Faster Algorithms for Growing Prioritized Disks and Rectangles*

Hee-Kap Ahn[1], Sang Won Bae[2], Jongmin Choi[1], Matias Korman[3], Wolfgang Mulzer[4], Eunjin Oh[1], Ji-won Park[5], André van Renssen[6,7], and Antoine Vigneron[8]

1 **Dept. Computer Science and Engineering, POSTECH, Pohang, Republic of Korea**
`{heekap,icothos,jin9082}@postech.ac.kr`

2 **Dept. Computer Science, Kyonggi University, Kionggi, Republic of Korea**
`swbae@kgu.ac.kr`

3 **Tohoku University, Sendai, Japan**
`mati@dais.is.tohoku.ac.jp`

4 **Institut für Informatik, Freie Universität Berlin, Berlin, Germany**
`mulzer@inf.fu-berlin.de`

5 **School of Computing, KAIST, Daejeon, Republic of Korea**
`wldnjs1727@kaist.ac.kr`

6 **National Institute of Informatics (NII), Tokyo, Japan**
`andre@nii.ac.jp`

7 **JST, ERATO, Kawarabayashi Large Graph Project**

8 **School of Electrical and Computer Engineering, UNIST, Ulsan, 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 in the plane that grow at possibly different speeds. Whenever two disks meet, the one with the higher 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. Our 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 the 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.

* The work by H.-K. Ahn, J. Choi, E. Oh was supported by the MSIT(Ministry of Science and ICT), Korea, under the SW Starlab support program(IITP–2017–0–00905) supervised by the IITP(Institute for Information & communications Technology Promotion.). S.W. Bae was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2015R1D1A1A01057220). M. K. was supported in part by KAKENHI Nos. 15H02665 and 17K12635, Japan. W. M. was supported in part by DFG Grants MU 3501/1 and MU 3501/2. J.-W. Park was supported by the NRF Grant 2011-0030044 (SRC-GAIA) funded by the Korea government (MSIP). A. v. R. was supported by JST ERATO Grant Number JPMJER1201, Japan. A. Vigneron was supported by the 2016 Research Fund (1.160054.01) of UNIST (Ulsan National Institute of Science and Technology).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems–Geometrical problems and computations

Keywords and phrases map labeling, growing disks, elimination order

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.3

1 Introduction

Suppose we are given a sequence $D_1, \dots, D_n$ of n *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 $t(i, j) > 0$ be the time when D_i and D_j touch, i.e., $t(i, j) = |p_i p_j|/(v_i + v_j)$. 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 instants 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 [7]. 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 and in which order the labels need to be discarded. Funke, Krumpe, and Storandt [7] 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 time $O(\Delta^2 n(\log n + \Delta^2))$. 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 [8] presented two further parameterized algorithms for the problem. The first algorithm runs in time $O(n \log \Delta(\log n + \Delta^{d-1}))$, 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, our 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 [6].

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 combine it with an advanced data structure for querying lower envelopes of algebraic surfaces [1, 11] and with bucketing. In particular, the algorithm runs in $O(n^{5/3+\varepsilon})$ and $O(n^{11/6+\varepsilon})$ expected time for disks and rectangles in two dimensions, respectively. These are the first subquadratic-time algorithms for the problem. 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 running time of these algorithms also depends on the *spread* Φ of the disk centers (that is, the ratio

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. Φ 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 $\mathbb{R}^2$	expected $O(n^{5/3+\varepsilon})$	Bucketing	Section 3
Rectangles in $\mathbb{R}^2$	expected $O(n^{11/6+\varepsilon})$		
$\mathcal{SA}_k$, $k \geq 4$	expected $O(n^{2-1/(2k-2)+\varepsilon})$		
Cubes in $\mathbb{R}^d$	$O(n \log^{d+2} n)$	Linearity of queries	Section 4
Disks in $\mathbb{R}^2$	$O(n \log \Phi \min\{\log \Delta, \log \Phi\})$	Quadtree	Section 5.1
Disks in $\mathbb{R}^2$	$O(n(\log n + \min\{\log \Delta, \log \Phi\}))$	Compressed quadtree	Section 5.2

of the maximum to the minimum distance between disk centers) and the ratio Δ between the fastest and slowest speed of the disks. Table 1 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 [2] that runs in $O(\Delta^2 n(\log n + \Delta^2))$ time.

Note. Parallel to our work, Castermans *et al.* [4] considered a variant of the problem for squares in the plane. Whenever two squares meet, they are replaced by a new one located at their weighted center. Like us, they are interested in the elimination/replacement sequence. Although our algorithms are slightly faster (by polylogarithmic factors) and more general (their algorithm can only handle square shapes), we emphasize that they are not comparable, since our techniques do not apply in their setting.

Notation. For any $1 \leq i \leq n$, we denote by t_i the time at which disk D_i is eliminated. Since D_1 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 $t_1, \dots, t_{i-1}$ are known.

▶ **Lemma 2.1.** *Let $i \in \{2, \dots, n\}$, and let*

$$j^* = \operatorname{argmin}_{j=1,\dots,i-1} \{t(i,j) \mid t(i,j) \leq t_j\}.$$

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. ◀

Lemma 2.1 leads to a straightforward iterative algorithm, see Algorithm 1.

Algorithm 1 A quadratic time algorithm

```
function EliminationOrder(p_1, ..., p_n, v_1, ..., v_n)
    t_1 ← ∞
    for i ← 2, n do
        t_i ← t(i, 1)
        for j ← 2, i − 1 do
            if t_j ≥ t(i, j) and t_i ≥ t(i, j) then
                t_i ← t(i, j)
    S ← (D_1, ..., D_n)
    Sort S using key t_i for each disk D_i
    return S
```

▶ **Theorem 2.2.** *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 2.1. The running time analysis is straightforward. Lemma 2.1 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. ◀

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 $B \subset \{1, \dots, n\}$ 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 2.1 shows that we can find j^* as follows

$$j^* = \text{argmin}_{j \in B}\{t(q, j) \mid t(q, j) \le t_j\}.$$

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 $f_j : \mathbb{R}^3 \to \mathbb{R}$ 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 3.1.** *Let $B \subset \{1, \dots, n\}$ 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 describe our subquadratic algorithm. Set $m = \lfloor n^{1/3} \rfloor$. We group the disks into $\lceil n/m \rceil$ *buckets* $B_1, \ldots, B_{\lceil n/m \rceil}$ such that the kth bucket B_k contains the disks $D_{(k-1)m+1}, \ldots, D_{km}$. There are $O(n^{2/3})$ buckets, each of which 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 $O(n^{1+\varepsilon})$ expected 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 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 $O(n^{5/3+\varepsilon})$ expected time.

▶ **Theorem 3.2.** *The elimination sequence of a set of n growing disks can be computed in $O(n^{5/3+\varepsilon})$ expected time 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 $\mathbb{R}^5$. Given m growing rectangles, such a data structure with query time $O(\log m)$ can be constructed in $O(m^{6+\varepsilon})$ expected time for any fixed $\varepsilon > 0$ [11]. We now apply the same approach as for growing disks, but using buckets of size $m = \lfloor n^{1/6} \rfloor$.

▶ **Theorem 3.3.** *The elimination sequence of a set of n growing rectangles can be computed in $O(n^{11/6+\varepsilon})$ expected time 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 [11] (again, for any fixed $\varepsilon > 0$). The optimal size of the buckets is $n^{1/(2k-2)}$, which gives an overall running time of $O\left(n^{\frac{4k-5}{2k-2}+\varepsilon}\right)$, which is subquadratic for any fixed $k \geq 4$.

▶ **Theorem 3.4.** *The elimination sequence of a set of n growing objects of any semi-algebraic shape, each described with $k \geq 4$ parameters can be computed in $O\left(n^{2-\frac{1}{2k-2}+\varepsilon}\right)$ expected time for any $\varepsilon > 0$.*

4 Growing cubes

Axis-aligned cubes in $\mathbb{R}^d$ are described with $d+1$ parameters. Thus, the approach from the previous section applies. 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 an almost linear 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, $\{(x, y) \in \mathbb{R}^2 \mid$

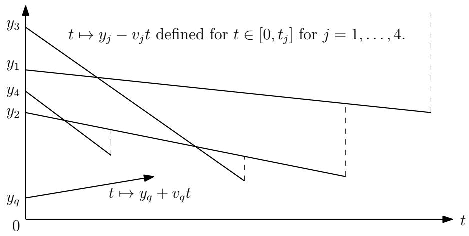

Figure 1 The lower envelope of four line segments. An elimination query for a square D_q with center (x_q, y_q) and growth rate v_q consists of shooting a ray $t \mapsto y_q + v_q t$ from below.

$y - y_i \geq |x - x_i|\}$, $\{(x, y) \in \mathbb{R}^2 \mid x - x_i \geq |y - y_i|\}$, $\{(x, y) \in \mathbb{R}^2 \mid -(y - y_i) \geq |x - x_i|\}$, 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 $B \subset \{1, \ldots, n\}$ 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 4.1.** *Let $B \subset \{1, \ldots, n\}$, $|B| = m$. 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 $t \in [0, t_j]$. See Figure 1. 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 [12]. 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 [10]. Then we build a ray-shooting data structure for this lower envelope, which takes $O(m)$ preprocessing time with $O(\log m)$ query time [5]. ◀

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

▶ **Lemma 4.2.** *Let $B \subset \{1, \ldots, n\}$, $|B| = m$. We can preprocess B in time $O(m \log^3 m)$ so that elimination queries can be answered in $O(\log^3 m)$ 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 [3], where the coordinate axes have been rotated by an angle of $\pi/4$, so that 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 4.1.

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 $D_1, \dots, D_n$, 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 ν.

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 4.2. Thus, after we have determined t_j for all $j < i$, we can find t_i in $O(\log^4 n)$ 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 4.3.** *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 Φ denote the *spread* of the disk centers and Δ denote the ratio of the growth rates, i.e., $\Phi = \max_{1 \le i < j \le n} |p_i p_j| / \min_{1 \le i < j \le n} |p_i p_j|$ and $\Delta = \max_{i \in \{1,\dots,n\}} v_i / \min_{j \in \{1,\dots,n\}} v_j$. 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 $O(n(\log n + \min\{\log \Phi, \log \Delta\}))$ time using a compressed quadtree. To simplify the notation, 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* $\mathcal{Q}$ for the disk centers. It is a rooted tree in which every internal node has four children. Each node ν of $\mathcal{Q}$ has an associated square *cell* $b(\nu)$. To obtain $\mathcal{Q}$, we recursively split the unit square. In each step, the current node is partitioned into four congruent quadrants (cells) if its corresponding cell contains one or more disk centers. We stop when each cell at the bottom level contains at most one disk center and the diameter of the cell becomes smaller than a quarter of the smallest distance between disk centers. This takes $O(n \log \Phi)$ time as the depth of the quadtree is $O(\log \Phi)$. See Figure 2 (left) for an illustration.

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

▶ **Lemma 5.1.** *Let $i \in \{2, \dots, n\}$, and let $D_j (j \in \{1, \dots, i-1\})$ be the disk that eliminates D_i, i.e., $t_i = t(i, j)$. Then,*

$$d(\nu(i, t_i), \nu(j, t_i)) \le 2(|\nu(i, t_i)| + |\nu(j, t_i)|),$$

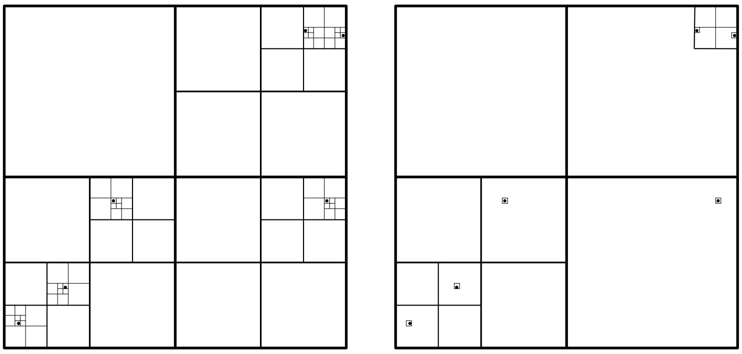

Figure 2 Obtaining a quadtree and its compressed quadtree: (left) a quadtree for 6 disk centers, where the subdivision process stops once a cell contains at most one disk center and the diameter of the cell becomes smaller than a quarter of the smallest distance between disk centers.; (right) the compressed quadtree obtained after eliminating the maximal singular paths.

and

$$1/(4\Delta) \leq |\nu(i,t_i)| \,/\, |\nu(j,t_i)| \leq 4\Delta.$$

Proof. We note three simple facts from the construction of $\mathcal{Q}$ and the definition of $\nu(\cdot,\cdot)$: (i) all non-empty leaf cells have the same diameter; (ii) for any $k \in \{1,\dots,n\}$ and $t > 0$, if $\nu(k,t)$ is not a leaf, then $|\nu(k,t)| \leq 2v_k t$; and (iii) for any $k \in \{1,\dots,n\}$ and $t \geq 0$, we have $|\nu(k,t)| \geq v_k t/2$.

For the first claim, let $q = \partial D_i^{t_i} \cap \partial D_j^{t_i}$. By fact (iii), we have $v_i t_i \leq 2\,|\nu(i,t_i)|$ and $v_j t_i \leq 2\,|\nu(j,t_i)|$. Hence, it follows that $d\,(\nu(i,t_i),\nu(j,t_i)) \leq d\,(q,\nu(i,t_i)) + d\,(q,\nu(j,t_i)) \leq v_i t_i + v_j t_i \leq 2\,|\nu(i,t_i)| + 2\,|\nu(j,t_i)|$.

Now we prove the second claim. Suppose first that $v_i \geq v_j$. If $\nu(j,t_i)$ is a leaf, $|\nu(i,t_i)| \,/\, |\nu(j,t_i)| \geq 1$, by fact (i). If $\nu(j,t_i)$ is not a leaf, it follows from facts (ii) and (iii) that

$$\frac{|\nu(i,t_i)|}{|\nu(j,t_i)|} \geq \frac{v_i t_i/2}{2v_j t_i} \geq \frac{1}{4} \geq \frac{1}{4\Delta}.$$

By construction, the leaf cell that contains p_i has diameter smaller than a quarter of the smallest distance between disk centers. Hence, the node $\nu(i,t_i)$ is not a leaf. Thus, by facts (ii) and (iii),

$$\frac{|\nu(i,t_i)|}{|\nu(j,t_i)|} \leq \frac{2v_i t_i}{v_j t_i/2} \leq \frac{4\max_i v_i}{\min_j v_j} \leq 4\Delta.$$

The argument for $v_j > v_i$ is analogous: if $\nu(i,t_i)$ is a leaf, then $|\nu(j,t_i)| \,/\, |\nu(i,t_i)| \geq 1$, by fact (i). If not, then

$$\frac{|\nu(j,t_i)|}{|\nu(i,t_i)|} \geq \frac{v_j t_i/2}{2v_i t_i} > \frac{1}{4} \geq \frac{1}{4\Delta},$$

by facts (ii) and (iii). Now, the node $\nu(j,t_i)$ cannot be a leaf, so by facts (ii) and (iii)

$$\frac{|\nu(j,t_i)|}{|\nu(i,t_i)|} \leq \frac{2v_j t_i}{v_i t_i/2} \leq \frac{4\max_j v_j}{\min_i v_i} \leq 4\Delta.$$

The lemma follows. ◀

Lemma 5.1 implies that instead of checking all disk pairs for elimination events, we can restrict ourselves to the nodes given by $\mathcal{Q}$. We say that two unrelated[1] nodes $\nu, \nu' \in \mathcal{Q}$ form a *candidate pair* if (i) $|\nu|/4\Delta \leq |\nu'| \leq 4\Delta|\nu|$ and (ii) $d(\nu, \nu') \leq 2(|\nu| + |\nu'|)$. In this case, we say that ν *forms the candidate pair* (ν, ν') with ν'. We denote by $\mathsf{CNP}(\nu)$ the set of candidate pairs formed by ν.

▶ **Lemma 5.2.** *Let $\nu \in \mathcal{Q}$. Then, $\mathsf{CNP}(\nu)$ has $O(\alpha)$ candidate pairs (ν, ν') with $|\nu| \leq |\nu'|$. All the sets $\mathsf{CNP}(\nu)$ over $\nu \in \mathcal{Q}$ can be computed in $O(n\alpha \log \Phi)$ time.*

Proof. Using a packing argument we can show that each level of $\mathcal{Q}$ contains at most $O(1)$ candidate pairs (ν, ν') that satisfy $|\nu| \leq |\nu'|$. Furthermore, by definition of Φ and of candidate pair, $|\nu'| = O(\min\{\Phi, \Delta\})|\nu|$, which implies that the levels of ν and ν' in $\mathcal{Q}$ differ by $O(\alpha)$. This implies that globally $\mathsf{CNP}(\nu)$ contains $O(\alpha)$ candidate pairs (ν, ν') with $|\nu| \leq |\nu'|$. Since $\mathcal{Q}$ has $O(n \log \Phi)$ nodes, and since $(\nu, \nu') \in \mathsf{CNP}(\nu)$ if and only if $(\nu', \nu) \in \mathsf{CNP}(\nu')$, there are $O(n\alpha \log \Phi)$ candidate pairs overall. While building $\mathcal{Q}$, we can find all sets $\mathsf{CNP}(\nu)$ 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. ◀

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

```
function EliminationOrder(p_1, ..., p_n, v_1, ..., v_n)
    Q ← ConstuctQuadTree(p_1, ..., p_n)
    CandidatePairs(Q)
    D(ν) ←⊥ for every node ν of Q
    D(root) ← 1
    for i ← 1, n do
        ν ← getLeaf(p_i)
        t_i ← ∞
        while ν ≠ root and t_i ≥ τ(ν, i) do
            D(ν) ← i
            for (ν, ν') in CNP(ν) do
                if D(ν') ≠⊥ and t_{D(ν')}, t_i ≥ t(i, D(ν')) then
                    t_i ← t(i, D(ν'))
            ν ← p(ν)
    S ← (D_1, ..., D_n)
    Sort S using key t_i for each disk D_i
    return S
```

▶ **Theorem 5.3.** *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 $\mathcal{Q}$ with $O(n \log \Phi)$ nodes. By Lemma 5.2, there are $O(n\alpha \log \Phi)$ candidate pairs, which can be found 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 $\nu \in \mathcal{Q}$ from the leaf-node containing p_i to

[1] That is, no node is an ancestor or descendant of the other node.

the root. It updates $D(\nu)$ if necessary until it encounters a node ν with $t_i < \tau(\nu, i)$. The inner **for**-loop iterates over every candidate pair (ν, ν') in $\mathsf{CNP}(\nu)$. It checks if disk $i = D_\nu$ and D'_ν have the possibility to touch by computing the time $t(i, D(\nu'))$; if so, it updates the elimination time for D_i. Thus, the algorithm takes $O(n\alpha \log \Phi)$ time. Since $\Phi = \Omega(\sqrt{n})$, this subsumes the time for the sorting step.[2] ◀

5.2 Using a compressed quadtree

Now we show how to improve the running time by using a *compressed quadtree*. Let $\mathcal{Q}$ be the (usual) quadtree for the n disk centers. The tree $\mathcal{Q}$ is obtained as in the previous section. We describe how to obtain the compressed quadtree $\mathcal{Q}_C$ from $\mathcal{Q}$. A node ν in $\mathcal{Q}$ is *empty* if $b(\nu)$ does not contain a disk-center, and *non-empty* otherwise. A *singular path* σ in $\mathcal{Q}$ is a path $\nu_1, \nu_2, \ldots, \nu_k$ of nodes such that (i) ν_k is a non-empty leaf or has at least two non-empty children; and (ii) for $i = 1, \ldots, k-1$, the node ν_{i+1} is the only non-empty child of ν_i. We call σ *maximal* if it cannot be extended by the parent of ν_1 (either because ν_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 $\mathcal{Q}$, we remove from $\mathcal{Q}$ all proper descendants of ν_1 that are not descendants of ν_k, together with their incident edges. Then, we add a new *compressed edge* between ν_1 and ν_k. The resulting tree $\mathcal{Q}_C$ has $O(n)$ nodes. Each internal node has 1 or 4 children. There are algorithms that can compute $\mathcal{Q}_C$ in $O(n \log n)$ time [9]. A node ν from $\mathcal{Q}$ may appear as a node in $\mathcal{Q}_C$ or not. We let $\pi(\nu)$ be the lowest ancestor node and $\sigma(\nu)$ the highest descendant node (in both cases including ν) of ν in $\mathcal{Q}$ that appears also in $\mathcal{Q}_C$. See Figure 2 (right) for an illustration. For a node ν in $\mathcal{Q}_C$, we define the set of *compressed candidate pairs* $\mathsf{CNP}_C(\nu)$ for ν as

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

For a pair $(\nu, \nu') \in \mathsf{CNP}_C(\nu)$, we say ν *forms the candidate pair* with ν' in $\mathcal{Q}_C$. The following lemmas will be handy for the rest of the section.

▶ **Lemma 5.4.** *Let $(\nu, \nu') \in \mathsf{CNP}(\nu)$, such that $p(\nu) \neq p(\nu')$. Then, (i) we have $(p(\nu), p(\nu')) \in \mathsf{CNP}(p(\nu))$. Moreover, (ii) if $|\nu| \leq |\nu'|$, then $(\nu'', \nu') \in \mathsf{CNP}(\nu'')$ for any ancestor ν'' of ν 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 4Δ.

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

Proof. First, we note that $\pi(\nu)$ and $\pi(\nu')$ are distinct, since ν and ν' are unrelated nodes in $\mathcal{Q}$, so their least common ancestor in $\mathcal{Q}$ must have two non-empty children. Since the lemma is symmetric in ν and ν', we may assume without loss of generality that $|\pi(\nu)| \leq |\pi(\nu')|$. We apply Lemma 5.4(i) repeatedly until we meet $\pi(\nu)$ or $\pi(\nu')$, whichever happens first. If we meet $\pi(\nu)$, we have $(\pi(\nu), \nu'') \in \mathsf{CNP}(\pi(\nu))$ for some ancestor ν'' of ν' in $\mathcal{Q}$. Since $\pi(\nu)$ is

[2] A packing argument shows that the spread of any d-dimensionsonal n-point set is $\Omega(n^{1/d})$: if any two points have distance at least 1, the point set must cover at least $\Omega(n)$ units of volume and hence must have diameter $\Omega(n^{1/d})$.

encountered first, we have $\pi(\nu'') = \pi(\nu')$, so it follows that $(\pi(\nu), \pi(\nu')) \in \mathsf{CNP}_C(\pi(\nu))$. If we meet $\pi(\nu')$, we have $(\nu'', \pi(\nu')) \in \mathsf{CNP}(\nu'')$ for some ancestor ν'' of ν. Since $|\pi(\nu)| \leq |\pi(\nu')|$ and again $\pi(\nu'') = \pi(\nu)$, it follows that $(\pi(\nu), \pi(\nu')) \in \mathsf{CNP}(\pi(\nu))$ by Lemma 5.4(ii), and thus $(\pi(\nu), \pi(\nu')) \in \mathsf{CNP}_C(\pi(\nu))$. ◀

As with Lemma 5.2, we argue that $\mathsf{CNP}_C(\nu)$ has $O(\alpha)$ candidate pairs. To that end, we charge each pair $(\nu, \pi(\nu')) \in \mathsf{CNP}_C(\nu)$ to a pair $(\nu, \nu'') \in \mathsf{CNP}(\nu)$ with $|\nu| \leq |\nu''|$, such that each such pair in $\mathsf{CNP}(\nu)$ is charged at most once. First, if $|\nu| \leq |\nu'|$, we can charge $(\nu, \pi(\nu')) \in \mathsf{CNP}_C(\nu)$ directly to $(\nu, \nu') \in \mathsf{CNP}(\nu)$ (in this way, we may even charge several such pairs in $\mathsf{CNP}(\nu)$ for $(\nu, \pi(\nu'))$). Second, if $|\nu'| < |\nu|$, by Lemma 5.4(ii) there is an ancestor ν'' of ν' with $|\nu| = |\nu''|$ and $(\nu, \nu'') \in \mathsf{CNP}(\nu)$. Furthermore, since by definition of $\mathsf{CNP}_C(\nu)$ we have $|\nu| \leq |\pi(\nu')|$, it follows that $\pi(\nu'') = \pi(\nu')$, so we can charge the pair $(\nu, \pi(\nu')) \in \mathsf{CNP}_C(\nu)$ to the pair $(\nu, \nu'') \in \mathsf{CNP}(\nu)$. It follows that there are $O(n\alpha)$ compressed candidate pairs in total. The following lemma shows how to compute $\mathsf{CNP}_C(\nu)$ for all nodes ν in $\mathcal{Q}_C$.

▶ **Lemma 5.6.** *We can compute all the sets $\mathsf{CNP}_C(\nu)$ over $\nu \in \mathcal{Q}_C$ in $O(n\alpha)$ total time.*

Proof. We traverse the nodes in $\mathcal{Q}_C$ from the root in BFS-fashion, ordered by decreasing diameter. We compute $\mathsf{CNP}_C(\nu)$ for each node ν in order. For a node ν in $\mathcal{Q}_C$, we put into $\mathsf{CNP}_C(\nu)$ all pairs $(\nu, \nu') \in \mathsf{CNP}(\nu)$ with $\nu' \in \mathcal{Q}_C$ and $|\nu| = |\nu'|$. Furthermore, we check all pairs (ν, ν') with $|\nu| < |\nu'|$ and (a) $(p(\nu), \nu') \in \mathsf{CNP}_C(p(\nu))$ or (b) $(\nu', p(\nu)) \in \mathsf{CNP}_C(\nu')$. We add (ν, ν') to $\mathsf{CNP}_C(\nu)$ if (ν, ν') 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 $\mathsf{CNP}_C(p(\nu))$ and $\mathsf{CNP}_C(\nu')$ 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 $\mathcal{Q}_C$. For $|\nu| < |\nu'|$, since $|\mathsf{CNP}_C(p(\nu))| = O(\alpha)$, 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 $\nu \in \mathcal{Q}_C$.

Now we show that the algorithm correctly computes all the compressed candidate pairs in $\mathsf{CNP}_C(\nu)$. Consider a pair $(\nu, \pi(\nu')) \in \mathsf{CNP}_C(\nu)$, where $(\nu, \nu') \in \mathsf{CNP}(\nu)$ and $|\nu| \leq |\pi(\nu')|$. If $|\nu| = |\pi(\nu')|$, we have $(\nu, \pi(\nu')) \in \mathsf{CNP}(\nu)$ so the algorithm will find it. If $|\nu| < |\pi(\nu')|$, let η be the parent of ν *in* $\mathcal{Q}$. If $\pi(\nu') = \nu'$, we have $(\eta, \pi(\nu')) \in \mathsf{CNP}(\eta)$ by Lemma 5.4(ii), since $|\eta| \leq |\pi(\nu')|$. If $|\pi(\nu')| > |\nu'|$, let η' be the parent of ν' *in* $\mathcal{Q}$. Lemma 5.4(i) implies $(\eta, \eta') \in \mathsf{CNP}(\eta)$. Since $\pi(\eta) = p(\nu)$ (as a node in $\mathcal{Q}_C$ this time) and $\pi(\eta') = \pi(\nu')$, we conclude with Lemma 5.5 that $(p(\nu), \pi(\nu')) \in \mathsf{CNP}_C(p(\nu))$ or $(\pi(\nu'), p(\nu)) \in \mathsf{CNP}_C(\pi(\nu'))$. ◀

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 $\mathcal{D}$ 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 ν has a single child ν' 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 (ν, ν') for $\nu, \nu' \in \mathcal{Q}_C$ defines a candidate pair of disks $(D_C(\nu), D_C(\nu'))$ if both $D_C(\nu), D_C(\nu') \neq \perp$. We let $\mathcal{D}_C$ denote the set of all candidate pairs of disks defined by compressed candidate pairs. We claim that $\mathcal{D} \subseteq \mathcal{D}_C$. That is, even though the compressed quadtree has fewer candidate pairs of nodes, we discard only candidates that are already in $\mathcal{D}_C$. We first introduce a helpful lemma.

▶ **Lemma 5.7.** *Let $\nu \in \mathcal{Q}$, and consider the nodes $\sigma(\nu)$ and $\pi(\nu)$ in $\mathcal{Q}_C$. If $\pi(\nu)\sigma(\nu)$ is a compressed edge, then for any node $\nu' \in \mathcal{Q}$ on the singular path for $\pi(\nu)\sigma(\nu)$, we have $D(\nu') \in \{D(\sigma(\nu)), \perp\}$.*

Proof. Recall that, for any node $\eta \in \mathcal{Q}$, we have $D(\eta) = i$ if and only if D_i occupies η and $b(\eta)$ contains p_i. Since each node ν' on the singular path has only one non-empty child, the only disk that can occupy ν' is $D(\sigma(\nu))$. ◀

▶ **Lemma 5.8.** *$\mathcal{D} \subseteq \mathcal{D}_C$.*

Proof. Let $(D(\nu), D(\nu')) \in \mathcal{D}$. If $\nu \in \mathcal{Q}_C$, $\pi(\nu) = \nu$ and $D_C(\pi(\nu)) = D(\nu)$. If $\nu \notin \mathcal{Q}_C$, then if $D(\pi(\nu)) \neq \perp$, by Lemma 5.7, $D(\pi(\nu)) = D(\sigma(\nu))$ and hence $D_C(\pi(\nu)) = D(\sigma(\nu))$. If $D(\pi(\nu)) = \perp$, then the child node of $\pi(\nu)$ in $\mathcal{Q}_C$ is $\sigma(\nu)$, and therefore $D_C(\pi(\nu)) = D(\sigma(\nu))$. Thus, in both cases, we have $D_C(\pi(\nu)) = D(\sigma(\nu))$. Since $D(\nu) \neq \perp$, we have $D(\nu) = D(\sigma(\nu))$ by Lemma 5.7, so $D_C(\pi(\nu)) = D(\nu)$. The same holds for ν'. Finally, $(\nu, \nu') \in \mathsf{CNP}(\nu)$ implies tha $(\pi(\nu), \pi(\nu')) \in \mathsf{CNP}_C(\pi(\nu))$ or $(\pi(\nu'), \pi(\nu)) \in \mathsf{CNP}_C(\pi(\nu'))$ by Lemma 5.5. We conclude that $(D(\nu), D(\nu')) = (D_C(\pi(\nu)), D_C(\pi(\nu')) \in \mathcal{D}_C$. ◀

▶ **Theorem 5.9.** *The elimination sequence of n disks can be computed in $O(n \log n + n\alpha)$ 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 $O(n \log n + n\alpha)$ time. After that, we make the candidate pairs symmetric so that for all pairs ν, ν', we have $(\nu, \nu') \in \mathsf{CNP}_C(\nu)$ if and only if $(\nu', \nu) \in \mathsf{CNP}_C(\nu')$. This takes $O(n\alpha)$ time. Finally, we proceed as in Algorithm 2, but using $\mathcal{Q}_C$ instead of $\mathcal{Q}$ and the compressed candidate pairs instead of the (regular) candidate pairs. By Lemma 5.8, 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 $\mathcal{Q}_C$ and the number of compressed candidates, i.e., $O(n\alpha)$. The total running time of the algorithm is $O(n \log n + n\alpha)$. ◀

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 and smaller than 2, which implies an $\Omega(n \log n)$ lower bound. Place n growing disks $D_1, \ldots, D_n$ centered at points $(2, 0), (4, 0), \ldots (2n, 0)$, all with growth rate $v_i = 1$. Also, place n 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 disk 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 holds for squares.

▶ **Theorem 6.1.** *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 motivating discussions.

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 Thom Castermans, Bettina Speckmann, Frank Staals, and Kevin Verbeek. Agglomerative clustering of growing squares. *CoRR*, abs/1706.10195, 2017. URL: `http://arxiv.org/abs/1706.10195`.

5 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.

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

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

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

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

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

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

12 Jiří Matoušek. *Lectures on Discrete Geometry.* Springer-Verlag, 2002.