Discrete Comput Geom 25:591–610 (2001)
DOI: 10.1007/s00454-001-0015-1

Discrete & Computational
Geometry

Kinetic Connectivity for Unit Disks*

L. Guibas,[1] J. Hershberger,[2] S. Suri,[3] and L. Zhang[1†]

[1]Computer Science Department, Stanford University,
Stanford, CA 94305, USA
guibas@cs.stanford.edu

[2]Mentor Graphics Corp., 8005 SW Boeckman Road,
Wilsonville, OR 97070, USA
john_hershberger@mentor.com

[3]Computer Science Department, Engineering I,
Room 2106, University of California,
Santa Barbara, CA 93106, USA
suri@cs.ucsb.edu

Abstract. We describe a kinetic data structure (KDS) that maintains the connected components of the union of a set of unit-radius disks moving in the plane. We assume that the motion of each disk can be specified by a low-degree algebraic trajectory; this trajectory, however, can be modified in an on-line fashion. While the disks move continuously, their connectivity changes at discrete times. Our main result is an $O(n)$ space data structure that takes $O(\log n/\log\log n)$ time per connectivity query of the form "are disks A and B in the same connected component?" A straightforward approach based on dynamically maintaining the overlap graph requires $\Omega(n^2)$ space. Our data structure requires only linear space and must deal with $O(n^{2+\varepsilon})$ updates in the worst case, each requiring $O(\log^2 n)$ amortized time, for any $\varepsilon > 0$. This number of updates is close to optimal, since a set of n moving unit disks can undergo $\Omega(n^2)$ connectivity changes.

1. Introduction

Motivated by applications in mobile communication and ad hoc networks, we study a basic geometric problem in the plane: kinetic connectivity of unit disks. Specifically,

* A preliminary version of this paper appeared in the *Proceedings of the ACM Symposium on Computational Geometry*, 2000. Work by Leonidas Guibas and Li Zhang was supported in part by National Science Foundation Grant CCR-9623851, by US Army MURI Grant DAAH04-96-1-0007, and by a Grant from Intel Corporation. Subhash Suri's research was partially supported by NSF Grant CCR-9901958.

† Current address: Compaq Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, USA. l.zhang@compaq.com.

given a set of unit radius disks moving independently in the plane, we want a *compact* data structure that can support efficient queries of the following form: "Are disks A and B in the same connected component?" (Two disks A and B are connected if either A and B overlap, or if there is a sequence of disks $A = D_0, D_1, \ldots, D_k = B$ such that D_i overlaps D_{i+1}, for all $i = 0, 1, \ldots, k-1$.) The problem of determining the connected components of the union of *stationary* disks, or other geometric shapes, has been well studied in computational geometry, and several efficient algorithms are known [2], [7], [9]. In this paper, however, we are interested in maintaining the connected components of the disks as they move around in the plane.

One obvious approach to dealing with motion is to use discrete sampling: we can take snapshots of the disks at regular intervals, and recompute the connected components at each instant. This simple approach tends to be very inefficient because sampling must be fine enough so as not to miss any critical changes in connectivity. Recomputing the connectivity from scratch at each instant also seems wasteful. Instead, we use the kinetic data structure (KDS) framework [1], which is an event-driven data structure. Rather than updating the connectivity at regular, fixed time intervals, a KDS responds to certain "critical" geometric conditions or events and is guaranteed to discover all connectivity changes. We show that our KDS for connectivity has many desirable properties: it has linear size, supports fast connectivity queries, and requires roughly quadratically many updates, each of which takes poly-logarithmic time.

We assume that each disk in the set has a published *flight plan*—a specification of its future motion, at least in the short term. We make no assumption about the motion except that it is described by some low-degree algebraic curve so that, for any two disks, we can determine in constant time when they meet or separate for the first time. The flight plan of a disk can also change at any time, as long as the KDS is notified of this change.

Maintaining the connectivity of moving disks has potential applications in ad hoc mobile networks [8], [12]. Briefly, an ad hoc network consists of a set of mobile hosts (or devices) in which peer-to-peer communication occurs without the use of base stations. Each host has its own unique IP address, and it transmits its position through a beacon. Two hosts within each other's transmission radius can communicate. The hosts themselves act like "mobile routers," and cooperatively forward messages not addressed to them. In order for such ad hoc networks to work correctly, at the very least, each host needs to know the names of all other reachable hosts. If we treat the communication range of each host as a unit-radius disk, the reachability problem is precisely our disk connectivity problem. Due to the "light-weight" nature of mobile hosts, it is important to keep the computation load and memory requirement to a minimum. Our KDS achieves both of these goals.

Another application of disk (or ball) connectivity arises in maintaining group communication in military operations. During reconnaissance missions, a set of military personnel or airplanes must remain in radio contact throughout the mission, even as the individual members of the group move independently. A break in the group communication is considered a fatal error, requiring a termination of the mission. Again, the connectivity of disks can be used to model this problem. Connectivity of moving shapes is also relevant to the study of biological processes in which coalitions of moving entities are determined by contact, and we want to track the evolution of coalitions

over time. Examples include computer animation of bird flocks where proximity determines subgroups and flight behavior, and cellular automata viewed at a macroscopic scale.

The number of connected components is a zero-order topological invariant of the shape defined by the union of the unit disks—it corresponds to the the zero Betti number of the shape. Higher order Betti numbers, such as the number of holes, may also be of interest. Edelsbrunner [3], [4] defines the *nerve* of a union of disks to be a natural simplicial complex associated with the disks. The nerve is a topological retract of the union of disks and thus has the same topological invariants. It can be readily extracted from the Delaunay triangulation of the disk centers. A Delaunay triangulation of a set of disks can easily be kinetized, but unfortunately no subcubic bound is known for the number of changes the Delaunay triangulation undergoes under motion.

The connectivity problem can be readily cast as a dynamic graph problem: model each disk by a node, and put an edge between two nodes if their disks overlap. As disks move, some edges are added or deleted. Using the flight paths of the disks, we can maintain a priority queue of times when the overlap between disks changes (either two disjoint disks meet, or two overlapping disks separate). We can use the dynamic graph data structure of Holm et al. [6] to maintain this *overlap graph*. This data structure supports edge insertions or deletions in $O(\log^2 n)$ amortized time, and connectivity queries in $O(\log n / \log\log n)$ time.

The chief drawback of using the overlap graph is that it can have $\Omega(n^2)$ edges in the worst case, and therefore it is not compact. The main contribution of this paper is a linear-size spanning subgraph of the overlap graph that can be maintained efficiently as the disks move. We use the dynamic structure of [6] to maintain the connectivity of our graph. Our structure is *efficient* in the KDS sense: if the disks move algebraically, the worst-case number of connectivity changes is $\Theta(n^2)$, and the worst-case number of changes to our graph is only $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$.

Maintaining the connectivity of disks appears to be quite different from maintaining the connectivity of moving rectangles [5]. In the case of rectangles, $O(n)$ cycles on the boundary of the union of rectangles contain a rectangle vertex. Rectangle adjacencies along these cycles give a linear-size spanning graph, which is maintained kinetically as the rectangles move. Finding a similar spanning graph for disks is quite a bit more complicated. In the case of rectangles, all the geometric primitives can be implemented using standard data structures such as interval and segment trees. In the case of disks, we need new structures, called shadows and shadow diagrams.

2. Preliminaries

Let S be a collection of n points moving in the plane. Each point is the center of a unit disk. We are interested in maintaining the connected components of these disks as the points move.

Two unit disks are *connected* if they overlap (that is, the separation of their centers is less than 2), or if they belong to the same connected component of the transitive closure of the "overlap" relation. We say that two points of S are connected if and only if their unit disks are connected.

We denote by $UDisk(p)$ the unit disk centered on a point p, and use $UDisk(S)$ to denote the union of the unit disks centered on points of S. Because most of the discussion below uses disks of radius 2, we use the simpler notation $D(p)$ and $D(S)$ to denote the corresponding concepts for radius-2 disks. In general, we use the following function-style notation for the Minkowski sum: if X is a shape with a reference point, then $X(p)$ is a translate of X with the reference point at p, and $X(P) = \bigcup_{p \in P} X(p)$ for any point set P. In other words, $X(p) = X + p$ and $X(P) = X + P$, where "+" denotes the Minkowski sum.

As part of our data structure we use sorted lists that support searching, insertion, deletion, splitting, and concatenation in $O(\log n)$ time per operation. Such lists are typically implemented as balanced binary trees [2] or skip lists [10]. In the remainder of this paper (particularly Section 5.1) we simply use the term *sorted list* to refer to such a data structure.

3. A Spanning Graph for Static Points

In this section we define a linear-size multigraph $\mathcal{G}$ on the points of S whose connectivity is the same as that induced by $UDisk(S)$. For each point p, $\mathcal{G}$ has up to eight edges connecting p to other points $q \in S$ that lie to its right and such that $UDisk(p) \cap UDisk(q) \neq \emptyset$.

It is convenient for us to convert the symmetrical "overlap" relation between unit disks into an asymmetrical containment relation between radius-2 disks and points. For any two points p and q, $UDisk(p) \cap UDisk(q) \neq \emptyset$ if and only if p is contained in the radius-2 disk $D(q)$.

To help define the graph $\mathcal{G}$, we select eight subsets of the points of S lying to the right of $p = (x_p, y_p)$. See Fig. 1. For each $i \in \{0, 1\}$ and $j \in \{0, 1, 2, 3\}$, define a *box set*

$$\square_{i,j}(p) = S \cap \{(x, y) \mid x_p + i < x \leq x_p + 1 + i, \\ y_p - 2 + j < y \leq y_p - 1 + j\}.$$

Lemma 3.1. *For any set $A = \square_{i,j}(p)$, the disks of $UDisk(A)$ form at most one connected component.*

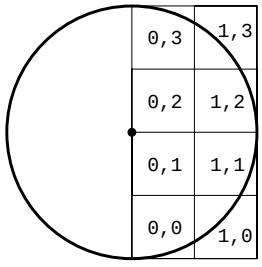

Fig. 1. Cover the right half of the disk with eight squares.

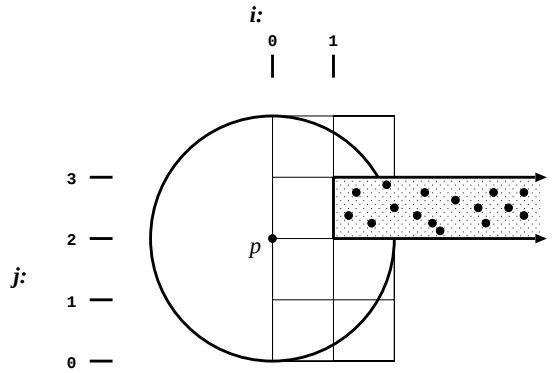

Fig. 2. The slab defining $\sqsubset_{1,2}(p)$ is shown shaded.

Proof. The claim follows from the fact that the intersection of any two unit disks with centers in A is nonempty. The set A is determined by the intersection of a unit square with S, and hence the maximum separation between any two points of A is $\sqrt{2}$. □

If $q \in S$ lies to the right of p and $D(q)$ contains p, then $q \in \square_{i,j}(p)$, for some $i \in \{0, 1\}$, $j \in \{0, 1, 2, 3\}$. (The right half of $D(p)$ is contained in the union of the eight unit squares that define the $\square_{i,j}(p)$ sets.) The preceding lemma implies that we can connect p to all connected components lying to its right by adding at most eight edges to the graph: one to the connected component of each $UDisk(\square_{i,j}(p))$.

For convenience of computation (see Section 4), we do not actually separate the points right of p into unit squares. Instead, we assign them to nondisjoint, semi-infinite slabs

$$S \cap \left\{(x, y) \mid x_p + i < x,\ y_p - 2 + j < y \leq y_p - 1 + j\right\},$$

for $i \in \{0, 1\}$ and $j \in \{0, 1, 2, 3\}$. We identify these *slab sets* by their indices (i, j), using the notation $\sqsubset_{i,j}(p)$. See Fig. 2. The edges of $\mathcal{G}$ from p to points of S to its right are defined as follows:

> For each set of points $\sqsubset_{i,j}(p)$, if $p \in D(\sqsubset_{i,j}(p))$, put into $\mathcal{G}$ an edge from p to the point $q \in \sqsubset_{i,j}(p)$ such that $D(q)$ has the leftmost intersection with the horizontal line $y = y_p$. Define $Target(p, i, j) \equiv q$. $Target(p, i, j)$ does not exist if $\sqsubset_{i,j}(p)$ is empty or $p \notin D(\sqsubset_{i,j}(p))$.

Note that $Target(p, 0, j)$ may be equal to $Target(p, 1, j)$, which is why $\mathcal{G}$ is a multigraph instead of a graph. This is because $\sqsubset_{0,j}(p)$ contains $\sqsubset_{1,j}(p)$. In fact, $\sqsubset_{0,j}(p) = \square_{0,j}(p) \cup \sqsubset_{1,j}(p)$, and even if $\square_{0,j}(p)$ is nonempty, $Target(p, 0, j)$ may belong to $\sqsubset_{1,j}(p)$. See Fig. 3.

Lemma 3.2. *For each p, $i \in \{0, 1\}$, and $j \in \{0, 1, 2, 3\}$, the point $Target(p, i, j)$ belongs to the connected component of $\square_{i,j}(p)$.*

Proof. We prove the lemma for $i = 1$ and $i = 0$ separately. The case $i = 1$ is easier, so we handle it first.

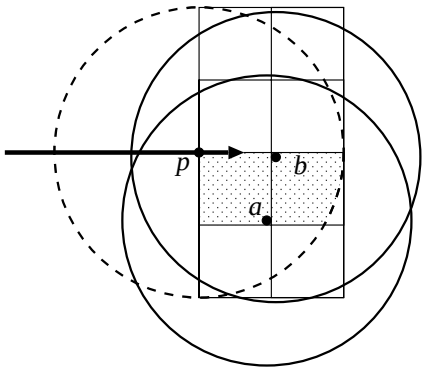

Fig. 3. *Target*$(p, 0, 1) = b$ belongs to $\sqsubset_{1,1}(p)$, even though $\square_{0,1}(p)$ is nonempty.

If $q \equiv$ *Target*$(p, 1, j)$ exists, then q must belong to $\square_{1,j}(p)$, the set contained in the unit square at the left end of $\sqsubset_{1,j}(p)$, since points to the right of $\square_{1,j}(p)$ are outside $D(p)$. There is only one component containing points of $\square_{1,j}(p)$, by Lemma 3.1, and so the lemma is true for $i = 1$.

Because $\sqsubset_{1,j}(p) \subseteq \sqsubset_{0,j}(p)$, we need to be sure that *Target*$(p, 0, j)$ is either in $\square_{0,j}(p)$ or connected to it. We consider the cases $j \in \{0, 3\}$ and $j \in \{1, 2\}$ separately.

First, consider the case $j = 0$ ($j = 3$ is symmetric). The portion of $D(p)$ inside the slab $\{(x, y) \mid x_p < x,\ y_p - 2 < y \leq y_p - 1\}$ has diameter exactly 2. See Fig. 4(a). (The diameter is determined by the lower left and upper right points of the region. However, the upper right point is the intersection of $D(p)$ with the radius-2 disk centered at $(x_p, y_p - 2)$, which shows that the diameter is 2.) All the points of $(\square_{0,0}(p) \cup \square_{1,0}(p)) \cap D(p)$ belong

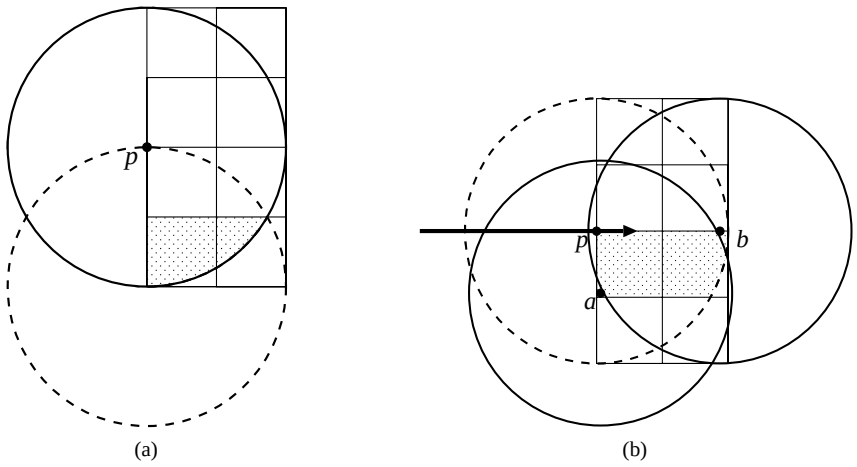

Fig. 4. (a) The diameter of $(\square_{0,0}(p) \cup \square_{1,0}(p)) \cap D(p)$ is at most 2. (b) Points a and b belong to different connected components; the horizontal line through p intersects $D(a)$ left of its intersection with $D(b)$.

to the same connected component, and hence even if *Target*$(p, 0, 0)$ belongs to $\square_{1,0}(p)$, it is still connected to $\square_{0,0}(p)$.[1]

Second, consider the case $j = 1$ ($j = 2$ is symmetric). The diameter of the portion of $D(p)$ inside the slab $\{(x, y) \mid x_p < x,\ y_p - 1 < y \leq y_p\}$ is larger than 2. Hence there may be two connected components of points inside that region. If there is only one component, then the lemma is true; therefore, suppose that there are two components. Let $a \in \square_{0,1}(p)$ and $b \in \square_{1,1}(p)$ belong to different components. See Fig. 4(b). Because the y-coordinates of a and b differ by at most 1, their x-coordinates differ by at least $\sqrt{3}$. The leftmost point of $D(b)$ is at most $2-\sqrt{3}$ to the left of a. Likewise the left intersection of $y = y_p$ with $D(a)$ is at least $\sqrt{3}$ to the left of a. Hence $y = y_p$ intersects $D(a)$ at least $2\sqrt{3} - 2$ to the left of its intersection with $D(b)$. That is, *Target*$(p, 0, 1) \in \square_{0,1}(p)$, which proves the lemma. □

Theorem 3.3. *The multigraph $\mathcal{G}$ has the same connected components as the connectivity graph of UDisk(S).*

Proof. By induction, it is sufficient to show that each point $p \in S$ is connected in $\mathcal{G}$ to all the connected components determined by points to the right of p and inside $D(p)$. The points right of p that are directly connected to p (i.e., that lie inside $D(p)$) are contained in $\square_{i,j}(p)$ for $i \in \{0, 1\}$, $j \in \{0, 1, 2, 3\}$. By Lemma 3.1, the points of each such set belong to a single connected component. The graph $\mathcal{G}$ contains edges from p to each *Target*(p, i, j); by Lemma 3.2, p is therefore connected to all components determined by points lying right of p and inside $D(p)$. □

4. Geometric Structures

In this section we describe geometric structures from which we can extract the spanning graph $\mathcal{G}$. We describe these structures for a static point set S; Section 5 shows how to maintain the structures as the points move.

We slice each disk $D(p)$ horizontally into four unit-height fragments, identified as $F_j(p)$, for $j \in \{0, 1, 2, 3\}$. Let $p = (x_p, y_p)$. Then

$$F_j(p) = D(p) \cap \{(x, y) \mid y_p - j + 1 \leq y < y_p - j + 2\}.$$

Note that the disk fragments are numbered top to bottom, while the unit squares are numbered bottom to top. (This is intentional, as Lemma 4.1 will reveal.) See Fig. 5. As above, if P is a set of points, we use the notation $F_j(P)$ to denote the union of disk fragments defined by the disks centered at points in P. We introduce these disk fragments because containment by a fragment corresponds to intersection with a disk whose center lies in a particular slab. This is made precise in the lemma below.

[1] Because *Target*$(p, 1, j)$ always belongs to the same connected component as *Target*$(p, 0, j)$, for $j \in \{0, 3\}$, we could reduce the size of $\mathcal{G}$ by eliminating the edges to *Target*$(p, 1, 0)$ and *Target*$(p, 1, 3)$ from each $p \in S$. However, this optimization makes no asymptotic difference, and so for uniformity of description we retain those edges in the rest of the paper.

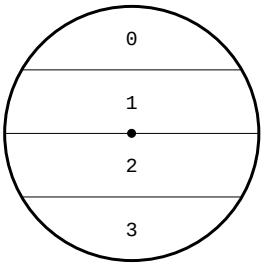

Fig. 5. Divide the disk into four horizontal slices.

As an extension of the notations $\square_{i,j}(p)$ and $\sqsubset_{i,j}(p)$ used in Section 3, we introduce the notation $\Xi_i(p)$ to denote the half-plane set $S \cap \{(x, y) \mid x_p + i < x\}$, for $i \in \{0, 1\}$. That is, $\Xi_i(p)$ is the subset of S lying more than distance i to the right of p. See Fig. 6.

Lemma 4.1. *Point p is contained in $D(\sqsubset_{i,j}(p))$ (that is, Target(p, i, j) is non-null) if and only if p is contained in $F_j(\Xi_i(p))$.*

Proof. See Fig. 7. Consider a point $q \in \sqsubset_{i,j}(p)$ such that $p \in D(q)$. Let $q = (x_q, y_q)$, and recall that $p = (x_p, y_p)$. Then $y_p + j - 2 < y_q \leq y_p + j - 1$, by definition of $\sqsubset_{i,j}(p)$. Equivalently, $y_q - j + 1 \leq y_p < y_q - j + 2$. That is, $p \in F_j(q)$. Because $\Xi_i(p) \supseteq \sqsubset_{i,j}(p)$, we have established the forward direction of the lemma. The argument to prove the reverse direction is similar. □

Because we are interested in the containment of a point p by disks whose centers lie to the right of p, we do not have to worry about the right boundaries of those disks. We can replace each disk by its *shadow*, which is the set of all points lying in the disk or directly to its right. In general, the *shadow* of a planar region Q is the Minkowski sum of Q with the non-negative x-axis,

$$\mathit{Shadow}(Q) = \{(x, y) \mid \exists (x', y) \in Q, x' \leq x\}.$$

Lemma 4.2. *Point p is contained in $F_j(\Xi_i(p))$ if and only if p is contained in $\mathit{Shadow}(F_j(\Xi_i(p)))$.*

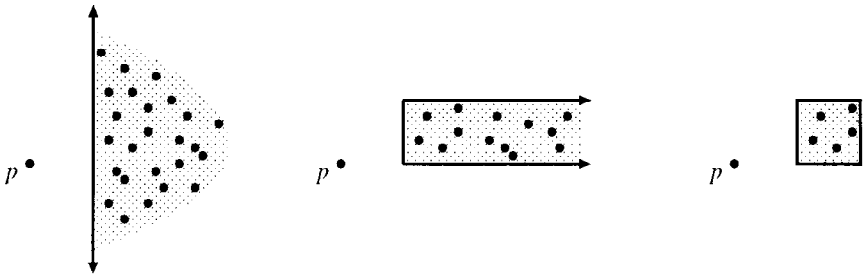

Fig. 6. The sets $\Xi_1(p)$, $\sqsubset_{1,2}(p)$, and $\square_{1,2}(p)$.

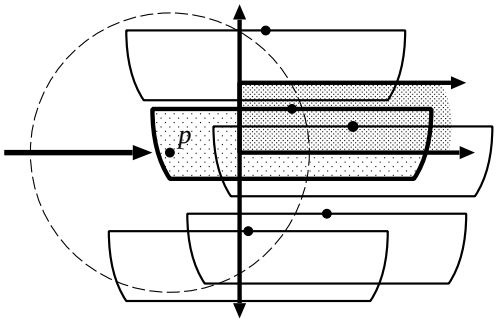

Fig. 7. The slab of $\sqsubset_{1,2}(p)$ is shown darkly shaded, and the copy of F_2 corresponding to *Target*$(p, 1, 2)$ is lightly shaded. The only copies of F_2 that contain p and have centers in the half-plane set $\Xi_1(p)$ are those with centers in the slab set $\sqsubset_{1,2}(p)$.

Proof. See Fig. 8. Because $F_j(\Xi_i(p)) \subset$ *Shadow*$(F_j(\Xi_i(p)))$, the lemma can fail to be true only if p lies strictly to the right of $F_j(\Xi_i(p))$. However, for any $q \in \Xi_i(p)$ such that p lies to the right of $F_j(q)$, p also lies to the right of $D(q)$, which implies that p is right of q, a contradiction. □

It follows that we can determine *Target*(p, i, j) by intersecting the boundary of *Shadow*$(F_j(\Xi_i(p)))$ with the horizontal line $y = y_p$. If the intersection occurs to the left of p, then *Target*(p, i, j) is the point q whose fragment $F_j(q)$ contributes the boundary arc on which the intersection lies. Otherwise *Target*(p, i, j) is null.

To compute and maintain the graph $\mathcal{G}$ efficiently, we need a geometric structure that represents *Shadow*$(F_j(\Xi_i(p)))$ for all $p \in S$, $i \in \{0, 1\}$, and $j \in \{0, 1, 2, 3\}$. We use a variant of the *maxima diagram* used by Basch et al. in their KDS for maintaining the closest pair of points in the plane [1].

Let C be any convex shape with constant complexity in the plane, such that the intersection of C with a horizontal line, or the intersection of two translated copies of C, can be computed in constant time. Furthermore, if the copies of C are translating algebraically as a function of t, it must be possible to compute in constant time the value of t at which a triple intersection occurs for three copies of C, or two copies and a horizontal line. We assume that C has a reference position (the origin in its own frame of reference), and that $C(p)$ denotes a copy of C placed with its reference position at p. We use $C \in \{F_0, F_1, F_2, F_3\}$ for our KDS, but we describe the geometric structure in terms of a general convex shape C.

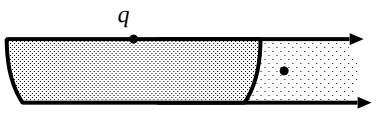

Fig. 8. $F_2(q)$ is darkly shaded; its shadow is the entire shaded area. Because $F_2(q)$ is left–right symmetric about q, a point to the right of $F_2(q)$ is also to the right of q.

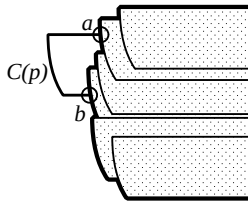

Fig. 9. *Max*(C, x_p) is shown dark. The feet of *Shadow*$(C(p))$ are circled. The shadow diagram records all the feet of *Shadow*$(C(p))$ for all $p \in S$.

For a given x-coordinate $\bar{x}$, let *Max*$(C, \bar{x})$ represent the left envelope of all placements of C at points of S to the right of $\bar{x}$. That is, if $S_{>\bar{x}}$ denotes all the points of S with x-coordinates greater than $\bar{x}$, then *Max*$(C, \bar{x})$ is ∂*Shadow*$(C(S_{>\bar{x}}))$, the boundary of *Shadow*$(C(S_{>\bar{x}}))$. This boundary *Max*$(C, \bar{x})$ consists of horizontal edges and edges of translated copies of C. We cannot afford to compute *Max*$(C, \bar{x})$ explicitly for each $\bar{x}$, as the total size of the shadows can be $\Omega(n^2)$ for n points. Instead, we represent the different *Max*$(C, \bar{x})$'s in a diagram with linear size, as is done for unions of wedges in [1].

We can compute *Max*(C, x) for all values of x by a right-to-left sweep over the points of S. Our linear-size data structure records the history of that sweep. During the sweep, *Max*(C, x) changes only when $x = x_p$ for some point $p = (x_p, y_p)$, $p \in S$. If we maintain the edges of *Max*(C, x) for the current value of x in a balanced binary tree, we can update the tree in logarithmic time when we sweep over each point of S. An edge of *Shadow*$(C(p))$ replaces a sequence of edges in *Max*(C, x_p).

For a given point $p = (x_p, y_p) \in S$, we define the *feet* of the shadow *Shadow*$(C(p))$ to be the edges of *Max*(C, x_p) intersected by ∂*Shadow*$(C(p))$. We represent the structure of *Max*(C, x) for all x in a shadow diagram *ShadowDiagram*(C), which records the feet of *Shadow*$(C(p))$ for each $p \in S$. The shadow diagram also includes a vertical line at $x = \infty$, so that the vertical order of y-disjoint shadows is recorded by their intersections with the vertical line. The shadow diagram, along with the x-order of S, completely characterizes the combinatorial structure of *Max*(C, x) for all x, as can be easily seen by right-to-left induction on S. See Fig. 9.

During the sweep construction of the shadow diagram, we can answer horizontal ray shooting queries on the current version of *Max*(C, x) by binary search on the sorted list that represents *Max*(C, x). These ray shooting queries are what we need to determine *Target*(p, i, j) when C is chosen to be one of the F_j.

5. Kinetic Data Structures

We are interested in maintaining *ShadowDiagram*(C) as the points of S move about. We compute the structure initially by plane sweep.

In this section we fix C to be one of the F_j, and largely omit it from the notation. We replace *Shadow*$(C(p))$ by *Shadow*(p), and *Max*(C, x) by *Max*(x).

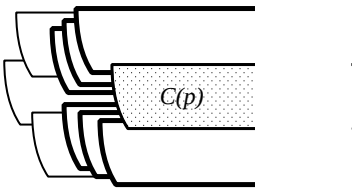

Fig. 10. Elements of *Parents*(p) are shown dark.

5.1. *Support Data Structures*

For each point $p = (x_p, y_p) \in S$, we record the portion of *Max*(x_p) that is hidden by *Shadow*(p), storing the sequence of edges of different $C(q)$ in a sorted list that supports logarithmic-time operations [2], [10]. The edge sequence is represented as a sequence of points of $S_{>x_p}$, with the edges to be computed on demand. We refer to this list as *Cover*(p), the *cover list of* p, because the edge sequence is covered by *Shadow*(p). In Fig. 9, the portion of *Max*(x_p) between the circled feet of *Shadow*(p) is *Cover*(p).

Consider what happens to a particular shadow *Shadow*(p) during the sweep construction of the shadow diagram. When the sweep passes $x = x_p$, an edge of *Shadow*(p) is added to *Max*(x_p), covering some other edges. As the sweep progresses, newly added shadows cover more and more of *Shadow*(p). (Note that at most a single edge of *Shadow*(p) appears on *Max*(x) for any value of x.) Whenever the remaining edge of *Shadow*(p) is partially covered during the process, the newly added shadow has one foot on *Shadow*(p).

We store a sorted list of all the shadows that have *Shadow*(p) as one foot—these are *Shadow*(q) for certain points $q \in S$ left of p. We denote this list by *Parents*(p). As with the list *Cover*(p), we represent the members of *Parents*(p) implicitly as a sequence of points of $S_{<x_p}$, stored in the order their shadow edges intersect ∂*Shadow*(p). See Fig. 10.

5.2. *Certificates*

Two kinds of certificates are needed to maintain the shadow diagram, *x-order* certificates and *arc order* certificates.

The x-order certificates maintain the x-sorted order of the points in S. For two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$, the certificate *x-order*(p, q) holds if and only if $x_p < x_q$. We create x-order certificates based on the initial sorted order of the points of S, then maintain them as the points' x-order changes.

The arc order certificates maintain the intersection order of triples of shadow edges. For every pair of consecutive elements a and b of *Parents*(p), we maintain a certificate *arc-order*(a, b, p) that holds if and only if ∂*Shadow*(a) intersects ∂*Shadow*(p) clockwise of ∂*Shadow*(b). (Note that ∂*Shadow*(p) and ∂*Shadow*(q) intersect in at most one point, for any p and q.) We also maintain arc order certificates for the ends of each shadow edge: if the upper end of ∂*Shadow*(p) terminates on ∂*Shadow*(a) (so $p \in$ *Parents*(a)), and b is the uppermost element of *Parents*(p), then we maintain a certificate *arc-order*(a, b, p).

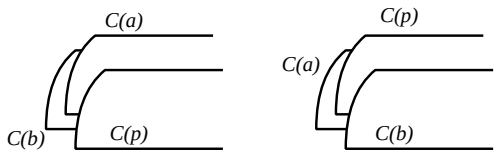

Fig. 11. Two types of arc order certificates, denoted *arc-order*(a, b, p) in both cases.

We maintain a similar certificate for the bottom end of each shadow edge. Generally speaking, we define and maintain an arc order certificate for every pair of feet that appear consecutively along some shadow in the shadow diagram. See Fig. 11. As a special case, we treat the vertical line at $x = \infty$ as a shadow boundary: we maintain arc order certificates for the shadow edges with feet on that vertical line (i.e., shadow edges that do not terminate on any other shadow).

Lemma 5.1. *Let* (X, A) *be the* x*-order and arc order certificates determined by some shadow diagram ShadowDiagram*(C)*. If the points of* S *move continuously, but all the certificates in* (X, A) *remain true, then the combinatorial structure of ShadowDiagram*(C) *does not change.*

Proof. Two shadow diagrams have the same combinatorial structure if and only if, for each $p = (x_p, y_p) \in S$, the two versions of $Max(x_p)$ have the same edge sequence, and the feet of $\partial Shadow(p)$ lie on the same edges of $Max(x_p)$. Given two shadow diagrams SD and SD' with the same certificate set (X, A), we prove that they have the same combinatorial structures by induction, using a right-to-left sweep over the points of S.

First, because the x-order certificates are the same for SD and SD', the sweep encounters the points in the same order. For a given point $p \in S$, we assume inductively that $Max(x_p)$ is the same for SD and SD'. This is trivially true for the base case, in which p is the rightmost point of S. The curve $\partial Shadow(p)$ intersects $Max(x_p)$ in two points. The arc order certificates between neighbors in $Max(x_p)$ and between $\partial Shadow(p)$ and the edges of $Max(x_p)$ it contacts are the same in both SD and SD', and imply that the feet of $Shadow(p)$ are the same in the two shadow diagrams. Hence the sequence of edges of $Max(x_p)$ covered by $Shadow(p)$ is the same in both cases, implying the equivalence of the two instances of $Max(x_p - \varepsilon)$ for an infinitesimal ε, which establishes the inductive hypothesis for the next point of S to the left of p. □

5.3. *Maintaining the Shadow Diagram*

We now describe how to update the shadow diagram when a certificate fails. There are two parts to the update: modifying $Cover(p)$ and $Parents(p)$ for any affected points p, and updating the set of certificates. Each of these has to be done for x-order and for arc order certificates.

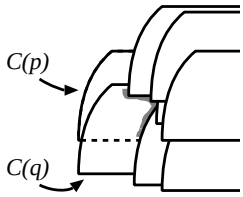

Fig. 12. When x-*order*(p, q) fails, the portion of *Cover*(q) that lies inside *Shadow*(p) is transferred to *Cover*(p).

5.3.1. *Failures of x-Order Certificates.* When an x-order certificate x-*order*(p, q) fails, it is easy to update the set of x-order certificates. Suppose that the other two x-order certificates involving p and q are x-*order*(a, p) and x-*order*(q, b). Then we remove all three certificates from the set of x-order certificates and add new certificates x-*order*(a, q), x-*order*(q, p), and x-*order*(p, b). The new certificates certify the new x-order of the points of S.

When x-*order*(p, q) fails, we may also have to update the cover lists and parent lists of some points in S, as well as some arc order certificates. If q does not appear at either end of *Cover*(p) (equivalently, $p \notin$ *Parents*(q)), then the shadow diagram does not change, and no lists or arc order certificates need to be updated.

If x-*order*(p, q) fails and q appears at one end of *Cover*(p), we must update lists and arc order certificates. (Note that q cannot appear at both ends of *Cover*(p) except in the degenerate case in which p and q have equal y-coordinates.) See Fig. 12. The only cover lists that change when x-*order*(p, q) fails are those of p and q. We search on *Cover*(q) to find the sublist L that lies inside *Shadow*(p). We remove q from one end of *Cover*(p) and replace it by L. Symmetrically, we remove L from *Cover*(q) and replace it by p. (Duplicate points may need to be added/removed at the ends of the transferred sublist L.) Four parent lists are affected by the x-swap of p and q. Point q stops being a parent of a point at one end of L, and becomes a parent of p. Symmetrically, p stops being a parent of q and becomes a parent of a point at the other end of L. Each added/removed parent causes the addition/removal of three arc order certificates.

5.3.2. *Failures of Arc Order Certificates.* When an arc order certificate *arc-order*(a, b, p) fails, we must update the shadow diagram in the vicinity of the triple intersection of ∂*Shadow*(a), ∂*Shadow*(b), and ∂*Shadow*(p). There are two kinds of arc order certificates, depending on whether a and b both belong to *Parents*(p) (Type 1), or whether one is a foot of the edge from ∂*Shadow*(p) (Type 2). See Fig. 11. Note that there are also certificates *arc-order*(a, b, p) in which both a and b are feet, but such certificates never fail, because ∂*Shadow*(p) always has a nontrivial edge in the shadow diagram.

When an arc order certificate of Type 1 fails, it is replaced by an arc order certificate of Type 2, and vice versa. For example, in Fig. 13, the Type 1 certificate *arc-order*(a, b, p) is replaced by the Type 2 certificate *arc-order*(b, p, a). To update the shadow diagram, we make a constant number of local modifications: *Parents*$(\cdot)$ and *Cover*$(\cdot)$ change for $O(1)$ points in the vicinity of the triple intersection, and arc order certificates for the neighbors of the triple intersection must be updated.

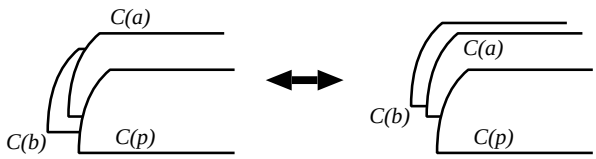

Fig. 13. When the Type 1 certificate *arc-order*(a, b, p) fails, it is replaced by the Type 2 certificate *arc-order*(b, p, a), and vice versa.

It is possible for multiple arc order certificate failures to happen simultaneously. In particular, there may be multiple arc order certificates associated with the parents of a horizontal shadow edge belonging to some ∂*Shadow*(p). See Fig. 14. If another horizontal shadow edge belonging to some ∂*Shadow*(q) passes through the first edge, all of the arc order certificates *arc-order*(a, b, p) associated with the horizontal edge must be replaced by the equivalent *arc-order*(a, b, q) certificates. Because p and q have different equations of motion, all these certificate updates are necessary, but they happen in the same instant of simulation time. (We can remove this degeneracy symbolically by giving each horizontal edge its own unique slope that is within some infinitesimal $\pm\varepsilon$ of zero—as a result, two horizontal edges will intersect in at most one point, and the arc order certificates fail in sequence, rather than simultaneously. However, we do not gain any computational advantage: although the certificates do not fail simultaneously, they fail within an infinitesimal time interval.)

5.4. *Maintaining* $\mathcal{G}$

We have seen that the two data structures *Cover*(p) and *Parents*(p) are sufficient to let us maintain the shadow diagram as the points of S move, when coupled with a linear number of KDS certificates. However, we also need to maintain the answers to a linear number of *shadow-shooting* queries of the form

> For a given point $p = (x_p, y_p) \in S$ and some constant c, what edge of *Max*$(x_p + c)$ is intersected by the horizontal line $y = y_p$?

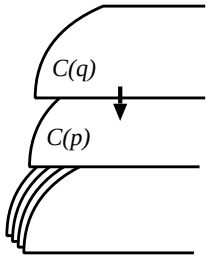

Fig. 14. When q moves below p, all of the shadow feet on ∂*Shadow*(p) transfer simultaneously to ∂*Shadow*(q).

The results of these queries, with $c \in \{0, 1\}$ and the implicit shape $C \in \{F_0, F_1, F_2, F_3\}$, determine the possible values of $\mathit{Target}(p, i, j)$ for all $p \in S$, $i \in \{0, 1\}$, and $j \in \{0, 1, 2, 3\}$. If q is the query result and $p \in D(q)$, i.e., the distance between p and q is less than two, then $q = \mathit{Target}(p, i, j)$ and there is an edge of $\mathcal{G}$ from p to q.

We can answer these queries initially by performing binary search on $\mathit{Max}(x_p + c)$ during the sweep line construction of the shadow diagram. One more distance comparison per query determines the edges of $\mathcal{G}$.

To certify the correctness of the results, we need four more certificates per query. We sort the query x-coordinates into the x-order of the points of S, and use x-order certificates to maintain the total x-sorted order of queries and points. Abusing notation slightly, we extend the $x\text{-}\mathit{order}(\cdot, \cdot)$ certificate so that an argument p refers to the x-position of a point p, and $p + c$ refers to the shifted x-position of a query point. Thus $x\text{-}\mathit{order}(a, p + c)$ certifies that $x_a < x_p + c$.

We use two more certificates, either arc order certificates or *y-order* certificates, to certify the edge of $\mathit{Max}(x_p + c)$ that is intersected by the horizontal line $y = y_p$. If $y = y_p$ intersects an edge that belongs to $\partial\mathit{Shadow}(q)$, let a and b be the points of $\mathit{Parents}(q)$ whose shadow edges bound the interval of $\partial\mathit{Shadow}(q)$ intersected by $y = y_p$. If multiple queries map to the same interval (a, b) of $\mathit{Parents}(q)$, then they are maintained in vertical order with y-order certificates: $y\text{-}\mathit{order}(p, p')$ holds if and only if $y_p < y_{p'}$. We also maintain the y-ordered list of queries for a given interval (a, b) in a sorted list. The lowest and highest queries in the list are compared against a and b using arc order certificates. By abuse of notation we refer to these certificates as $\mathit{arc\text{-}order}(a, y_p, q)$ and $\mathit{arc\text{-}order}(y_p, b, q)$.

Finally, we use a *distance* certificate $\mathit{near}(p, q)$ or $\mathit{not\text{-}near}(p, q)$ to certify whether the distance from p to q is less than or greater than two. (Here q is the answer to a shooting query $(x_p + c, y_p)$.) We put an edge in $\mathcal{G}$ if and only if the distance is less than two.

To summarize, the certificates needed to maintain the shadow diagram (Section 5.2) are

$x\text{-}\mathit{order}(p, q)$ Says that $x_p < x_q$.
$\mathit{arc\text{-}order}(a, b, p)$ Let α be the intersection of $\partial\mathit{Shadow}(a)$ with $\partial\mathit{Shadow}(p)$, and let β be the intersection of $\partial\mathit{Shadow}(b)$ with $\partial\mathit{Shadow}(p)$. The certificate says that α and β occur in counterclockwise order along $\partial\mathit{Shadow}(p)$.

These certificates are augmented to maintain query results by the following:

$x\text{-}\mathit{order}(p + c, q + c')$ Says that $x_p + c < x_q + c'$.
$\mathit{arc\text{-}order}(y_q, b, p)$
$\mathit{arc\text{-}order}(a, y_q, p)$ Replace the appropriate one of α or β in the description above by the intersection of $y = y_q$ with $\partial\mathit{Shadow}(p)$. Then the certificate says that α and β occur in counterclockwise order along $\partial\mathit{Shadow}(p)$.
$y\text{-}\mathit{order}(p, q)$ Says that $y_p < y_q$.
$\mathit{near}(p, q)$ Says that $p \in D(q)$.
$\mathit{not\text{-}near}(p, q)$ Says that $p \notin D(q)$.

Lemma 5.2. *Suppose that a shadow diagram, a set of shadow-shooting queries, and the graph $\mathcal{G}$ are correctly computed, and certified as described above. Then as long as*

no certificate fails, the shadow diagram, shadow-shooting query results, and graph $\mathcal{G}$ remain correct.

Proof. The shadow diagram remains correct, by Lemma 5.1. Hence we just need to prove that the query results and graph $\mathcal{G}$ are correct. Consider a query (x_p+c, y_p). Because the x-order certificates maintain the x-sorted order of queries and points, the sequence $Max(x_p + c)$ that is the target of the shooting query does not change. Suppose that the answer to the shooting query $(x_p + c, y_p)$ is initially an edge of $\partial Shadow(q)$, for some q. The line $y = y_p$ intersects a particular interval (a, b) of $Parents(q)$. All the shooting queries whose results lie in the same interval (a, b) of $Parents(q)$ are sorted vertically, and their vertical order is maintained by y-order certificates. The y-order of the lowest and highest queries relative to the intersections of $\partial Shadow(a)$ and $\partial Shadow(b)$ with $\partial Shadow(q)$ is maintained by arc order certificates, and hence as long as no certificates fail, the answer to the query (x_p+c, y_p) remains unchanged. Finally, for a query/answer pair p and q, the graph $\mathcal{G}$ contains an edge (p, q) if and only if $p \in D(q)$; this condition or its negation is certified by a distance certificate, so $\mathcal{G}$ does not change as long as no certificate fails. □

To maintain these queries as the points move, we must describe how to update the certificates and query results when a certificate fails. There are four kinds of certificates, and each has its own update strategy. Furthermore, these certificates interact with the certificates for the shadow diagram, and we need to update the query certificates when the shadow diagram certificates fail.

Distance certificate failures are the easiest to repair. We simply replace the certificate by its negation, and add/remove the corresponding edge of $\mathcal{G}$.

It is also easy to repair the failure of a certificate $y\text{-}order(p, p')$. Two queries change y-order, but both continue to project onto the same interval (a, b) of $Parents(q)$. We simply exchange p and p' in the y-sorted list of queries whose solution is (a, b). We delete the certificate $y\text{-}order(p, p')$ and create a new certificate $y\text{-}order(p', p)$. We delete the other two y-order or arc order certificates in which query points p and p' participate, and replace them by new certificates in which p is replaced by p', or vice versa.

When an x-order certificate involving a query fails, the answer to the query may change. However, if the certificate involves two queries, neither answer changes, and we simply update the x-order certificates to reflect the new order.

If $x\text{-}order(p + c, q)$ fails, we check whether q's edge is the answer to the $p + c$ query. If not, we simply update the x-order certificates. If yes, then the answer to the query changes. We use binary search on $Cover(q)$ to find the new answer to the query; suppose the old query answer was interval (a, b) in $Parents(q)$, and the new answer is interval (a', b') in $Parents(q')$. Then we remove the query from the y-ordered list for (a, b) and insert it into the list for (a', b'). We update the y-order certificates and the arc order certificates involving the query's y_p and its old and new neighbors. We also update the query's distance certificate and the graph $\mathcal{G}$.

If $x\text{-}order(q, p + c)$ fails, we reverse the update of the preceding paragraph. If the y-interval of $\partial Shadow(q)$ includes y_p, then q is the new answer to the query. We remove the query from its previous y-ordered list and insert it into the one for the appropriate interval of $Parents(q)$, updating y-order, arc order, and distance certificates as needed,

as well as updating $\mathcal{G}$. If the answer to the query does not change, then we simply update the x-order certificates.

When an arc order certificate involving a query fails, we remove the query from the y-ordered list of its current interval (a, b) of *Parents*(q), where q is the current answer to the query. The arc order certificate compares y_p with the position of a vertex of the shadow diagram. There are one or two edges of the shadow diagram on the opposite side of the vertex from the interval (a, b). One of them is the new answer to the query, and comparison of $x_p + c$ with the positions of the points of S that generate the edges tells which it is (call it q'; q' may be q). We insert the query into the y-ordered list for the appropriate interval of *Parents*(q') and update all the affected y-order, arc order, and distance certificates, as well as $\mathcal{G}$.

Lemma 5.3. *Suppose that a shadow diagram, a set of shadow-shooting queries, and the graph $\mathcal{G}$ are correctly computed, and certified as described above. Suppose that the points of S move continuously, and we respond to each certificate failure by updating the structures and certificates as described above. Then the shadow diagram, shadow-shooting queries, graph $\mathcal{G}$, and their certificates are always correct for the current configuration of S.*

5.5. *Efficiency*

In this section we argue that the total number of certificate failures is $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$. More concretely, we argue that the number of certificate failures is $O(n\lambda_s(n))$, where $\lambda_s(n)$ is the worst-case length of a Davenport–Schinzel sequence whose parameter s depends on the algebraic degree of the motion of points in S [11]. Since the total number of connectivity changes is $\Omega(n^2)$ in the worst case, our KDS is efficient.

We bound the number of certificate failures for the different kinds of certificates independently. The kinds of certificates we consider are x-order certificates, y-order certificates, arc order certificates involving only shadow edges, arc order certificates involving queries and shadow edges, and distance certificates.

First, we observe that the total number of x-order and y-order certificate failures is $O(n^2)$. Each such certificate is determined by the relative positions of two points of S, perhaps with an offset: $x_p + c < x_q$, for $c \in \{-1, 0, 1\}$. Any such predicate can change its truth value at most a constant number of times for points in algebraic motion.

Second, consider arc order certificates involving only shadow edges. Note that whenever an arc order certificate involving only shadow edges fails, one foot of some shadow changes. However, the total number of times that the feet of any one shadow can change is $O(\lambda_s(n))$. To see this, consider the edge of ∂*Shadow*(p) that extends upwards from the leftmost point of $C(p)$ and then off to $x = \infty$. Call this semi-infinite edge e. The foot of e (the upper foot of *Shadow*(p)) is given by the first intersection of e with a *candidate* shadow; the candidates are shadows whose reference points are right of p and whose y-intervals include the horizontal part of e. The total number of candidates is $O(n)$, and the total number of changes to the candidate set is also $O(n)$. This follows because each change to the candidate set is associated either with an x-order change of p and some q, or with a y-order change of $y_p + c$ and y_q for some q, where c is a fixed

constant determined by the vertical distance between y_p and the horizontal part of e. Because the points move algebraically, each point q can enter or leave the candidate set of p $O(1)$ times. Since the motion of the points of S is algebraic, any pair of candidates can alternate $O(1)$ times as the first shadow intersected by e. Hence the identity of the upper (or lower) foot forms a Davenport–Schinzel sequence over time, which implies that it changes $O(\lambda_s(n))$ times, for some constant s.

Third, consider arc order certificates involving queries and shadow edges. The result of a query at any instant is the intersection of a horizontal line with a particular edge of the shadow diagram. Define the *covering edge* of a query to be the edge of the shadow diagram that intersects the query line immediately to the left of the query result. Note that the number of times the result of a query (the identity of the intersected edge) can change is $O(\lambda_s(n))$, by an argument similar to the one bounding the number of foot changes. Similarly, the covering edge of a query can change $O(\lambda_s(n))$ times. (The covering edge of a query $(x_p + c, y_p)$ is simply given by the rightmost $q \in S$ such that $x_q < x_p + c$ and $Shadow(q)$ intersects $y = y_p$.) Now observe that every certificate failure involving a query line and a vertex of the shadow diagram corresponds either to a change in the query result or to a change in the covering edge of the query. Hence the total number of such certificate failures is $O(\lambda_s(n))$ per query.

Finally, the total number of distance certificate failures is $O(n^2)$, because the distance between any pair of points (p, q) can change from greater than two to less than two or vice versa only $O(1)$ times.

5.6. *Assembling the Pieces*

We now have all the pieces needed to put together a KDS to maintain the connectivity of unit disks moving in the plane.

For each of the four disk fragments F_j, for $j \in \{0, 1, 2, 3\}$, we construct the shadow diagram $ShadowDiagram(F_j)$ in $O(n \log n)$ time by a right-to-left sweep over the points of S. For each point $p \in S$, we perform shadow-shooting queries in $Max(F_j, x_p + i)$ for $i \in \{0, 1\}$. This gives two queries per point of S in each of the four different shadow diagrams. The results of these queries define the possible values of $Target(p, i, j)$ for each point $p \in S$, $i \in \{0, 1\}$, and $j \in \{0, 1, 2, 3\}$. If q is the query result and $p \in D(q)$, then $q = Target(p, i, j)$; if $p \notin D(q)$, then $Target(p, i, j)$ is null.

The graph $\mathcal{G}$ contains up to eight edges from each $p \in S$ to points to its right, specifically to each non-null $Target(p, i, j)$. By Theorem 3.3, $\mathcal{G}$ has the same connectivity as the disks. We use the structure of Holm et al. [6] to maintain the connectivity of $\mathcal{G}$ dynamically, as edges are added and removed. Connectivity queries on this structure take $O(\log n / \log \log n)$ time apiece.

To maintain $\mathcal{G}$ as the points of S move, we create the $O(n)$ certificates described in Sections 5.2 and 5.4, which certify the correctness of the shadow diagram, the shadow-shooting queries, and $\mathcal{G}$. Following the standard KDS framework, we put these certificates into a priority queue ordered by failure time, then process certificate failures as they occur.

Each certificate failure can be processed in $O(\log n)$ time, exclusive of the time needed to maintain the connectivity structure: we perform a constant number of logarithmic-time operations on $Cover(\cdot)$ and $Parents(\cdot)$ lists, then delete and create a constant number of

certificates, as described in Sections 5.3 and 5.4. Inserting/deleting these certificates from the priority queue takes $O(\log n)$ time apiece.

When a certificate fails, the results of $O(1)$ queries may change, which means that $O(1)$ edges of $\mathcal{G}$ change. Updating the connectivity structure takes $O(\log^2 n)$ amortized time per edge insertion or deletion [6].

Taking the event bounds of Section 5.5 into account, we have the following theorem.

Theorem 5.4. *Given a collection of n unit disks moving in the plane, we can maintain the connected components of the disks in a KDS that supports connectivity queries in* $O(\log n/\log\log n)$ *time apiece*; *each KDS certificate failure takes* $O(\log^2 n)$ *amortized time to process. The KDS is compact and efficient.*

Unfortunately, our disk connectivity KDS is neither local nor responsive. The KDS is not local because a single point of S may participate in $\Theta(n)$ certificates. It is responsive only in an amortized sense because the connectivity graph update bound is only amortized [6]; it is also true that a linear number of events may occur simultaneously, as noted in Section 5.3.2, but this is not technically a responsiveness problem, because each event (certificate failure) is processed quickly.

The problem of simultaneous events could probably be avoided by grouping together the parents whose shadow boundaries intersect a single horizontal shadow edge, then using a vertically ordered tournament to pick out the arc order certificate for the horizontal edge due to fail first. These arc order certificates would no longer refer directly to the horizontal edge, and so would not have to be replaced when one horizontal edge sweeps over another. The (nontrivial) details are left to the reader.

Note that although the KDS is efficient, it is still possible for it to undergo many events while the connectivity of the unit disks changes very little. For example, if n disjoint disks with centers on $x = 0$ move past n disjoint disks with centers on $x = 3$, our data structure undergoes $\Theta(n^2)$ events while the disk connectivity does not change at all. Similar problems affect most KDSs: because they maintain internal state that is not externally visible, they may process many more internal events than external ones, for some input data.

6. Conclusion

Although our result is stated for the connectivity of unit disks, it also applies to translated copies of any convex shape D that satisfies certain mild conditions. Let M be the Minkowski sum of D with its reflection through the origin: $M = D + D^{-1}$. Given two points p and q, $D(p) \cap D(q) \neq \emptyset$ if and only if $p \in M(q)$. Our result applies to D if we can compute intersections of translates of M quickly and can partition the right half of M into $O(1)$ slabs and rectangles such that the analogues of Lemmas 3.1 and 3.2 hold.

Note that in addition to answering connectivity queries, we can use our graph $\mathcal{G}$ to list the elements of the connected component containing a query disk in time proportional to the component's size.

Our result can be improved slightly in two ways. First, we note that we need only six queries (= graph edges) per point, not eight. We do not need to compute $Target(p, 1, 0)$

or *Target*$(p, 1, 3)$, because *Target*$(p, 0, 0)$ and *Target*$(p, 1, 0)$ are in the same connected component, as are *Target*$(p, 0, 3)$ and *Target*$(p, 1, 3)$. Second, we can reduce the number of x-order events, perhaps substantially, at the expense of a somewhat more complex certificate structure. We do not need to maintain the complete x-order of all points and queries. Instead, we need x-order certificates only for pairs of shadows such that one is a foot of the other, and for query/shadow pairs for which an x-order change will change the query result.

A challenging open problem is to extend our result to disks of different radii, or to unit balls in three dimensions. Both extensions seem to be quite difficult.

References

1. J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. *J. Algorithms*, 31(1):1–28, 1999.
2. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.
3. H. Edelsbrunner. Modeling with simplicial complexes: topology, geometry, and algorithms. In *Proceedings of the* 6*th Canadian Conference on Computational Geometry*, pages 36–44, 1994.
4. H. Edelsbrunner. The union of balls and its dual shape. *Discrete Comput. Geom*., 13:415–440, 1995.
5. J. Hershberger and S. Suri. Kinetic connectivity of rectangles. In *Proceedings of the* 15*th ACM Symposium on Computational Geometry*, pages 237–246, 1999.
6. J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. In *Proceedings of the* 30*th Annual ACM Symposium on Theory of Computing*, pages 79–89, 1998.
7. H. Imai and Ta. Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J. Algorithms*, 4:310–323, 1983.
8. D. B. Johnson. Routing in ad hoc networks of mobile hosts. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 158–163, 1994.
9. F. P. Preparata and M. I. Shamos. *Computational Geometry*: *An Introduction*. Springer-Verlag, New York, 1985.
10. W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.
11. M. Sharir and P. K. Agarwal. *Davenport–Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.
12. C.-K. Toh. A novel distributed routing protocol to support ad-hoc mobile computing. In *Proceedings of the* 15*th IEEE Annual International Phoenix Conference on Computers and Communication*, pages 480–486, 1996.

Received September 20, 2000, *and in revised form January* 19, 2001. *Online publication April* 6, 2001.