ELSEVIER

Computational Geometry 6 (1996) 371–391

Computational Geometry
Theory and Applications

Fast algorithms for collision and proximity problems involving moving geometric objects

Prosenjit Gupta[a,1], Ravi Janardan[a,*,1], Michiel Smid[b,2]

[a] *Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, USA*
[b] *Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany*

Communicated by M. Atallah; submitted 29 April 1994; accepted 17 July 1995

Abstract

Consider a set of geometric objects, such as points, line segments, or axes-parallel hyperrectangles in $\mathbb{R}^d$, that move with constant but possibly different velocities along linear trajectories. Efficient algorithms are presented for several problems defined on such objects, such as determining whether any two objects ever collide and computing the minimum interpoint separation or minimum diameter that ever occurs. In particular, two open problems from the literature are solved: deciding in $o(n^2)$ time if there is a collision in a set of n moving points in $\mathbb{R}^2$, where the points move at constant but possibly different velocities, and the analogous problem for detecting a red–blue collision between sets of red and blue moving points. The strategy used involves reducing the given problem on moving objects to a different problem on a set of static objects, and then solving the latter problem using techniques based on sweeping, orthogonal range searching, simplex composition, and parametric search.

1. Introduction

Problems involving geometric objects that are in time-dependent motion arise in diverse applications, such as, for instance, traffic control, robotics, manufacturing, and animation, to name just a few. In such problems, we are given a collection of geometric objects, such as points, line segments, or polyhedra, along with a description of their motion, which is usually specified by a low-degree polynomial in the time parameter t. The objective is to answer questions concerning (i) properties of the objects (e.g., the closest pair) at a given time instant t or in the so-called "steady-state", i.e., at $t = \infty$; or (ii) the combinatorics of the entire motion, i.e., from $t = 0$ to $t = \infty$ (e.g., the number of topologically

* Corresponding author.

[1] The research was supported in part by NSF grant CCR-92-00270. E-mail: {pgupta,janardan}@cs.umn.edu.

[2] Supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II). E-mail: michiel@mpi-sb.mpg.de.

different Euclidean minimum spanning trees (EMSTs) determined by a set of moving points); or (iii) the existence of certain properties (e.g., collision) or computing the optimal value of some property (e.g., the smallest diameter) over the entire motion.

The systematic study of such dynamic problems was initiated by Atallah [4]. Examples of problems considered by him include computing the time intervals during which a given point appears on the convex hull of a set of moving points and determining the steady-state closest/farthest pair, EMST, and smallest enclosing circle for moving points. Fu and Lee [7,8] show how to maintain the Voronoi diagram and the EMST of moving points in the plane. In [17], Monma and Suri investigate combinatorial and algorithmic questions concerning the EMSTs that arise from the motion of one or more of the input points. In [11], Huttenlocher et al. consider the problem of computing the minimum Hausdorff distance between two rigid planar point-sets under Euclidean motion. In [10] Golin et al. solve a number of query problems on moving points, such as reporting the closest pair and the maximal points at a given query time instant t.

1.1. *Summary of results and contributions*

In this paper, we address problems of type (iii) above. Specifically, we consider sets of moving objects such as points, line segments, or axes-parallel hyperrectangles in $\mathbb{R}^d$ and are interested in questions such as: "Do two objects ever collide?" and "What is the smallest interpoint distance or smallest diameter ever attained?" Note that since our problems do not involve a query time instant, the answer is determined solely by the input, namely the initial positions of the objects, their velocities, and their trajectories. Throughout, we assume that the objects all start moving at $t = 0$, have constant, but possibly different velocities, and move along straight-line trajectories. (We say that two objects have the same (respectively different) velocities, if their velocity vectors have (respectively do not have) the same magnitude. Neither notion refers to the directions of the velocity vectors.)

Of course, the problems that we consider can be solved easily in quadratic time, by brute-force. The challenge is to do significantly better, which makes the solutions interesting and nontrivial. Table 1 summarizes our results. We note that throughout the paper, the dimension d is assumed to be a constant. The constant factors in all time bounds depend on d. Our work is focussed on optimizing the running time as a function of n.

Collision detection problems arise very often in robotics. For an overview of these problems, we refer to the book by Fujimura [9] and the references listed there. Ottmann and Wood [19] gave efficient solutions for collision detection for points moving on the real line. They also raised the open question of deciding in $o(n^2)$ time whether there is any collision in a set of n moving points in $\mathbb{R}^2$, where the points move along straight lines at constant but possibly different velocities. A related open problem raised by Atallah [4] is to decide in $o(mn)$ time whether there is a red–blue collision between a set of m red points and n blue points, all moving in the plane with different constant velocities. We answer both these questions affirmatively. We are not aware of any previous work on the closest/farthest pair questions that we address.

Our strategy for solving these dynamic problems is to reduce the problem at hand to a different problem on a set of static objects. We then solve the latter problem using techniques such as sweeping, orthogonal range searching, halfspace range searching, simplex compositions, and parametric search.

The rest of this paper is organized as follows. In Sections 2 and 3 we discuss the problem of detecting collisions and computing the minimum distance over time in point sets that are moving with the same

Table 1
Summary of results for problems on n moving objects in $\mathbb{R}^d$, $d \geqslant 2$. (As is customary, we assume throughout that d is a constant.) Each object moves with constant velocity from $t = 0$ to $t = \infty$. The velocities are either the same for all objects, or each object has a possibly different velocity. The trajectories of the objects may come from c different directions, two (or d) orthogonal directions, or may be arbitrary. All bounds are "big-oh" and worst-case. (The constant factors depend on the dimension d.) The problems that are indicated are collision detection, computing the closest distance, and minimum L_2-diameter over all times $t \geqslant 0$. The line segments have arbitrary directions, but each one moves along its supporting line

objects	dimension	# directions	velocities	problem	time
points	d	c	same	$\exists$? collision	$cn \log n$
points	2	2 orthogonal	same	closest distance	$n \log n$
points	2	2 orthogonal	different	$\exists$? collision	$n^{3/2}(\log n)^{4/3}$
points	2	arbitrary	different	$\exists$? collision	$n^{5/3}(\log n)^{6/5}$
points	2	arbitrary	different	L_2-diameter	$n \log^3 n$
boxes	d	d orthogonal	same	$\exists$? collision	$n \log^{2d-3} n$
boxes	d	d orthogonal	different	$\exists$? collision	$n^{3/2+\varepsilon}$
segments	2	arbitrary	different	$\exists$? collision	$n^{5/3+\varepsilon}$

velocity or with different velocities, respectively. In Section 4, we consider the collision detection problem for moving boxes in $\mathbb{R}^d$. In Section 5, we consider the problem of collision detection for moving line segments in the plane, where each segment moves along its supporting line. In Section 6, we consider the problem of computing the minimum L_2-diameter over time. We conclude in Section 7 with some remarks and open problems.

2. Collision detection for points moving with the same velocity

Let S be a set of n points in $\mathbb{R}^d$, where $d \geqslant 2$ is a constant. At time $t = 0$, all points start moving. We want to decide if any two points of S ever collide, and if so, we want to compute the first time instant at which a collision occurs. In this section, we assume that all points move with the same constant velocity $v > 0$. In Section 2.1, we consider the case where the trajectories are in one of c different directions. In Section 2.2, we treat the special case where the points are planar and move in two orthogonal directions. In this case, we can even compute the closest distance among the points over all times $t \geqslant 0$ efficiently.

2.1. *Points moving in c different directions*

We consider the case where the trajectory of each point is oriented in one of c different directions. Note that each trajectory is a ray.

We start with the planar case, i.e., $d = 2$. Consider one of the directions. Let $C = (x, y)$ be a coordinate system whose y-axis is parallel to this direction and whose x-axis is orthogonal to it.

We call the points of S that move in the positive y-direction *blue points*. Consider one of the other directions, and let ϕ, $-\pi/2 < \phi < \pi/2$, be the angle it makes with the positive x-axis. We call the points of S that move along this direction in such a way that their x-coordinates increase *red points*. We will solve our collision detection problem for these red and blue points. (Other combinations of directions can be handled similarly; there are O(1) such combinations.)

For any blue point b, let (b_x, b_y) be its position at time $t = 0$. Then, at time t, this point is at position $(b_x, b_y + vt)$. Similarly, the position of any red point r at time t can be written as $(r_x + vt\cos\phi, r_y + vt\sin\phi)$.

Lemma 2.1. *Let r and b be red and blue points, respectively, such that $r_x \leqslant b_x$. Let $\alpha_\phi = (1 - \sin\phi)/\cos\phi$. Then, r and b collide iff $\alpha_\phi b_x + b_y = \alpha_\phi r_x + r_y$. Moreover, if there is a collision between these points, then it takes place at time $t = (b_x - r_x)/(v\cos\phi)$.*

Proof. Assume that r and b collide. Then, there is a time t such that $b_x = r_x + vt\cos\phi$ and $b_y + vt = r_y + vt\sin\phi$. This implies that $\alpha_\phi b_x + b_y = \alpha_\phi r_x + r_y$ and $t = (b_x - r_x)/(v\cos\phi)$.

Conversely, assume that $\alpha_\phi b_x + b_y = \alpha_\phi r_x + r_y$. Then it is straightforward to verify that points r and b have the same coordinates at time $t = (b_x - r_x)/(v\cos\phi)$. Since $r_x \leqslant b_x$ and $\cos\phi > 0$, the value of t is nonnegative. Thus r and b collide. □

This lemma leads to the following algorithm. We sort the red and blue points according to their x-coordinates. Then we sweep from left to right. During this sweep, we maintain a balanced binary search tree T that stores all red points that have been visited already in its leaves. These points are sorted according to their $(\alpha_\phi r_x + r_y)$-values. Red points for which these values are equal are sorted according to their r_x-value.

If the sweep line visits a red point r, then we insert $\alpha_\phi r_x + r_y$ into T. If a blue point b is visited, then we search in T for the rightmost leaf storing the value $\alpha_\phi b_x + b_y$. If this leaf does not exist, then b does not collide with any red point. Otherwise, if r is the red point that corresponds to the value that is stored in this leaf, then r and b collide. Moreover, this point r has maximum r_x-coordinate among all red points that ever collide with b. Therefore, by Lemma 2.1, the first time at which b collides with any red point is $t_b = (b_x - r_x)/(v\cos\phi)$.

Having visited all red and blue points, we know that the minimum t_b-value computed is the first time at which there is a collision between a red and a blue point. If no t_b-value has been computed, no collision will ever take place.

Theorem 2.1. *Consider a set S of n points in the plane, where each point is moving with the same constant velocity along a ray that is oriented in one of c different directions. We can determine in $O(cn\log n)$ time and $O(n)$ space if any two points of S ever collide, and if so, the first time a collision takes place.*

Proof. We repeat the given algorithm for all pairs of different directions. It is easy to detect a collision between points that move along the same line and in opposite directions. Correctness follows from the discussion above. Let n_i be the number of points that move along the ith direction. Then our algorithm takes time proportional to

$$\sum_{i=1}^{c}\sum_{j=1}^{c}(n_i + n_j)\log n = \sum_{i=1}^{c}(cn_i + n)\log n = 2cn\log n.$$

Clearly, the algorithm uses only $O(n)$ space. □

We now generalize the result of Theorem 2.1 to the d-dimensional case, where $d > 2$. Hence, the points of S move in $\mathbb{R}^d$ and the trajectory of each point is oriented in one of c different directions.

Consider two different directions. These are determined by two vectors, say $\vec{r}$ and $\vec{s}$, that start in the origin. Let S' be the set of all points of S that move in one of these directions. Let H be the two-dimensional plane that contains the vectors $\vec{r}$ and $\vec{s}$. We define a coordinate system, as follows: two orthogonal coordinate axes lie in H; the remaining $d-2$ coordinate axes are pairwise orthogonal and they are orthogonal to H. We give each point of S' coordinates in this new coordinate system. Note that for each point of S', its first two coordinates "move" in H, whereas the last $d-2$ coordinates are fixed. Therefore, we partition the set S' into subsets such that two points belong to the same subset iff their last $d-2$ coordinates coincide. Then, in order to detect collisions in S', it suffices to detect collisions in each subset separately. But, there is a collision within one subset A iff there is a collision within the projection A_2 of A onto the first two axes. That is, it suffices to solve the collision detection problem for this subset A_2. This is a planar problem, where the trajectory of each point is oriented in one of two different directions. Hence, we can use the algorithm of Theorem 2.1 to solve the problem for A_2.

Using a similar analysis as in the proof of Theorem 2.1, we get the following result.

Theorem 2.2. *Consider a set S of n points in $\mathbb{R}^d$, $d \geqslant 2$, where each point is moving with the same constant velocity along a ray that is oriented in one of c different directions. We can determine in $O(cn\log n)$ time and $O(n)$ space if any two points of S ever collide, and if so, the first time a collision takes place.*

Remark 2.1. The result of Theorem 2.2 is optimal if c is a constant. To see this, recall the *Set Disjointness Problem*: "Decide if two sets R and B of real numbers are disjoint." In the algebraic computation tree model, this problem has an $\Omega(n\log n)$ lower bound. (See [21].) Clearly, this problem is just a special case of the red–blue collision detection problem for points on the real line all moving in the same direction with the same velocity.

Remark 2.2. If all the red points have the same constant velocity v_r and all the blue points have the same constant velocity v_b, then the same approach works. (In fact, it suffices to know just the ratio v_b/v_r.) This is because the collision condition in Lemma 2.1 becomes: given that $r_x \leqslant b_x$, points r and b collide iff

$$b_x(v_b/v_r - \sin\phi) + b_y\cos\phi = r_x(v_b/v_r - \sin\phi) + r_y\cos\phi.$$

So we store the right-handside in T for each red point r encountered and query with the left-handside whenever we see a blue point.

Remark 2.3. Our approach also works if each point p moves along a line segment, i.e., from $t = 0$ to $t = T_p$. This is because the given algorithm finds for each point p the first time instant at which a collision involving p occurs.

2.2. *The closest pair over time for points moving in two orthogonal directions*

In this section, we consider the case where each point moves in the plane, in the direction of the positive x-axis or y-axis. The points that move in the positive x-direction are called *red points*, and the points that move in the positive y-direction are called *blue points*. All points move with the same constant velocity $v > 0$.

In the previous section, we saw that in $O(n \log n)$ time, we can detect if a collision takes place between any pair of points. Here, we show that in the same amount of time, we can even compute the minimum separation over time, i.e., the smallest interpoint distance taken over all times $t \geqslant 0$.

First we consider the red points. It is clear that the distances among these points do not change over time. Therefore, the minimum separation over time among the red points is determined by the closest pair of red points at $t = 0$. Similarly for the minimum separation among the blue points. However, as we will see below, we do not have to compute the closest red–red and closest blue–blue pair separately.

It remains to consider the minimum red–blue separation. Consider any red point r and any blue point b. Their positions at time t are $r(t) = (r_x + vt, r_y)$ and $b(t) = (b_x, b_y + vt)$, respectively. The square of their Euclidean distance at time t is given by

$$Z_{rb}(t) = 2v^2t^2 + 2v\big((r_x - b_x) - (r_y - b_y)\big)t + (r_x - b_x)^2 + (r_y - b_y)^2,$$

which is a polynomial in t of degree two. Let t^*_{rb} be the time at which Z_{rb} is minimum, where $-\infty < t^*_{rb} < \infty$. We have

$$t^*_{rb} = \frac{(r_y - r_x) - (b_y - b_x)}{2v}.$$

Recall the points move from $t = 0$ to $t = \infty$. It is straightforward to verify that if $t^*_{rb} > 0$ then

$$\min_{t\geqslant 0} Z_{rb}(t) = \frac{1}{2}\big((r_x + r_y) - (b_x + b_y)\big)^2.$$

Otherwise, $\min_{t\geqslant 0} Z_{rb}(t) = (r_x - b_x)^2 + (r_y - b_y)^2$, namely the distance between r and b at $t = 0$. This suggests the following solution. First we compute the minimum red–blue distance at time $t = 0$. Then we compute the minimum red–blue separation over time among all red–blue pairs r, b for which $t^*_{rb} > 0$.

Consider computing the minimum red–blue distance at $t = 0$. Since we will eventually take the smallest of the minimum red–red, blue–blue, and red–blue separations at $t = 0$, we may as well take all red and blue points together and compute their overall minimum distance at $t = 0$, using a standard closest pair algorithm [21].

The minimum red–blue separation among all red–blue pairs r, b for which $t^*_{rb} > 0$ is computed as follows. For each red point r, we have to find among all blue points b such that $b_y - b_x \leqslant r_y - r_x$ the one for which $\frac{1}{2}((r_x + r_y) - (b_x + b_y))^2$ is minimum. Clearly, we can as well minimize $|(r_x + r_y) - (b_x + b_y)|$.

We sort all points p according to their $(p_y - p_x)$-values. Ties are broken such that blue points come first in the ordering. Then we sweep over the points in this ordering. During the sweep, we maintain all blue points that have been visited in the leaves of a balanced binary search tree T, in increasing order of their $(b_x + b_y)$-values.

If the sweep line visits a blue point b, then we insert $b_x + b_y$ into T. If a red point r is visited, then we search in T for the smallest (respectively largest) value $b'_x + b'_y$ (respectively $b''_x + b''_y$) that is at least (respectively at most) $r_x + r_y$. If $|(r_x + r_y) - (b'_x + b'_y)| < |(r_x + r_y) - (b''_x + b''_y)|$, then we know that

$$\min_{b \in T} \min_{t \geqslant 0} Z_{rb}(t) = \frac{1}{2}\left((r_x + r_y) - (b'_x + b'_y)\right)^2.$$

Otherwise, this minimum is equal to $\frac{1}{2}((r_x + r_y) - (b''_x + b''_y))^2$.

Having visited all points, we compute the smaller of the minimum overall distance at $t = 0$ and the smallest computed value $\min_b \min_t \sqrt{Z_{rb}(t)}$, taken over all points r. This gives the minimum separation over time among all points.

Theorem 2.3. *Consider a set S of n points in the plane, where each point is moving with the same constant velocity in the positive x-direction or the positive y-direction. In $O(n \log n)$ time using $O(n)$ space we can compute the closest distance between any two points over all times $t \geqslant 0$.*

Proof. The correctness of our algorithm follows from the discussion above. To prove the complexity bounds, note that we compute the minimum distance among all points at $t = 0$, sort all points by their $(p_y - p_x)$-values, perform at most n insertions into the tree T, and perform at most n binary searches in T. All this can be done in $O(n \log n)$ time using $O(n)$ space. □

Remark 2.4. Our algorithm can easily be extended such that for each point p, the closest distance to any other point over all times $t \geqslant 0$, is computed. That is, for each point p, we can compute a point $q^* \neq p$ and a time $t^*_p \geqslant 0$ such that

$$Z_{pq^*}(t^*_p) = \min_{q \neq p} \min_{t \geqslant 0} Z_{pq}(t).$$

This also takes $O(n \log n)$ time and $O(n)$ space. (In [21], it is shown how to compute for each point its nearest neighbor at time $t = 0$, in $O(n \log n)$ time and $O(n)$ space.)

3. Collision detection for points moving with different velocities

Let S be a set of n points in the plane that are moving with constant velocities. In contrast to the previous section, the velocities may be different for each point. As before, we want to decide if any two points of S ever collide.

3.1. Points moving in two orthogonal directions

We assume that the points only move in the positive x- or y-direction. The points that move in the positive x-direction are called *red points*. The other points—they move in the positive y-direction—are called *blue points*.

It is easy to detect a collision among the red points (or among the blue points) in $O(n \log n)$ time: sort all red points on the same horizontal line and scan to see if a larger-velocity point precedes a smaller-velocity point. Hence, it remains to consider red–blue collisions. Let $r(t) = (r_x + v_r t, r_y)$ be the position of the red point r at time t. Similarly, let $b(t) = (b_x, b_y + v_b t)$ be the position of the blue point b at time t.

Lemma 3.1. *Let r and b be red and blue points, respectively, such that $r_x \leqslant b_x$. Then, r and b collide iff*

$$r_x + (1/v_b)v_r r_y - (b_y/v_b)v_r = b_x.$$

Proof. Similar to the proof of Lemma 2.1. □

Let us represent each red point r of S by the point $r' = (r_x, v_r r_y, -v_r)$ in $\mathbb{R}^3$, and each blue point b of S by the plane b'' in $\mathbb{R}^3$ with equation $X + (1/v_b)Y + (b_y/v_b)Z = b_x$.

If $r_x \leqslant b_x$, then Lemma 3.1 implies that r and b collide iff point r' lies on the plane b''. Note that this plane can be written as the intersection of two halfspaces in $\mathbb{R}^3$. To solve our collision detection problem, we use the following result.

Theorem 3.1 (Matoušek [15]). *Let V be a set of n points in $\mathbb{R}^d$, let m be a parameter such that $n \leqslant m \leqslant n^d$, let h be an integer such that $1 \leqslant h \leqslant d+1$, and let $\delta > 0$ be any real number. In $O(n^{1+\delta} + m(\log n)^\delta)$ time, we can preprocess the points of V into a data structure of size $O(m)$ such that the points of V lying in the intersection of any h halfspaces can be counted in time*

$$O\left((n/m^{1/d})\left(\log\frac{m}{n}\right)^{h-(d-h+1)/d}\right).$$

We store the red points of S in the leaves of a balanced binary search tree, sorted by their r_x-values. At each internal node w of this tree, we store a data structure $D(w)$, storing the red points that are contained in the subtree of w. This structure supports the following query: given any blue point b, decide if a collision takes place between b and any red point stored in $D(w)$. By the discussion above, we can represent these red points r by the points r' in $\mathbb{R}^3$, and the query point b by the intersection of two halfspaces in $\mathbb{R}^3$. We take for $D(w)$ the data structure of Theorem 3.1 with $d = 3$, $h = 2$ and $m = n_w^{3/2}$, where n_w is the number of red points in the subtree of w.

Given this augmented binary tree, we can solve our red–blue collision detection problem, as follows. For each blue point b, we follow the path from the root of the tree to the red point with maximum r_x-value, such that $r_x \leqslant b_x$. This path defines $O(\log n)$ so-called canonical nodes, having the property that the red points stored in their subtrees partition the set of all red points r such that $r_x \leqslant b_x$. For each of these nodes w, we query the data structure $D(w)$ and count the number of red points that collide with b.

There is a red–blue collision iff for at least one blue point one of these queries gives a positive count.

Note that the data structure $D(w)$ has size $O(n_w^{3/2})$, building time $O(n_w^{3/2}(\log n)^\delta)$ and query time $O(\sqrt{n_w}(\log n)^{4/3})$. Estimating the complexity of the entire data structure level by level, we get a decreasing geometric series. Hence, the entire data structure has size $O(n^{3/2})$, can be built in time $O(n^{3/2}(\log n)^\delta)$ and has query time $O(\sqrt{n}(\log n)^{4/3})$. Since we do at most n queries, this proves the following theorem.

Theorem 3.2. *Consider a set S of n points in the plane, all moving with constant but possibly different velocities along trajectories that are parallel to the x- or y-axis. In $O(n^{3/2}(\log n)^{4/3})$ time and $O(n^{3/2})$ space, we can determine if any two points of S ever collide.*

3.2. Points moving in arbitrary directions

We now consider the case where the points move in arbitrary directions. Our approach is similar to that of the previous section. The position of any point p of S at time $t \geqslant 0$ is given by

$$p(t) = (p_x + v_{px}t, p_y + v_{py}t).$$

Here, v_{px} and v_{py} are the x- and y-components of p's velocity vector v_p, respectively.

Lemma 3.2. *Let p and q be points of S. These points collide iff*
(1) $(p_x - q_x)(v_{qy} - v_{py}) = (p_y - q_y)(v_{qx} - v_{px})$, *and*
(2) $p_x - q_x$ *and* $v_{qx} - v_{px}$ *are both less than, greater than, or equal to zero, and*
(3) $p_y - q_y$ *and* $v_{qy} - v_{py}$ *are both less than, greater than, or equal to zero.*

Proof. Assume p and q collide and let t^* be the time of collision. We consider two cases. First assume that $p_x \neq q_x$ and $p_y \neq q_y$. Thus $t^* > 0$. Then the first equation follows from the fact that $p(t)$ and $q(t)$ are equal at the time t^* of collision. Also, $t^* > 0$ implies that $(p_x - q_x)/(v_{qx} - v_{px})$ and $(p_y - q_y)/(v_{qy} - v_{py})$ are both positive, which in turn imply (2) and (3), respectively. Now consider the case where either $p_x = q_x$ or $p_y = q_y$, but not both. (We assume that all points of S are distinct at $t = 0$.) Assume w.l.o.g. that $p_x = q_x$. Since p and q collide, we must have $v_{px} = v_{qx}$. This implies (1) and (2). The fact that the time t^* of collision is positive implies (3).

To prove the converse, assume that the three conditions hold. Since p and q are distinct at $t = 0$, we know that $p_x \neq q_x$ or $p_y \neq q_y$ or both. Assume w.l.o.g. that $p_x \neq q_x$. Let $t^* = (p_x - q_x)/(v_{qx} - v_{px})$. Then (2) implies that $t^* > 0$. If $p_y \neq q_y$, then (1) implies that $p(t^*) = q(t^*)$, i.e., p and q collide. If $p_y = q_y$, then (3) implies that $v_{qy} = v_{py}$. In this case also, we have $p(t^*) = q(t^*)$. □

For any point p of S, we define the point p' in $\mathbb{R}^5$ by

$$p' = (p_x, p_y, v_{px}, v_{py}, p_x v_{py} - p_y v_{px}),$$

and for any point q in S we define the hyperplane q'' in $\mathbb{R}^5$ by

$$q'': \ -v_{qy}X_1 + v_{qx}X_2 + q_yX_3 - q_xX_4 + X_5 + (q_xv_{qy} - q_yv_{qx}) = 0.$$

Then, condition (1) of Lemma 3.2 holds iff the point p' is contained in the hyperplane q''.

We store the points of S in a five-layer data structure that is defined as follows. First, we store the points in the leaves of a balanced binary search tree, sorted by their p_x-values. Each node u in this tree contains a pointer to a balanced binary search tree that stores the same points as in u's subtree, but sorted by their v_{px}-values. Similarly, each node in this tree contains a pointer to a balanced binary search tree storing points sorted by their p_y-values. Each node in the latter tree contains a pointer to a balanced binary search tree storing points sorted by their v_{py}-values. Let w be any node of the latter "fourth-layer-tree", and let S_w be the subset of S that is stored in w's subtree. Then w stores the data structure $D(w)$ of Theorem 3.1 with $d = 5$, $h = 2$ and $m_w = n_w^{5/3}$, where $n_w = |S_w|$, for the points $\{p' \colon p \in S_w\}$.

Given this data structure, we can solve our collision detection problem, as follows. For each point q of S, we determine $\mathrm{O}(\log^4 n)$ canonical nodes of the fourth layer, for each case resulting out of combining conditions (2) and (3) of Lemma 3.2. For each of these nodes w, we query the data structure $D(w)$ and count the number of points not equal to q that collide with q.

The data structure $D(w)$ at the last layer has size $S(n_w) = \mathrm{O}(m_w)$, can be built in time $P(n_w) = \mathrm{O}(n_w^{1+\delta} + m_w(\log n)^\delta)$ and has a query time of $Q(n_w) = \mathrm{O}((n_w/m_w^{1/5})(\log n)^{6/5})$. Therefore, again considering the complexity of the entire data structure level by level in each layer, we see that it has size $\mathrm{O}(S(n))$, building time $\mathrm{O}(P(n))$ and query time $\mathrm{O}(Q(n))$. Our complete algorithm has running time

$$\mathrm{O}(P(n) + nQ(n)) = \mathrm{O}(n^{5/3}(\log n)^{6/5}).$$

We have proved Theorem 3.3.

Theorem 3.3. *Consider a set S of n points in the plane, all moving with constant but possibly different velocities. In $\mathrm{O}(n^{5/3}(\log n)^{6/5})$ time and $\mathrm{O}(n^{5/3})$ space, we can determine if any two points of S ever collide.*

Remark 3.1. The subquadratic bound provided by Theorem 3.3 solves an open problem raised by Ottmann and Wood [19]. Theorem 3.3 also holds for determining a red–blue collision between a set of n red points and a set of n blue points that move at constant but possibly different velocities. In this case, we build the data structure for the red points and query it with the blue points. This solves an open problem mentioned by Atallah [4].

Remark 3.2. We leave open the problem of finding efficiently the closest distance taken over all times $t \geqslant 0$ in a set of n points moving in the plane with constant but possibly different velocities. Viewing the trajectory of each point as a ray in xyt-space, we have to find the shortest segment parallel to the xy-plane that connects two rays.

In [20], Pellegrini gives a randomized algorithm for finding the shortest vertical segment that connects two lines in a set of n lines in 3-space. This algorithm has expected running time $\mathrm{O}(n^{8/5+\varepsilon})$. It is not clear if his technique can be extended to solve our problem since (i) we are looking for a shortest segment parallel to the xy-plane and (ii) we want $t \geqslant 0$.

4. Collision detection for orthogonally moving boxes in $d \geqslant 2$ dimensions

Let $\mathcal{B} = B_1 \cup B_2 \cup \cdots \cup B_d$ be a collection of axes-parallel rectangles in $\mathbb{R}^d$, called *d-boxes* for short. (Recall that $d \geqslant 2$ is a constant.) Each B_i contains n_i elements that move in the positive x_i-direction. The B_i's are disjoint and $\sum_{i=1}^d n_i = n$. The problem is to decide if any two d-boxes of $\mathcal{B}$ ever collide.

4.1. The equal-velocities case

Assume that all boxes move with the same constant velocity v. We consider collisions between boxes in B_1 with boxes in B_2. (Collisions between boxes in B_i with boxes in B_j, $j \neq i$, can be detected in the same way.) The boxes of B_1 and B_2 are colored red and blue, respectively. The position of any red box r is given by

$$r(t) = [r_1 + vt : r_1 + vt + L_{r1}] \times \prod_{i=2}^{d} [r_i : r_i + L_{ri}].$$

Here, $(r_1, \ldots, r_d)$ is the position of the "lower-left", i.e., lexicographically smallest corner of r at $t = 0$ and L_{ri} is the length of r along the ith dimension. Similarly, the position of any blue box b is given by

$$b(t) = [b_1 : b_1 + L_{b1}] \times [b_2 + vt : b_2 + vt + L_{b2}] \times \prod_{i=3}^{d} [b_i : b_i + L_{bi}],$$

where $(b_1, \ldots, b_d)$ is the position of the "lower-left" corner of b at $t = 0$ and L_{bi} is the length of b along the ith dimension.

Lemma 4.1. *The boxes r and b collide iff the static $(d+1)$-boxes r' and b' intersect, where*

$$r' = [r_1 + r_2 : r_1 + r_2 + L_{r1} + L_{r2}] \times [r_1 : \infty) \times (-\infty : r_2 + L_{r2}] \times \prod_{i=3}^{d} [r_i : r_i + L_{ri}]$$

and

$$b' = [b_1 + b_2 : b_1 + b_2 + L_{b1} + L_{b2}] \times (-\infty : b_1 + L_{b1}] \times [b_2 : \infty) \times \prod_{i=3}^{d} [b_i : b_i + L_{bi}].$$

Proof. It is clear that in order for r and b to collide the $(d-2)$-boxes $\prod_{i=3}^{d} [r_i : r_i + L_{ri}]$ and $\prod_{i=3}^{d} [b_i : b_i + L_{bi}]$ must intersect. Put another way, it suffices to prove the lemma for the case $d = 2$.

The 2-boxes r and b collide iff there is a $t \geqslant 0$ such that the 2-boxes $[r_1 + vt : r_1 + vt + L_{r1}] \times [r_2 : r_2 + L_{r2}]$ and $[b_1 : b_1 + L_{b1}] \times [b_2 + vt : b_2 + vt + L_{b2}]$ intersect.

Using the fact that the intervals $[\alpha : \beta]$ and $[\gamma : \delta]$ intersect iff $\gamma \leqslant \beta$ and $\delta \geqslant \alpha$, it follows that r and b collide iff there is a $t \geqslant 0$ such that $b_1 \leqslant r_1 + vt + L_{r1}$, $b_1 + L_{b1} \geqslant r_1 + vt$, $b_2 + vt \leqslant r_2 + L_{r2}$ and $b_2 + vt + L_{b2} \geqslant r_2$. This is true iff there is a $t \geqslant 0$ such that $b_1 - r_1 - L_{r1} \leqslant vt \leqslant b_1 + L_{b1} - r_1$ and $r_2 - b_2 - L_{b2} \leqslant vt \leqslant r_2 + L_{r2} - b_2$. This in turn is equivalent to $r_2 - b_2 - L_{b2} \leqslant b_1 + L_{b1} - r_1$, $b_1 + L_{b1} - r_1 \geqslant 0$, $b_1 - r_1 - L_{r1} \leqslant r_2 + L_{r2} - b_2$ and $r_2 + L_{r2} - b_2 \geqslant 0$.

Again using the fact that the intervals $[\alpha : \beta]$ and $[\gamma : \delta]$ intersect iff $\gamma \leqslant \beta$ and $\delta \geqslant \alpha$, it follows that r and b collide iff the 3-boxes $[r_1 + r_2 : r_1 + r_2 + L_{r1} + L_{r2}] \times [r_1 : \infty) \times (-\infty : r_2 + L_{r2}]$ and $[b_1 + b_2 : b_1 + b_2 + L_{b1} + L_{b2}] \times (-\infty : b_1 + L_{b1}] \times [b_2 : \infty)$ intersect. □

We map the red d-boxes r to the $(d+1)$-boxes r' and the blue d-boxes b to the $(d+1)$-boxes b'. Then our problem of detecting collisions between moving boxes in B_1 with boxes in B_2 is equivalent to detecting intersections between the static red $(d+1)$-boxes and the blue $(d+1)$-boxes.

We first solve this problem for the planar case. So, let $d = 2$. Note that $[r_1 : \infty) \times (-\infty : r_2 + L_{r2}]$ and $(-\infty : b_1 + L_{b1}] \times [b_2 : \infty)$ intersect iff the planar point $(r_1, r_2 + L_{r2})$ lies in the north–west quadrant of the point $(b_1 + L_{b1}, b_2)$.

This observation leads to the following solution. We sort the left and right endpoints of the first intervals of all boxes r' and b'. Then we sweep over them in this order. During this sweep, we maintain a priority search tree T.

If the sweep line visits a red left endpoint $r_1 + r_2$, then we insert the point $(r_1, r_2 + L_{r2})$ into T. If a red right endpoint $r_1 + r_2 + L_{r1} + L_{r2}$ is visited, then the point $(r_1, r_2 + L_{r2})$ is deleted from T. If the sweep line visits a blue left endpoint $b_1 + b_2$ or a blue right endpoint $b_1 + b_2 + L_{b1} + L_{b2}$, then we search in T for the red points that are in the north–west quadrant of the point $(b_1 + L_{b1}, b_2)$. Of course, the entire algorithm stops as soon as a collision is detected.

However, in this way, we might miss intersections between boxes r' and b' such that the first interval of r' is completely contained in the first interval of b'. Therefore, we repeat the above sweep algorithm with the roles of r and b interchanged.

It is clear that the entire algorithm correctly solves our problem and that its running time is bounded by $O(n \log n)$ and the space used is $O(n)$ since T has query and update time $O(\log n)$ and uses $O(n)$ space.

We now extend this solution to the d-dimensional case. For $3 \leqslant i \leqslant d$, the intervals $[r_i : r_i + L_{ri}]$ and $[b_i : b_i + L_{bi}]$ intersect iff $b_i \leqslant r_i + L_{ri}$ and $b_i + L_{bi} \geqslant r_i$. Therefore, we can obtain a solution for the d-dimensional problem by adding $2(d-2)$ orthogonal range restrictions (see [22]) to the solution for the planar case. Specifically, instead of using a priority search tree T as the supporting structure for the sweep, we use a structure T' which is obtained by adding the $2(d-2)$ range restrictions to T. Since each range restriction adds a $\log n$ factor to the bounds of T, it follows that the overall algorithm for red–blue collision detection takes time $O(n \log^{2d-3} n)$ and space $O(n \log^{2d-4} n)$.

Note that we also have to detect collisions among boxes in B_i. Since the distance between two boxes in B_i is the same for all time instants, it suffices to detect intersections at time $t = 0$. For $d = 2$, this problem can be solved in $O(n \log n)$ time and $O(n)$ space by a simple plane sweep algorithm. For $d > 2$, we add $2(d-2)$ orthogonal range restrictions, as above.

We summarize our result.

Theorem 4.1. *Let $\mathcal{B}$ be a collection of n boxes in $\mathbb{R}^d$, where each box moves parallel to one of the coordinate axes. If all the boxes move with the same constant velocity, then we can decide in $O(n \log^{2d-3} n)$ time and $O(n \log^{2d-4} n)$ space whether any two boxes ever collide.*

4.2. *The different-velocities case*

In this section, we consider the case where the boxes move with constant but possibly different velocities. Our solution uses the method of simplex composition, which was introduced by van Kreveld [12]. We first review this method briefly.

4.2.1. *Simplex composition*

Let $\mathcal{S}$ be a set of n geometric objects in $\mathbb{R}^d$ and let T be a data structure for some query problem on $\mathcal{S}$. Suppose that we wish now to solve our query problem not w.r.t. $\mathcal{S}$ but w.r.t. a subset $\mathcal{S}'$ of $\mathcal{S}$ that satisfies some condition. Moreover, suppose that $\mathcal{S}'$ can be specified by putting $\mathcal{S}$ in 1–1 correspondence with a set $\mathcal{P}$ of n points in $\mathbb{R}^d$ and letting $\mathcal{S}'$ correspond to the subset $\mathcal{P}'$ of $\mathcal{P}$ that is contained in some query simplex. Van Kreveld gives an efficient data structure to solve the query problem on $\mathcal{S}'$, based on combining cutting trees and partition trees. (See [13,14].) He calls his technique *a simplex composition on $\mathcal{P}$ to T*. His result is as follows. (We only state the part of his result that is relevant to us. Also, the building time stated below is not given in [12] but can be derived easily.)

Theorem 4.2 (van Kreveld [12]). *Let $\mathcal{P}$ be a set of n points in $\mathbb{R}^d$, and let S be a set of n objects in $\mathbb{R}^d$ in correspondence with $\mathcal{P}$. Let T be a data structure on S having building time $p(n)$, size $f(n)$ and query time $g(n)$. Let ε be a positive constant and let m be a parameter such that $n \leqslant m \leqslant n^d$. The application of simplex composition on $\mathcal{P}$ to T results in a data structure having building time $O(m^\varepsilon(m + p(n)))$, size $O(m^\varepsilon(m + f(n)))$ and query time $O(n^\varepsilon(g(n) + n/m^{1/d}))$.*

4.2.2. Solving the collision detection problem

We now return to the problem at hand. Again, as in Section 4.1, we consider collisions between boxes in B_1 with boxes in B_2. These boxes are colored red and blue, respectively. The position of any red box r is given by

$$r(t) = [r_1 + v_r t : r_1 + v_r t + L_{r1}] \times \prod_{i=2}^{d} [r_i : r_i + L_{ri}],$$

and the position of any blue box b is given by

$$b(t) = [b_1 : b_1 + L_{b1}] \times [b_2 + v_b t : b_2 + v_b t + L_{b2}] \times \prod_{i=3}^{d} [b_i : b_i + L_{bi}].$$

Lemma 4.2. *The boxes r and b collide iff*
(1) $v_r r_2 - v_r(b_2 + L_{b2}) - v_b(b_1 + L_{b1}) + v_b r_1 \leqslant 0$, *and*
(2) $v_r(r_2 + L_{r2}) - v_r b_2 - v_b b_1 + (r_1 + L_{r1})v_b \geqslant 0$, *and*
(3) $b_1 + L_{b1} \geqslant r_1$, *and*
(4) $r_2 + L_{r2} \geqslant b_2$, *and*
(5) $\prod_{i=3}^{d}[r_i : r_i + L_{ri}]$ *intersects* $\prod_{i=3}^{d}[b_i : b_i + L_{bi}]$.

Proof. As in Lemma 4.1, it suffices to prove the lemma for $d = 2$. The 2-boxes r and b collide iff there is a $t \geqslant 0$ such that $[r_1 + v_r t : r_1 + v_r t + L_{r1}]$ and $[b_1 : b_1 + L_{b1}]$ intersect and $[r_2 : r_2 + L_{r2}]$ and $[b_2 + v_b t : b_2 + v_b t + L_{b2}]$ intersect. This in turn is true iff there is a $t \geqslant 0$ such that $(b_1 - r_1 - L_{r1})/v_r \leqslant t \leqslant (b_1 - r_1 + L_{b1})/v_r$ and $(r_2 - b_2 - L_{b2})/v_b \leqslant t \leqslant (r_2 - b_2 + L_{r2})/v_b$. The latter two conditions are equivalent to the four conditions $(r_2 - b_2 - L_{b2})/v_b \leqslant (b_1 + L_{b1} - r_1)/v_r$, $(b_1 - r_1 - L_{r1})/v_r \leqslant (r_2 + L_{r2} - b_2)/v_b$, $b_1 + L_{b1} \geqslant r_1$ and $r_2 + L_{r2} \geqslant b_2$. □

The first two conditions of Lemma 4.2 are equivalent to saying that the point

$$b' = (-(b_2 + L_{b2}), -v_b(b_1 + L_{b1}), v_b)$$

is in the lower halfspace

$$r'\colon\ v_r X_1 + X_2 + r_1 X_3 + v_r r_2 \leqslant 0,$$

and the point

$$b'' = (-b_2, -v_b b_1, v_b)$$

is in the upper halfspace

$$r'': \; v_r X_1 + X_2 + (r_1 + L_{r1})X_3 + v_r(r_2 + L_{r2}) \geqslant 0.$$

Thus, these two conditions can be handled by doing two halfspace compositions in $\mathbb{R}^3$. Specifically, we apply Theorem 4.2 as follows. We take B_2, compute b' for each $b \in B_2$, and build the partition-tree–cutting-tree structure underlying Theorem 4.2 on these points b'. At each node v of this structure we store a secondary structure $D(v)$, which is the 3-dimensional halfspace range reporting structure given in [2]. $D(v)$ is built on a set of points b'' such that b' is in v's subtree. $D(v)$ can be trivially modified so that for any query halfspace, it returns "false" iff the halfspace contains some point of $D(v)$. Denote this structure for B_2 by T_2. We take B_1 and for each $r \in B_1$ we compute r' and r''. We query the outer structure of T_2 with r' and identify a set of canonical nodes. At each canonical node v, we query $D(v)$ with r''. We report a collision (modulo the remaining conditions (3), (4) and (5) of Lemma 4.2) iff the query on some $D(v)$ returns "false". To handle the remaining conditions of Lemma 4.2, we apply to T_2 the $2d-2$ (orthogonal) range restrictions that these conditions specify.

Let us now analyze the complexity of this solution. Consider the structure T_2. Suppose that $D(v)$ is built on s_v points and let $p(s_v)$, $f(s_v)$ and $g(s_v)$ be the preprocessing time, space and query time of $D(v)$, respectively. From [2], we have $p(s_v) = f(s_v) = O(s_v \log s_v)$ and $g(s_v) = O(\log s_v)$. Let m_2 be a parameter, where $n_2 \leqslant m_2 \leqslant n_2^3$ and let $\varepsilon > 0$ be an arbitrarily small constant. Let $P(n_2)$, $F(n_2)$ and $G(n_2)$ be the preprocessing time, space and query time of T_2, respectively. From Theorem 4.2, we get $P(n_2) = F(n_2) = O(m_2^{1+\varepsilon} + m_2^{\varepsilon} n_2 \log n_2) = O(m_2^{1+\varepsilon})$. (This follows since $m_2 \geqslant n_2$. Note that here, and subsequently, we absorb polylog factors in the ε by picking a slightly larger ε.) Also, $G(n_2) = O(n_2^{\varepsilon} \log n_2 + n_2^{1+\varepsilon}/m_2^{1/3}) = O(n_2^{1+\varepsilon}/m_2^{1/3})$. (This follows since the second term, i.e., $n_2^{1+\varepsilon}/m_2^{1/3}$, has a minimum value of n_2^{ε}, when $m_2 = n_2^3$, and this is within a logarithmic factor of the first term.) The addition of the range restrictions to T_2 contributes a $\log^{\Theta(d)} n_2$ factor to each of $P(n_2)$, $F(n_2)$ and $G(n_2)$, so that the bounds for the range-restricted T_2 are asymptotically the same as above.

Let $R(n_1, n_2)$ be the time to build the range-restricted T_2 and to query it with each of the n_1 boxes of B_1. Then $R(n_1, n_2) = P(n_2) + n_1 \cdot G(n_2) = O(m_2^{1+\varepsilon} + n_1 \cdot n_2^{1+\varepsilon}/m_2^{1/3})$. Choosing $m_2 = (n_1 \cdot n_2^{1+\varepsilon})^{1/(4/3+\varepsilon)}$ gives

$$R(n_1, n_2) = O\Big(\big(n_1 \cdot n_2^{1+\varepsilon}\big)^{(1+\varepsilon)/(4/3+\varepsilon)}\Big) = O\big((n_1 \cdot n_2)^{3/4+\varepsilon}\big) = O\big(n^{3/2+\varepsilon}\big).$$

(For simplicity, we continue to use ε here rather than introduce new constants ε', ε'', etc.) It can be verified that the total space is also $O(n^{3/2+\varepsilon})$.

In this way, we detect collisions between boxes in B_i and boxes in B_j, $j \neq i$. Collisions between boxes in B_i can be detected with the same approach and within the same time and space bounds.

Since we repeat the above for each pair B_i, B_j, the total running time is bounded by $O(d^2 n^{3/2+\varepsilon})$. By a more careful analysis, similar to the one in the proof of Theorem 2.1, we can reduce the constant d^2 to d. We summarize our result.

Theorem 4.3. *Let $\mathcal{B}$ be a collection of n boxes in $\mathbb{R}^d$, where each box moves parallel to one of the coordinate axes with a constant but possibly different velocity. We can decide in $O(n^{3/2+\varepsilon})$ time and space whether any two boxes ever collide.*

5. Collision detection for moving segments in the plane

Let S be a set of n moving segments in the plane. We assume that each segment $p \in S$ moves along its supporting line H_p with constant velocity $v_p > 0$. For simplicity, we assume that no segment is horizontal or vertical. Thus each segment has an associated upper endpoint and a lower endpoint at all times.

We treat separately the case where two segments moving along the same line of support collide. This can be done easily in $O(n \log n)$ time.

For segment $p \in S$, let $p_l(t)$ (respectively $p_u(t)$) be the lower (respectively upper) endpoint of p at time t. Let m_p denote the slope of H_p and L_p the length of p. Let $p_l(0) = (a_p, b_p)$ and $p_u(0) = (c_p, d_p)$. Then $m_p = (d_p - b_p)/(c_p - a_p)$. Two (static) segments intersect if and only if each segment intersects the line supporting the other. The motion of a segment $p \in S$ can have one of four different orientations, namely NE, NW, SE and SW. Thus when we consider pairs of segments, there are sixteen different cases that arise. We only consider those pairs of segments such that each segment has positive slope and is moving along its supporting line in a direction such that the x-coordinates of the endpoints increase. (All other cases can be similarly treated.)

For any segment $p \in S$, we denote by H_p^+ (respectively H_p^-) the halfplane above (respectively below) the line H_p. The following lemma is straightforward.

Lemma 5.1. *A pair of line segments p and q, $m_p > 0$, $m_q > 0$ and $m_p < m_q$, intersect iff $\exists t \geqslant 0$ such that $q_u(t) \in H_p^+$, $q_l(t) \in H_p^-$, $p_u(t) \in H_q^-$ and $p_l(t) \in H_q^+$.*

For segment $p \in S$, the line H_p is given by the equation: $y - b_p = m_p(x - a_p)$. We denote by v_{px} (respectively v_{py}) the components of the velocity v_p in the x- and y-directions, respectively. Then $v_{px} = v_p \cdot (c_p - a_p)/L_p$ and $v_{py} = v_p \cdot (d_p - b_p)/L_p$. Similarly, we define H_q, v_{qx} and v_{qy}.

We have

$$\begin{aligned}
&\exists t \geqslant 0 \text{ such that } q_u(t) \in H_p^+ \\
&\quad\Leftrightarrow (c_q + v_{qx}t, d_q + v_{qy}t) \in H_p^+ \\
&\quad\Leftrightarrow d_q + \frac{v_q}{L_q}(d_q - b_q)t - b_p - m_p c_q - m_p \frac{v_q}{L_q}(c_q - a_q)t + m_p a_p \geqslant 0 \\
&\quad\Leftrightarrow d_q(c_p - a_p) - c_q(d_p - b_p) \\
&\qquad + \frac{v_q}{L_q}\big((c_p - a_p)(d_q - b_q) - (c_q - a_q)(d_p - b_p)\big)t - A_p \geqslant 0, \qquad (1)
\end{aligned}$$

where $A_p = (b_p(c_p - a_p) - a_p(d_p - b_p))$. Similarly,

$$\begin{aligned}
&\exists t \geqslant 0 \text{ such that } q_l(t) \in H_p^- \\
&\quad\Leftrightarrow (a_q + v_{qx}t, b_q + v_{qy}t) \in H_p^- \\
&\quad\Leftrightarrow b_q + \frac{v_q}{L_q}(d_q - b_q)t - b_p - m_p a_q - m_p \frac{v_q}{L_q}(c_q - a_q)t + m_p a_p \leqslant 0 \\
&\quad\Leftrightarrow b_q(c_p - a_p) - a_q(d_p - b_p) \\
&\qquad + \frac{v_q}{L_q}\big((c_p - a_p)(d_q - b_q) - (c_q - a_q)(d_p - b_p)\big)t - A_p \leqslant 0. \qquad (2)
\end{aligned}$$

Conditions (1) and (2) imply that

$$\begin{aligned}
&(\exists t \geqslant 0 \text{ such that } q_u(t) \in H_p^+) \text{ and } (\exists t \geqslant 0 \text{ such that } q_l(t) \in H_p^-)\\
&\Leftrightarrow \exists t \geqslant 0 \text{ such that } \frac{L_q}{v_q}\big(A_p - d_q(c_p - a_p) + c_q(d_p - b_p)\big)\\
&\qquad\leqslant \big[(c_p - a_p)(d_q - b_q) - (c_q - a_q)(d_p - b_p)\big]t\\
&\qquad\leqslant \frac{L_q}{v_q}\big(A_p - b_q(c_p - a_p) + a_q(d_p - b_p)\big).
\end{aligned} \tag{3}$$

Similarly, we get

$$\begin{aligned}
&(\exists t \geqslant 0 \text{ such that } p_u(t) \in H_q^-) \text{ and } (\exists t \geqslant 0 \text{ such that } p_l \in H_q^+)\\
&\Leftrightarrow \exists t \geqslant 0 \text{ such that } \frac{L_p}{v_p}\big(- A_q + d_p(c_q - a_q) - c_p(d_q - b_q)\big)\\
&\qquad\leqslant \big[(c_p - a_p)(d_q - b_q) - (c_q - a_q)(d_p - b_p)\big]t\\
&\qquad\leqslant \frac{L_p}{v_p}\big(- A_q + b_p(c_q - a_q) - a_p(d_q - b_q)\big).
\end{aligned} \tag{4}$$

Assume that $m_q > m_p$. Then $[(c_p - a_p)(d_q - b_q) - (c_q - a_q)(d_p - b_p)] > 0$. Then Lemma 5.1 is satisfied iff (a) the intervals defined by inequalities (3) and (4) intersect in a non-empty interval I, and (b) $\exists t \geqslant 0$, such that $((c_p - a_p)(d_q - b_q) - (c_q - a_q)(d_p - b_p))t$ lies in I.

Condition (a) holds $\Leftrightarrow$

$$(L_p/v_p)\{ - A_q + d_p(c_q - a_q) - c_p(d_q - b_q)\} \leqslant (L_q/v_q)\{A_p - b_q(c_p - a_p) + a_q(d_p - b_p)\} \tag{5}$$

and

$$(L_q/v_q)\{A_p - d_q(c_p - a_p) + c_q(d_p - b_p)\} \leqslant (L_p/v_p)\{ - A_q + b_p(c_q - a_q) - a_p(d_q - b_q)\}, \tag{6}$$

which is equivalent to saying that the point

$$p_1' = \left(d_p, -c_p, -v_pA_p/L_p, v_p(c_p - a_p)/L_p, -\frac{v_p}{L_p}(d_p - b_p)\right)$$

in $\mathbb{R}^5$ lies in the halfspace $q_1'' \in \mathbb{R}^5$ defined as

$$\frac{v_q}{L_q}(c_q - a_q)X_1 + (d_q - b_q)\frac{v_q}{L_q}X_2 + X_3 + b_qX_4 + a_qX_5 - \frac{v_q}{L_q}A_q \leqslant 0, \tag{7}$$

and the point

$$p_2' = \left(b_p, -a_p, -v_pA_p/L_p, v_p(c_p - a_p)/L_p, -\frac{v_p}{L_p}(d_p - b_p)\right)$$

in $\mathbb{R}^5$ lies in the halfspace $q_2'' \in \mathbb{R}^5$ defined as

$$\frac{v_q}{L_q}(c_q - a_q)X_1 + (d_q - b_q)\frac{v_q}{L_q}X_2 + X_3 + d_qX_4 + c_qX_5 - \frac{v_q}{L_q}A_q \geqslant 0. \tag{8}$$

Condition (b) holds $\Leftrightarrow$

$$\frac{L_q}{v_q}\{A_p - b_q(c_p - a_p) + a_q(d_p - b_p)\} \geqslant 0 \tag{9}$$

and

$$\frac{L_p}{v_p}\{ -A_q + b_p(c_q - a_q) - a_p(d_q - b_q)\} \geqslant 0, \tag{10}$$

which is equivalent to saying that the point $p_3' = (-A_p/(c_p - a_p), -(d_p - b_p)/(c_p - a_p))$ in $\mathbb{R}^2$ lies in the halfplane $q_3' = X_1 + a_q X_2 + b_q \geqslant 0$, and the point $p_4' = (b_p, -a_p)$ in $\mathbb{R}^2$ lies in the halfplane $q_4' = (c_q - a_q)X_1 + (d_q - b_q)X_2 - A_q \geqslant 0$.

Now we apply Theorem 4.2 as in Section 4.2.

To handle conditions (7) and (8), we apply two halfspace compositions in $\mathbb{R}^5$. We build a partition-tree–cutting-tree structure D on points p_1' for $p \in S$. At each node u of this structure, we store a secondary structure which is the 5-dimensional partition-tree–cutting-tree structure on points p_2' for p such that $p_1' \in S(u)$. At each node v of $S(u)$, for each u of D, we store a 2-dimensional partition-tree–cutting-tree structure on points p_3' for p such that $p_2' \in S(v)$. For each node w at the third layer, we store a structure $D'(w)$ for halfplanar range searching [6]. This completes the definition of D. We build a binary search tree T storing the points sorted by their m_p values at the leaves. At each internal node v of T, we store an instance of D built on points at the leaves of the subtree at v. Thus the overall data structure consists of five layers.

Given a query segment $q \in S$, we can detect if q intersects any segment $p \in S$ as follows. We search in T to find $O(\log n)$ canonical nodes resulting from the condition $m_p < m_q$ in Lemma 5.1. Then for layer i, $2 \leqslant i \leqslant 5$, we query with q_{i-1}'. We repeat this for each $q \in S$. A collision between two segments p and q with $m_p = m_q$ can be detected easily in $O(n \log n)$ time since we must have $H_p = H_q$ in this case.

The time and space bounds follow from repeated application of Theorem 4.2. For parameter m satisfying $n \leqslant m \leqslant n^5$, the total time taken is $O(m^{1+\varepsilon} + n \cdot n^{1+\varepsilon}/m^{1/5})$. Choosing $m = n^{(2+\varepsilon)/(6/5+\varepsilon)}$, we get a total time of $O(n^{5/3+\varepsilon})$. The total space is also $O(n^{5/3+\varepsilon})$.

Theorem 5.1. *Given a set of n line segments in the plane, where the segments move along their lines of support with constant but possibly different velocities, we can decide in $O(n^{5/3+\varepsilon})$ time and space, whether any two segments ever collide.*

6. Computing the minimum L_2-diameter over all times $t \geqslant 0$

Let S be a set of n points in the plane that are moving at constant but possibly different velocities. The diameter of the points at time t is the largest Euclidean distance among all pairs of points at time t. In this section, we consider the problem of computing the minimum L_2-diameter of S over all times $t \geqslant 0$. We will solve this problem using Megiddo's parametric search technique. (See [1,16,18].) First we review this paradigm.

6.1. Parametric search

The parametric search technique is a powerful tool for solving efficiently a variety of optimization problems. Suppose we have a decision problem $\mathcal{P}(t)$ that receives as input n data items and a real parameter t. Assume that $\mathcal{P}$ is monotone, meaning that if $\mathcal{P}(t_0)$ is true for some t_0, then $\mathcal{P}(t)$ is also true for all $t < t_0$. Our aim is to find the maximum value of t for which $\mathcal{P}(t)$ is true. We denote this value by t^*.

Assume we have a sequential algorithm A_s that, given the n data items and t, decides if $\mathcal{P}(t)$ is true or not. The control flow of this algorithm is governed by comparisons, each of which involves testing the sign of some low-degree polynomial in t. Let T_s and C_s denote the running time and the number of comparisons made by algorithm A_s, respectively. Note that by running A_s on input t, we can decide if $t \leqslant t^*$ or $t > t^*$: we have $t \leqslant t^*$ iff $\mathcal{P}(t)$ is true.

The parametric search technique simulates A_s generically on the unknown critical value t^*. Whenever A_s reaches a branching point that depends on a comparison operation, the comparison can be reduced to testing the sign of a suitable low-degree polynomial $f(t)$ at $t = t^*$. The algorithm computes the roots of this polynomial and checks each root a to see if it is less than or equal to t^*. In this way, the algorithm identifies two successive roots between which t^* must lie and thus determines the sign of $f(t^*)$. In this way we get an interval I in which t^* can possibly lie. Also the comparison now being resolved, the generic execution can proceed. As we proceed through the execution, each comparison that we resolve results in constraining I further and we get a sequence of progressively smaller intervals each known to contain t^*. The generic simulation (since it is able to correctly resolve each comparison at each branching point in its execution) will run to completion and we are left with an interval I that contains t^*. It can be shown that for any real number $r \in I$, $\mathcal{P}(r)$ is true. Therefore, t^* must be the right endpoint of I.

Since A_s makes at most C_s comparisons during its execution, the entire simulation and, hence, the computation of t^* take $O(C_sT_s)$ time. To speed up this algorithm, Megiddo replaces A_s by a parallel algorithm A_p that uses P processors and runs in T_p parallel time. At each parallel step, let A_p make a maximum of W_p independent comparisons. Then our algorithm simulates A_p sequentially, again at the unknown value t^*. At each parallel step, we get at most W_p low-degree polynomials in t. We compute the roots of all of them and do a binary search among them using repeated median finding to make the probes for t^*. For each probe, we run the sequential algorithm A_s. In this way, we get the correct sign of each polynomial in t^*, and our algorithm can simulate the next parallel step of A_p.

For the simulation of each parallel step, we spend $O(W_p)$ time for median finding. Hence, the entire simulation of this step takes time $O(W_p + T_s \log W_p)$. As a result, the entire algorithm computes t^* in time $O(W_pT_p + T_sT_p \log W_p)$. Since $W_p \leqslant P$, the running time is bounded by $O(PT_p + T_sT_p \log P)$.

6.2. Applying parametric search

Let t^* be the time at which the Euclidean diameter of S is minimum. If t^* is not unique, then we take the largest possible value.

The position of any point p of S is given by $p(t) = (p_x + v_{px}t, p_y + v_{py}t)$. Let $Z_{pq}(t)$ denote the square of the Euclidean distance between the points p and q. Note that $Z_{pq}(t)$ is a polynomial of degree two. Let t^*_{pq} be the time at which p and q are closest. Again, if t^*_{pq} is not unique, then we take the largest possible value. (Recall that we only consider time instants that are nonnegative. Also, if p and q have the same velocity vectors, then their distance is invariant over time, and we have $t^*_{pq} = \infty$.) Finally, let $D(t)$ denote the diameter of S at time t.

Lemma 6.1. *Let $t \geqslant 0$. Then $t > t^*$ iff $t > t^*_{pq}$ for all points $p, q \in S$ such that $Z_{pq}(t) = D^2(t)$.*

Proof. Let $t > t^*$ and assume there are $p, q \in S$ such that $Z_{pq}(t) = D^2(t)$, but $t \leqslant t^*_{pq}$. Since Z_{pq} is non-increasing as t increases in the interval $[0 : t^*_{pq}]$, we have $Z_{pq}(t^*) \geqslant Z_{pq}(t)$. But $Z_{pq}(t) = D^2(t)$, which is strictly larger than $D^2(t^*)$, because $t > t^*$ and because of how we have chosen t^* as the largest time instant at which the diameter is minimum. Hence, $Z_{pq}(t^*) > D^2(t^*)$, a contradiction.

To prove the converse, assume that $t > t^*_{pq}$ for all $p, q \in S$ such that $Z_{pq}(t) = D^2(t)$. By our choice of t^*_{pq} as the largest time instant at which Z_{pq} is minimum, it follows that $Z_{pq}(t)$ is strictly increasing as t increases in the interval $[t^*_{pq} : \infty)$. Now, for any $t' > t$, we have

$$D^2(t') \geqslant Z_{pq}(t') > Z_{pq}(t) = D^2(t).$$

Therefore $t^* \leqslant t$. Let ε be a positive real number. For ε sufficiently small, there is a pair p, q such that $Z_{pq}(t) = D^2(t)$ and $Z_{pq}(t-\varepsilon) = D^2(t-\varepsilon)$. Note that since $t > t^*_{pq}$, we have $t - \varepsilon > t^*_{pq}$. Thus $Z_{pq}(t-\varepsilon) < Z_{pq}(t)$, i.e., $D^2(t-\varepsilon) < D^2(t)$. Therefore, $t^* < t$. □

Remark 6.1. Lemma 6.1 also holds if there are points whose separation is invariant over time, since the proof does not assume the absence of such points. Alternatively, this can be seen from the following argument. Let p, q be two such points. Let $[t_1, t_2]$ be a time interval such that for any $t \in [t_1, t_2]$ we have $Z_{pq} = D^2(t)$. Since $D(t^*)$ is the minimum diameter, we have $D(t) \geqslant D(t^*)$. Moreover, since Z_{pq} is invariant over time, we have $D^2(t^*) \geqslant Z_{pq}(t^*) = Z_{pq}(t) = D^2(t)$. It follows that $D(t) = D(t^*)$ and hence by the definition of t^* we have $t^* \geqslant t_2$.

Also, by definition of t^*_{pq}, we have $t^*_{pq} = \infty$. Hence, if $t \in [t_1, t_2]$, then we have $t \leqslant t^*_{pq}$. Lemma 6.1 says that then $t \leqslant t^*$, which is true.

The decision problem $\mathcal{P}(t)$ we need to solve is as follows. Given n points in the plane, all moving with constant but possibly different velocities and a real number $t \geqslant 0$, decide if $t > t^*_{pq}$ for all points $p, q \in S$ such that $Z_{pq}(t) = D^2(t)$.

Clearly, $\mathcal{P}(t)$ is monotone (with "true" and "false" interchanged) and by Lemma 6.1 t^* is the maximum t for which $\mathcal{P}(t)$ is false. Thus, in order to apply parametric search, we need an algorithm that finds all pairs $p, q \in S$ that achieve the diameter at time t, and checks for each such pair if $t > t^*_{pq}$, i.e., if t is to the right of the lowest point of the parabola segment Z_{pq}.

The sequential algorithm A_s does the following. It first computes the position of the points at time t. Then it computes all pairs $p, q \in S$ such that $Z_{pq}(t) = D^2(t)$. (See [21].) Finally, for each such pair, it checks if $t > t^*_{pq}$. All this can be done in $T_s = \mathrm{O}(n \log n)$ time. (Note that there are only $\mathrm{O}(n)$ diametral pairs.)

For the parallel algorithm A_p, we use the parallel version of the above algorithm as given in Akl and Lyons [3]. This version uses $P = n$ processors and runs in $T_p = \mathrm{O}(\log n)$ parallel time on a CREW PRAM.

We run the parametric search with these algorithms. Having found t^*, we run A_s once more with this value to get the diameter at time t^*. Substituting the values for P, T_s and T_p, we conclude that the running time of the entire algorithm is bounded by $\mathrm{O}(n \log^3 n)$.

Theorem 6.1. *Consider a set S of n points in the plane, all moving with constant but possibly different velocities. In $\mathrm{O}(n \log^3 n)$ time, we can compute the minimum L_2-diameter of S taken over all times $t \geqslant 0$.*

7. Concluding remarks and open problems

We have given several techniques to solve collision detection and distance problems on moving objects. Our main strategy was to reduce the problem to a problem for other objects that do not move and then to solve the latter by known techniques.

A number of interesting open problems remain. First, can some of our bounds be improved? E.g., can we compute the minimum L_2-diameter over time of n moving planar points in $O(n \log n)$ time? Can we extend our solution to three dimensions? Note that in our approach, we need a parallel algorithm for finding all diametral pairs at a given time t. Fortunately, in 3-space, there are only $O(n)$ such pairs and, moreover, there is an efficient algorithm to compute them [5]. What is lacking is an efficient parallel algorithm. In dimensions higher than three, our approach will not be efficient because the number of diametral pairs is $\Theta(n^2)$. (See [21, pp. 182–183].)

Another open problem is to extend our solution of Section 2.2 for computing the minimum interpoint distance over time to the case where the points move with constant but possibly different velocities and directions. (See also Remark 3.2.) Finally, can we generalize our approach to detect collisions in a set of moving simplices?

Acknowledgements

The authors would like to thank the referee for several suggestions that helped improve the presentation.

References

[1] P.K. Agarwal, M. Sharir and S. Toledo, Applications of parametric searching in geometric optimization, J. Algorithms 17 (1994) 292–318.

[2] A. Aggarwal, M. Hansen and T. Leighton, Solving query-retrieval problems by compacting Voronoi diagrams, Proc. 22nd ACM Symp. Theory Comput. (1990) 331–340.

[3] S.G. Akl and K.A. Lyons, Parallel Computational Geometry (Prentice-Hall, Englewood Cliffs, 1993).

[4] M.J. Atallah, Some dynamic computational geometry problems, Comput. Math. Appl. 11 (1985) 1171–1181.

[5] B. Chazelle, H. Edelsbrunner, L. Guibas and M. Sharir, Diameter, width, closest line pair, and parametric searching, Discrete Comput. Geom. 10 (1993) 183–196.

[6] B. Chazelle, L.J. Guibas and D.T. Lee, The power of geometric duality, BIT 25 (1985) 76–90.

[7] J.-J. Fu and R.C.T. Lee, Voronoi diagrams of moving points in the plane, Internat. J. Comput. Geom. Appl. 1 (1991) 23–32.

[8] J.-J. Fu and R.C.T. Lee, Minimum spanning trees of moving points in the plane, IEEE Trans. Comput. 40 (1991) 113–118.

[9] K. Fujimura, Motion Planning in Dynamic Environments (Springer, Berlin, 1991).

[10] M.J. Golin, C. Schwarz and M. Smid, Further dynamic computational geometry, Proc. 4th Canad. Conf. Comput. Geom. (1992) 154–159.

[11] D.P. Huttenlocher, K. Kedem and J.M. Kleinberg, On dynamic Voronoi diagrams and the minimum Hausdorff distance for point sets under Euclidean motion in the plane, Proc. 8th ACM Symp. Comput. Geom. (1992) 110–119.

[12] M. van Kreveld, New results on data structures in computational geometry, Ph.D. Thesis, University of Utrecht, The Netherlands (1992).
[13] J. Matoušek, Cutting hyperplane arrangements, Discrete Comput. Geom. 6 (1991) 385–406.
[14] J. Matoušek, Efficient partition trees, Discrete Comput. Geom. 8 (1992) 315–334.
[15] J. Matoušek, Range searching with efficient hierarchical cuttings, Discrete Comput. Geom. 10 (1993) 157–182.
[16] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, J. ACM 30 (1983) 852–865.
[17] C.L. Monma and S. Suri, Transitions in geometric minimum spanning trees, Proc. 7th ACM Symp. Comput. Geom. (1991) 239–249.
[18] K. Mulmuley, Computational Geometry: An Introduction Through Randomized Algorithms (Prentice-Hall, Englewood Cliffs, 1994).
[19] T. Ottmann and D. Wood, Dynamic sets of points, Comput. Vision Graphics Image Processing 27 (1984) 157–166.
[20] M. Pellegrini, Incidence and nearest-neighbor problems for lines in 3-space, Proc. 8th ACM Symp. Comput. Geom. (1992) 130–137.
[21] F.P. Preparata and M.I. Shamos, Computational Geometry, An Introduction (Springer, New York, 1988).
[22] D.E. Willard and G.S. Lueker, Adding range restriction capability to dynamic data structures, J. ACM 32 (1985) 597–617.