and \( \{(x, y) \in \mathbb{R}^2 \mid -(x - x_i) \geq |y - y_i|\} \).

Suppose that \( p_j \) is in the north quadrant of \( p_i \). Then, the possible elimination time of \( D_i \) and \( D_j \) is \( t(i, j) = |y_j - y_i|/(v_i + v_j) \). Thus, suppose we have a set \( B \subseteq \{1, \ldots, n\} \) of \( m \) growing cubes, and let \( q > \max B \) such that all centers \( p_j \) with \( j \in B \) lie in the north quadrant of \( p_q \). Then, an elimination query for \( q \) in \( B \) is essentially a two-dimensional problem: the \( x \)-coordinates do not matter any more. We can solve it using ray-shooting for the lower envelope of a set of line segments in \( \mathbb{R}^2 \).

**Lemma 7.** Let \( B \subseteq \{1, \ldots, n\} \), \(|B| = m\). We can preprocess \( B \) in \( O(m \log m) \) time, so that elimination queries can be answered in \( O(\log m) \) time, given that the centers of the squares in \( B \) lie in the north quadrant of the query square \( D_q \).

**Proof.** For each \( j \in B \), consider the line segment \( t \mapsto y_j - v_j t \), defined for \( t \in [0, t_j] \). All these line segments intersect the line \( t = 0 \), so their lower envelope has at most \( \lambda_2(m) = 2m - 1 \) edges, where \( \lambda_2(m) \) denotes the maximum length of a Davenport-Schinzel sequence of order 2 with alphabet size \( m \). An elimination query for a square \( D_q \) with center \((x_q, y_q)\) and growth rate \( v_q \) consists of shooting a ray \( t \mapsto y_q + v_q t \) from below. Thus, we first compute the lower envelope in \( O(m \log m) \) time. Then we build a ray-shooting data structure for this lower envelope, which takes \( O(n) \) preprocessing time with \( O(\log m) \) query time.

We now give a slightly less efficient data structure that does not require \( B \) to be in the north quadrant of \( D_i \).

**Lemma 8.** Let \( B \subseteq \{1, \ldots, n\} \), \(|B| = m\). We can preprocess \( B \) in time \( O(m \log^3 m) \) so that elimination queries can be answered in \( O(\log^2 m) \) time.

**Proof.** Our aim is to build a data structure for each quadrant that answers which square (if any) of \( B \) in the quadrant will be the first to eliminate the query square. To answer a query \( D_q \), we query the data structure for each quadrant, and we return the minimum value.

For each quadrant, the data structure is a two-dimensional range tree, where the coordinate axes have been rotated by an angle \( \pi/4 \), so the new coordinate axes are the bisectors of the original ones. For each canonical subset of each range tree, we construct the data structure of Lemma 7.

Now, given the query disk \( D_q \) and a quadrant, the centers of the disks of \( B \) in this quadrant are in the union of \( O(\log^2 m) \) canonical subsets. So we query the \( O(\log^2 m) \) corresponding data structures in \( O(\log m) \) time each, and we return the result with the smallest timestamp. All these data structures can be built in \( O(m \log^3 m) \) time.

Once we have the data structure for elimination queries, we can apply the bucketing technique from Section 3. This time we will use varying bucket sizes as points are processed. More precisely, we construct a balanced binary tree \( T \) whose leaves represent the squares \( D_1, \ldots, D_n \), from left to right. As usual, a node \( \nu \in T \) represents the subset that consists of the leaves in the subtree that is rooted in \( \nu \).

As soon as the elimination times of all the disks associated with a node of \( T \) have been determined, we compute the elimination query structure from Lemma 7. Thus, after we have determined \( t_j \) for all \( j < i \), we can find \( t_i \) in \( O(\log^2 n) \) time by querying the data structures recorded at \( O(\log n) \) nodes of \( T \) (at most one node per level in the tree will be queried). The running time is bounded by the time needed to preprocess the points for elimination queries \( O(n \log^3 n) \) per level. So overall, this algorithm runs in \( O(n \log^4 n) \) time. In higher dimensions, this bound increases by a factor \( O(\log n) \) per dimension, as we need one more level in the range tree.

**Theorem 9.** The elimination sequence of a set of \( n \) axis-aligned cubes in fixed dimension \( d = O(1) \) can be computed in \( O(n \log^{d+2} n) \) time.
5 Quadtree-based approach

Let $\Phi$ denote the spread of the disk centers and $\Delta$ denote the ratio of the growth rates, i.e., $\Phi = \max_{1 \leq i < j \leq n} |p_ip_j| / \min_{1 \leq i < j \leq n} |p_ip_j|$ and $\Delta = \max_{i \in \{1, \ldots, n\}} v_i / \min_{j \in \{1, \ldots, n\}} v_j$. We first present an algorithm that runs in $O(n \log \Phi \min\{\log \Phi, \log \Delta\})$ time using a quadtree. Then, we present an improved algorithm that runs in $O(n(\log n + \min\{\log \Delta, \log \Phi\}))$ time using a compressed quadtree. To simplify the presentation, we set $\alpha = \min\{\log \Phi, \log \Delta\}$.

5.1 Using an (uncompressed) quadtree

Without loss of generality, all disk centers lie in the unit square $[0, 1]^2$, and their diameter is 1. We construct a quadtree $Q$ for the disk centers. It is a rooted tree in which every internal node has four children. Each node $\nu$ of $Q$ has an associated square cell $b(\nu)$. To obtain $Q$, we recursively split the unit square. In each step, the current node is partitioned into four congruent quadrants (cells). We stop when each cell at the bottom level contains at most one disk center and any cell containing a disk center is surrounded by two layers of empty cells. This takes $O(n \log \Phi)$ time as the depth of the quadtree is $O(\log \Phi)$.

For a node $\nu \in Q$, we let $p(\nu)$ be the parent node of $\nu$. We denote by $|\nu|$ the diameter of the cell $b(\nu)$. For two nodes $\nu, \nu' \in Q$, we write $d(\nu, \nu')$ for the smallest distance between a point in $b(\nu)$ and a point in $b(\nu')$. For $t \geq 0$, let $D_t$ be the disk $D_t$ at time $t$. We say that $D_i^t$ occupies a node $\nu$ if (i) $p_i \in b(\nu)$; (ii) $\nu$ is a leaf or $b(\nu) \subseteq D_i^t$; and (iii) $D_i^t$ has not been eliminated yet. At each moment, each node $\nu$ is occupied by at most one disk, and we denote by $D(\nu)$ the index of disk that occupies $\nu$. If there is no such disk, we set $D(\nu) = \emptyset$. We denote by $\nu(D_i^t)$ the node of the largest cell of $Q$ that is occupied by $D_i^t$.

**Lemma 10.** Let $i \in \{2, \ldots, n\}$, and let $j \in \{1, \ldots, i - 1\}$ be the disk that eliminates $D_i$, i.e., $t_i = t(i, j)$. Then,

$$d(\nu(D_i^t), \nu(D_j^t)) \leq 2(\|\nu(D_i^t)\| + \|\nu(D_j^t)\|),$$

and

$$1/(4\Delta) \leq \|\nu(D_i^t)\| / \|\nu(D_j^t)\| \leq 4\Delta.$$

**Proof.** Let $q = \partial D_i^t \cap \partial D_j^t$. By the definition of $\nu(\cdot)$, we have $v_i t_i \leq 2\|\nu(D_i^t)\|$ and $v_j t_j \leq 2\|\nu(D_j^t)\|$. Hence, it follows that $d(q, b(\nu(D_i^t))) \leq 2\|\nu(D_i^t)\|$ and $d(q, b(\nu(D_j^t))) \leq 2\|\nu(D_j^t)\|$. This implies the first claim.

For the second claim, suppose first that $v_i \geq v_j$. In this case, $\|\nu(D_i^t)\| / \|\nu(D_j^t)\| \geq 1/4$. By construction, the leaf cell that contains $p_i$ is surrounded by empty leaf cells. Hence, the node $\nu(D_i^t)$ is not a leaf. It follows that $\|\nu(D_i^t)\| \leq 2v_i t_i$. Furthermore, regardless of whether $\nu(D_j^t)$ is a leaf or not, we have $\|\nu(D_j^t)\| \geq v_j t_j/2$. Thus,

$$1/4 \leq \|\nu(D_i^t)\| / \|\nu(D_j^t)\| \leq 2v_i t_i/(v_j t_j/2) \leq 4 \max_{i \leq j} v_i / v_j \leq 4\Delta.$$

The argument for $v_j \geq v_i$ is similar, with the roles of $D_i^t$ and $D_j^t$ reversed. Since $\Delta \geq 1$, the lemma follows. \hfill \Box

**Lemma 11.** Let $\nu \in Q$. Then, $\text{CNP}(\nu)$ has $O(\alpha)$ candidate pairs $(\nu, \nu')$ with $|\nu| \leq |\nu'|$. All the sets $\text{CNP}(\nu)$ over $\nu \in Q$ can be computed in $O(n \alpha \log \Phi)$ time.

\footnote{That is, no node is an ancestor or descendant of the other node.}
Proof. By an area packing argument, each level of $Q$ contains at most $O(1)$ candidate pairs $(\nu, \nu')$ with $|\nu| \leq |\nu'|$. Furthermore, by definition of $\Phi$ and of candidate pair, $|\nu'| = O(\min\{\Phi, \Delta\})|\nu|$, so the levels of $\nu$ and $\nu'$ in $Q$ differ by $O(\alpha)$. Thus, $\text{CNP}(\nu)$ contains $O(\alpha)$ candidate pairs $(\nu, \nu')$ with $|\nu| \leq |\nu'|$. Since $Q$ has $O(n \log \Phi)$ nodes, and since $(\nu, \nu') \in \text{CNP}(\nu)$ if and only if $(\nu', \nu) \in \text{CNP}(\nu')$, there are $O(n \alpha \log \Phi)$ candidate pairs overall.

While building $Q$, we can find all sets $\text{CNP}(\nu)$ in $O(n \alpha \log \Phi)$ time by maintaining pointers between nodes whose cells are neighboring and by traversing the cells, using these pointers when needed. \qed

Our algorithm for computing the elimination sequence of the input disks is given as Algorithm 2. We use $\tau(\nu, i)$ for the first time at which $b(\nu)$ is covered by disk $D_i$.

**Algorithm 2 Quadtree based algorithm**

1: function ANDORDER($p_1, \ldots, p_n, v_1, \ldots, v_n$)
2: $Q \leftarrow \text{ConstructQuadTree}(p_1, \ldots, p_n)$
3: CandidatePairs($Q$)
4: $D(\nu) \leftarrow \bot$ for every node $\nu$ of $Q$
5: $D(\text{root}) \leftarrow 1$
6: for $i \leftarrow 1, n$ do
7: $\nu \leftarrow \text{getLeaf}(p_i)$
8: $t_i \leftarrow \infty$
9: while $\nu \neq \text{root}$ and $t_i \geq \tau(\nu, i)$ do
10: $D(\nu) \leftarrow i$
11: for $(\nu, \nu')$ in $\text{CNP}(\nu)$ do
12: if $D(\nu') \neq \bot$ and $t_{D(\nu')} \geq t(D(\nu), D(\nu'))$ then
13: $t_i \leftarrow t(D(\nu), D(\nu'))$
14: $\nu \leftarrow p(\nu)$
15: $S \leftarrow \langle D_1, \ldots, D_n \rangle$
16: Sort $S$ using key $t_i$ for each disk $D_i$
17: return $S$

**Theorem 12.** The elimination sequence of $n$ growing disks can be computed in $O(n \alpha \log \Phi)$ time, where $\alpha = \min\{\log \Phi, \log \Delta\}$.

Proof. We can compute in $O(n \log \Phi)$ time the quadtree $Q$ with $O(n \log \Phi)$ nodes. By Lemma 14 there are $O(n \alpha \log \Phi)$ candidate pairs, which can be computed in $O(n \alpha \log \Phi)$ time.

The outer for-loop iterates over the input disks in decreasing order of priority. In the while-loop, the algorithm traverses each node $\nu \in Q$ from the leaf-node containing $p_i$ to the root. It updates $D(\nu)$ if necessary until it encounters a node $\nu$ with $t_i < \tau(\nu, i)$. The inner for-loop computes the time at which for every candidate pair in $\text{CNP}(\nu)$, the corresponding candidate pair of disks touch and updates the elimination time for $D_i$. Therefore, the algorithm takes $O(n \alpha \log \Phi)$ time. Since $\Phi = \Omega(\sqrt{n})$, this subsumes the time for the final sorting step. \qed

5.2 Using a compressed quadtree

Now we show how to improve the running time by using a compressed quadtree. Let $Q$ be the (usual) quadtree for the $n$ disk centers. This $Q$ is obtained as in the previous section, but now we stop subdividing a square once it does not contain any more disk centers. We describe how to obtain the compressed quadtree $Q_C$ from $Q$. A node $\nu$ in $Q$ is empty if $b(\nu)$ does not contain a disk-center, and non-empty otherwise. A singular path $\sigma$ in $Q$ is a path $\nu_1, \nu_2, \ldots, \nu_k$ of nodes such that (i) $\nu_k$ is a non-empty leaf or has at least two non-empty children; and (ii) for $i = 1, \ldots, k-1$, the node $\nu_{i+1}$ is the only non-empty child of $\nu_i$. We call $\sigma$ maximal if it cannot be extended by the parent of $\nu_1$ (either because $\nu_1$ is the root or because $p(\nu_1)$ has two non-empty children). For each maximal singular path $\sigma = \nu_1, \ldots, \nu_k$
in $Q$, we remove from $Q$ all proper descendants of $\nu_1$ that are not descendants of $\nu_k$, together with their incident edges. Then, we add a new compressed edge between $\nu_1$ and $\nu_k$. The resulting tree $Q_C$ has $O(n)$ nodes. Each internal node has 1 or 4 children. There are algorithms that can compute $Q_C$ in $O(n \log n)$ time \cite{[8]}. For simplicity, we assume that each disk center $p_i$ is a node of size zero, connected to the leaf of $Q_C$ containing $p_i$ by a compressed edge. A node $\nu$ from $Q$ may appear as a node in $Q_C$ or not. We let $\pi(\nu)$ be the lowest ancestor node and $\sigma(\nu)$ the highest descendant node (in both cases including $\nu$) of $\nu$ in $Q$ that appears also in $Q_C$.

For a node $\nu$ in $Q_C$, we define the set of compressed candidate pairs $\text{CNP}_C(\nu)$ for $\nu$ as

$$\text{CNP}_C(\nu) = \{(\nu, \pi(\nu)) \mid (\nu, \nu') \in \text{CNP}(\nu), |\nu| \leq |\pi(\nu)|\}.$$ 

For a pair $(\nu, \nu') \in \text{CNP}_C(\nu)$, we say $\nu$ forms the pair with $\nu'$ in $Q_C$. The following lemmas will be handy for the rest of the section.

**Lemma 13.** Let $(\nu, \nu') \in \text{CNP}(\nu)$, such that $p(\nu) \neq p(\nu')$. Then, (i) we have $(p(\nu), p(\nu')) \in \text{CNP}(p(\nu))$. Moreover, (ii) if $|\nu| \leq |\nu'|$, then $(\nu'', \nu') \in \text{CNP}(\nu'')$ for any ancestor $\nu''$ of $\nu$ with $|\nu''| \leq |\nu'|$.

**Proof.** For the first part (i), we have $d(p(\nu), p(\nu')) \leq d(\nu, \nu') \leq 2(|\nu| + |\nu'|) \leq 2(|p(\nu)| + |p(\nu')|)$ and $|p(\nu')|/|p(\nu)| = |\nu'|/|\nu|$ lies between $1/4 \Delta$ and $4 \Delta$.

For the second part (ii), we have $d(\nu'', \nu') \leq d(\nu, \nu') \leq 2(|\nu| + |\nu'|) \leq 2(|\nu''| + |\nu'|)$ and $1 \leq |\nu'|/|\nu''| \leq |\nu'|/|\nu| \leq 4 \Delta$.

**Lemma 14.** Let $\nu$ be a node of $Q$. Then, for every $(\nu, \nu') \in \text{CNP}(\nu)$, we have that $(\pi(\nu), \pi(\nu')) \in \text{CNP}_C(\pi(\nu))$ or $(\pi(\nu'), \pi(\nu)) \in \text{CNP}_C(\pi(\nu'))$.

**Proof.** First, we note that $\pi(\nu)$ and $\pi(\nu')$ are distinct, since $\nu$ and $\nu'$ are unrelated nodes in $Q$, so their least common ancestor in $Q$ must have two non-empty children. Since the lemma is symmetric in $\nu$ and $\nu'$, we may assume without loss of generality that $|\pi(\nu)| \leq |\pi(\nu')|$. We apply Lemma 13(i) repeatedly until we meet $\pi(\nu)$ or $\pi(\nu')$, whichever happens first. If we meet $\pi(\nu)$, we have $(\pi(\nu), \nu'') \in \text{CNP}(\pi(\nu))$ for some ancestor $\nu''$ of $\nu'$ in $Q$. Since $\pi(\nu)$ is encountered first, we have $\pi(\nu'') = \pi(\nu)$, so it follows that $(\pi(\nu), \pi(\nu')) \in \text{CNP}_C(\pi(\nu))$. If we meet $\pi(\nu')$, we have $(\nu'', \pi(\nu')) \in \text{CNP}(\nu'')$ for some ancestor $\nu''$ of $\nu$. Since $|\pi(\nu)| \leq |\pi(\nu')|$ and again $\pi(\nu'') = \pi(\nu)$, it follows that $(\pi(\nu), \pi(\nu')) \in \text{CNP}_C(\pi(\nu))$ by Lemma 13(ii), and thus $(\pi(\nu), \pi(\nu')) \in \text{CNP}_C(\pi(\nu'))$.

As with Lemma 14, we argue that $\text{CNP}_C(\nu)$ has $O(\alpha)$ candidate pairs. To that end, we charge each pair $(\nu, \pi(\nu')) \in \text{CNP}_C(\nu)$ to a pair $(\nu, \nu') \in \text{CNP}(\nu)$ with $|\nu| \leq |\nu'|$, such that each such pair in $\text{CNP}(\nu)$ is charged at most once. First, if $|\nu| \leq |\nu'|$, we can charge $(\nu, \pi(\nu')) \in \text{CNP}_C(\nu)$ directly to $(\nu, \nu') \in \text{CNP}(\nu)$ (in this way, we may even charge several such pairs in $\text{CNP}(\nu)$ for $(\nu, \pi(\nu'))$). Second, if $|\nu'| < |\nu|$, by Lemma 13(ii) there is an ancestor $\nu''$ of $\nu'$ with $|\nu| = |\nu''|$ and $(\nu, \nu'') \in \text{CNP}(\nu')$. Furthermore, since by definition of $\text{CNP}_C(\nu)$ we have $|\nu| \leq |\pi(\nu'|$, it follows that $\pi(\nu'') = \pi(\nu)$, so we can charge the pair $(\nu, \pi(\nu')) \in \text{CNP}_C(\nu)$ to the pair $(\nu, \nu'') \in \text{CNP}(\nu)$. It follows that there are $O(n\alpha)$ compressed candidate pairs in total. The following lemma shows how to compute $\text{CNP}_C(\nu)$ for all nodes $\nu$ in $Q_C$.

**Lemma 15.** We can compute all the sets $\text{CNP}_C(\nu)$ over $\nu \in Q_C$ in $O(n\alpha)$ total time.

**Proof.** We traverse the nodes in $Q_C$ from the root in BFS-fashion, ordered by decreasing diameter. We compute $\text{CNP}_C(\nu)$ for each node $\nu$ in order. For a node $\nu$ in $Q_C$, we put into $\text{CNP}_C(\nu)$ all pairs $(\nu, \nu') \in \text{CNP}(\nu)$ with $\nu' \in Q_C$ and $|\nu| = |\nu'|$. Furthermore, we check all pairs $(\nu, \nu')$ with $|\nu| < |\nu'|$ and (a) $(p(\nu), \nu') \in \text{CNP}_C(p(\nu))$ or (b) $(\nu', p(\nu)) \in \text{CNP}_C(\nu')$. We add $(\nu, \nu')$ to $\text{CNP}_C(\nu)$ if $(\nu, \nu')$ fulfills the requirements of a compressed candidate pair. This can be checked in $O(1)$ time. By our BFS-traversal, we already know the sets $\text{CNP}_C(p(\nu))$ and $\text{CNP}_C(\nu')$ for $|\nu| < |\nu'|$.

For $|\nu| = |\nu'|$, there are $O(1)$ pairs to check, and they can be found at the same time using appropriate pointers in $Q_C$. For $|\nu| < |\nu'|$, since $|\text{CNP}_C(p(\nu))| = O(\alpha)$, there are $O(\alpha)$ pairs to check for case (a). There can be $\omega(\alpha)$ pairs for case (b), but obviously there are $O(n\alpha)$ such pairs in total for all $\nu \in Q_C$. 

8
Now we show that the algorithm correctly computes all the compressed candidate pairs in \( \text{CNP}_C(\nu) \). Consider a pair \((\nu, \pi(\nu')) \in \text{CNP}_C(\nu)\), where \((\nu, \nu') \in \text{CNP}(\nu)\) and \(|\nu| \leq |\pi(\nu')|\). If \(|\nu| = |\pi(\nu')|\), we have \((\nu, \pi(\nu')) \in \text{CNP}(\nu)\) so the algorithm will find it. If \(|\nu| < |\pi(\nu')|\), let \(\eta\) be the parent of \(\nu\) in \(Q\). If \(\pi(\nu') = \nu'\), we have \((\eta, \pi(\nu')) \in \text{CNP}(\eta)\) by Lemma 13(i), since \(|\eta| \leq |\pi(\nu')|\). If \(|\pi(\nu')| > |\nu'|\), let \(\eta'\) be the parent of \(\nu'\) in \(Q\). Lemma 14 implies \((\eta, \eta') \in \text{CNP}(\eta)\). Since \(\pi(\eta) = \pi(\nu')\) (as a node in \(Q\) this time) and \(\pi(\eta') = \pi(\nu')\), we conclude with Lemma 13 that \((p(\nu), \pi(\nu')) \in \text{CNP}_C(p(\nu))\) or \((\pi(\nu'), p(\nu)) \in \text{CNP}_C(\pi(\nu'))\).

Recall that, in the uncompressed quadtree approach each candidate pair (of nodes) leads to a pair of disks that may touch at some time. We will call such a pair a candidate pair of disks. Note that two distinct candidate pairs may be associated to the same candidate pair of disks. Let \(D\) be the set of all candidate pairs of disks obtained using the uncompressed quadtree approach.

We set \(D_C(\nu)\) to \(D(\nu)\), if \(D(\nu) \neq \perp\). If \(D(\nu) = \perp\) and \(\nu\) has a single child \(\nu'\) connected by a compressed edge, we set \(D_C(\nu) = D(\nu')\). In all other cases, we set \(D_C(\nu) = \perp\). A compressed candidate pair \((\nu, \nu')\) for \(\nu, \nu' \in Q_C\) defines a candidate pair of disks \((D_C(\nu), D_C(\nu'))\) if both \(D_C(\nu), D_C(\nu') \neq \perp\). We let \(\mathcal{D}\) denote the set of all candidate pairs of disks defined by compressed candidate pairs. We claim that \(\mathcal{D} \subseteq D_C\). That is, even though the compressed quadtree has fewer candidate pairs of nodes, we discard only candidates that are already in \(D_C\). We first introduce a few helpful results.

**Lemma 16.** For a node \(\nu \in Q\), consider the nodes \(\sigma(\nu)\) and \(\pi(\nu)\) in \(Q_C\). For any node \(\nu'\) on the path in \(Q\) from the node corresponding to \(\sigma(\nu)\) to the node corresponding to \(\pi(\nu)\), we have \((\nu', \pi(\nu')) \in \{D(\sigma(\nu)), \perp\}\).

**Proof.** Recall that, for any node \(\eta \in Q\) we have \(D(\eta) = i\) if and only if \(D_i\) occupies \(\eta\) and \(b(\eta)\) contains \(p_i\). Since the path from \(\sigma(\nu)\) to \(\pi(\nu)\) corresponds to a compressed edge, each node along the path has only one non-empty child. Thus, the only disk that can occupy a node \(\nu'\) along the path is \(D(\sigma(\nu))\).

**Lemma 17.** \(\mathcal{D} \subseteq D_C\).

**Proof.** Let \((D(\nu), D(\nu')) \in \mathcal{D}\). Then \((\nu, \nu') \in \text{CNP}(\nu)\), and since \(D(\nu) \neq \perp\), we have \(D(\nu) = D(\sigma(\nu))\) by Lemma 16. On the other hand, we have \((\pi(\nu), \pi(\nu')) \in \text{CNP}_C(\pi(\nu))\) or \((\pi(\nu), \pi(\nu')) \in \text{CNP}_C(\pi(\nu'))\) by Lemma 13. If \(\nu \in Q_C\), \(\pi(\nu) = \nu\) and \(D_C(\pi(\nu)) = D(\nu)\). If \(\nu \notin Q_C\), then if \(D(\pi(\nu)) \neq \perp\), by Lemma 16 \(D(\pi(\nu)) = D(\sigma(\nu))\) and hence \(D_C(\pi(\nu)) = D(\nu)\). If \(D(\pi(\nu)) = \perp\), then the child node of \(\pi(\nu)\) in \(Q_C\) is \(\sigma(\nu)\), and therefore \(D_C(\pi(\nu)) = D(\nu)\). Thus, in both cases, we have \(D_C(\pi(\nu)) = D(\sigma(\nu))\), and therefore \(D_C(\pi(\nu)) = D(\nu)\). The same holds for \(\nu'\). We conclude that \((D(\nu), D(\nu')) = (D_C(\pi(\nu)), D_C(\pi(\nu'))) \in D_C\).

**Theorem 18.** The elimination sequence of \(n\) growing disks in the plane can be computed in \(O(n \log n + \alpha n)\) time, where \(\alpha = \min\{\log \Phi, \log \Delta\}\).

**Proof.** We compute the compressed quadtree for the disk centers, and we find the compressed candidate pairs. As described above, this takes \(O(n \log n + \alpha n)\) time. After that, we make the candidate pairs symmetric so that for all pairs \(\nu, \nu'\), we have \((\nu, \nu') \in \text{CNP}_C(\nu)\) if and only if \((\nu', \nu) \in \text{CNP}_C(\nu')\). This takes \(O(n \alpha n)\) time. Finally, we proceed as in Algorithm 2 but using \(Q_C\) instead of \(Q\) and the compressed candidate pairs instead of the (regular) candidate pairs. By Lemma 14 this algorithm still considers all the relevant candidate pairs of disks. The running time for the last step is proportional to the number of nodes in \(Q_C\) and the number of compressed candidates, i.e., \(O(n \alpha n)\). The total running time of the algorithm is \(O(n \log n + \alpha n)\).

6 Lower bound

We show that the elimination order can be used to sort \(n\) numbers \(v_{n+1}, \ldots, v_{2n}\) larger than 1, which implies an \(O(n \log n)\) lower bound for our problem in the algebraic decision tree model. Place \(n\) growing disks \(D_1, \ldots, D_n\) centered at points \((2, 0), (4, 0), \ldots, (2n, 0)\), all with growth rate \(v_1 = 1\). Then place \(n\) growing disks \(D_{n+1}, \ldots, D_{2n}\) centered at points \((2, 1), (4, 1), \ldots, (2n, 1)\) with growth rates \(v_{n+1}, \ldots, v_{2n}\). Observe that disk \(D_{n+i}\) will be eliminated by the corresponding \(D_i\) at \(t_{n+i} = t(n+i, i) = 1/(1+v_{n+i}) < \)
1/2 since \( t_i = 1/2 \) for \( 1 \leq i \leq n \). Then the elimination order of this set of growing disks gives the input growth rates \( \{v_{n+1}, \ldots , v_{2n}\} \) in reversed sorted order. The same argument applies to squares.

**Theorem 19.** It takes at least \( \Omega(n \log n) \) time to find the elimination order of a set of \( n \) growing disks or squares in the plane under the algebraic decision tree model.

**Acknowledgments.** This work was initiated during the 20th Korean Workshop on Computational Geometry. The authors would like to thank the other participants for interesting discussions during the workshop.

**References**

[1] Pankaj K. Agarwal, Boris Aronov, and Micha Sharir. Computing envelopes in four dimensions with applications. *SIAM J. Comput.*, 26(6):1714–1732, 1997.

[2] Daniel Bahrdt, Michael Becher, Stefan Funke, Filip Krumpe, André Nusser, Martin Seybold, and Sabine Storandt. Growing balls in \( \mathbb{R}^d \). In *Proc. 19th Workshop Algorithm Eng. Exp. (ALENEX)*, pages 247–258, 2017.

[3] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2008.

[4] Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas J. Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

[5] Michael Formann. Weighted closest pairs. In *Proc. 10th Sympos. Theoret. Aspects Comput. Sci. (STACS)*, pages 270–281, 1993.

[6] Stefan Funke, Filip Krumpe, and Sabine Storandt. Crushing disks efficiently. In *Proc. 27th Int. Workshop Comb. Alg. (IWOCA)*, pages 43–54, 2016.

[7] Stefan Funke and Sabine Storandt. Parametrized runtimes for ball tournaments. In *Proc. 33rd European Workshop Comput. Geom. (EWCG)*, pages 221–224, 2017.

[8] Sariel Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, Boston, MA, USA, 2011.

[9] John Hershberger. Finding the upper envelope of \( n \) line segments in \( O(n \log n) \) time. *Inform. Process. Lett.*, 33(4):169–174, 1989.

[10] Vladlen Koltun. Almost tight upper bounds for vertical decompositions in four dimensions. *J. ACM*, 51(5):699–730, 2004.

[11] Jiří Matoušek. *Lectures on Discrete Geometry*. Springer-Verlag, 2002.