Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computational Geometry 31 (2005) 3–30
Computational Geometry
Theory and Applications
www.elsevier.com/locate/comgeo

Smooth kinetic maintenance of clusters

John Hershberger

Mentor Graphics Corp., 8005 SW Boeckman Road, Wilsonville, OR 97070, USA

Received 6 June 2003; received in revised form 13 February 2004; accepted 25 February 2004

Available online 26 November 2004

Communicated by D.M. Mount

Abstract

We propose a simple, deterministic kinetic data structure (KDS) for maintaining a covering of moving points by axis-aligned unit boxes in $\mathbb{R}^d$. The number of boxes is always within a factor of 3^d of the best possible static covering. In the plane, this approximation factor (9) compares favorably with the approximation factor (around one million) of the best previous algorithm. The new KDS is efficient, local, compact, and responsive.

Keywords: Kinetic data structure; Clustering

1. Introduction

Clustering of geometric data is a central theme in operations research, statistics, and computational geometry. Most nontrivial variants of the problem are NP-complete, although good approximation algorithms often exist [1–4]. If the coordinates of the data elements change continuously, the problem of kinetic cluster maintenance arises: as the data elements move, the goal is to maintain a set of clusters that is as close as possible to an optimal static clustering. See Fig. 1 for an example of moving points covered by unit squares in the plane.

The kinetic clustering problem has its own challenges, over and above those of the static problem. For example, if facilities are associated with the cluster centers, it may be a practical requirement that the maximum speed of the facilities is not significantly greater than the maximum speed of the data elements.

E-mail address: john_hershberger@mentor.com (J. Hershberger).

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2004.02.004

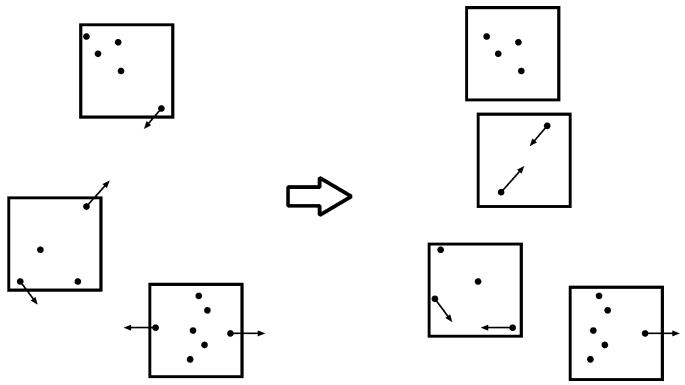


Fig. 1. Maintaining a unit-square covering of moving points.

It is not always possible to ensure this condition with optimum clusters, as Bespamyatnikh et al. have shown [5]. In other cases it is important that the set of active clusters change smoothly. For example, in a communication network, a large-scale reconfiguration of the clusters may make the network unusable for a period of time.

Fixed-diameter kinetic clusters are important for organizing collaborating mobile devices (sensor nets, wireless communicators, robots, e.g.) into *ad hoc* networks [6–8]. Clusters provide an organizational structure that helps optimize communication bandwidth, power usage, message routing [9], and network integrity [10]. Kinetic clusters also have applications in databases, where they can be used to support approximate or aggregate queries and filtering searches on time-varying data.

In 2001 Gao et al. [11] presented a randomized algorithm to maintain clusters for a set of points moving in the plane. The algorithm maintains a set of axis-aligned unit squares such that every moving point is contained in one of the squares. The number of squares is within a constant factor of the best possible, and the covering can be maintained by a kinetic data structure (KDS) that is efficient, local, compact, and responsive (see [12] for definitions).

This solution to the clustering problem, attractive as it is, has two significant flaws. The most important is the approximation quality. Although the number of covering squares is within a constant factor of the minimum possible, in the worst case the constant factor is on the order of 10^6. A second, lesser drawback is randomization. The efficiency, responsiveness, etc. of the KDS are not guaranteed, but hold only with high probability.

This paper proposes a simple, deterministic KDS that maintains a covering of moving points by unit boxes in $\mathbb{R}^d$. The worst-case approximation factor is 3^d. In the plane, this factor of $3^2 = 9$ compares very favorably with the previous algorithm's approximation factor of around one million. The KDS is efficient, local, compact, and responsive, with somewhat better bounds than the previous algorithm for all of these measures.

The new structure treats the d coordinates of the moving points separately. We maintain a one-dimensional clustering for the x_1-coordinates of the points. This partitions the points into slabs of at most unit width. Within each slab, we recursively partition the points according to coordinates $x_2, \ldots, x_d$. An important aspect of our solution is its *smoothness*: at each level of the hierarchy, each update moves $O(1)$ points between sets, and thus no expensive reconfigurations occur.

Because the new algorithm projects the points onto each dimension in turn, it is less suited for distributed implementation than the algorithm of Gao et al. [11]. Thus it may be more appropriate for use in databases and simulations than for use in distributed networks.

Below are several alternative approaches to the fixed-diameter kinetic clustering problem. We describe each scheme as applying to points in the plane, with the understanding that some of them can be extended to higher dimensions.

Gridding. Partition the plane into unit squares, and assign each point to a cluster corresponding to the square containing it. A drawback of this approach is that if the points move quickly, but in formation, they undergo unboundedly many cluster changes even though the actual groupings of the points do not change at all. The simplicity of this approach may make it useful when the points move slowly in a bounded domain.

Static clusters. Har-Peled [13] has shown how to partition n linearly moving points into k^2 *static* clusters so that at any time the diameter of each cluster is at most equal to the maximum cluster diameter in an optimal k-clustering for the current point positions. This is an intriguing solution, because the clusters require no maintenance as the points move. However, because the number of clusters is fixed, the maximum diameter is not. The number of clusters is also significantly more than the minimum possible. Furthermore, this solution requires that all the motions be known in advance.

R-trees. R-trees are frequently used in databases to maintain clusters [14–16]. However, the quality (cluster diameter) of an R-tree approximation degrades as the points move. Restoring the cluster diameter requires reclustering, which may be quite expensive. Techniques exist to maintain an R-tree using mostly local changes [17], but even so an R-tree has no guarantee of clustering optimality.

kd-trees. Agarwal, Gao and Guibas [18] introduced a kinetic version of a kd-tree that can be used to maintain clusters. This solution depends more strongly on predicting the future motion of the points (the flight plans) than does our approach.

Randomized clustering. This is the scheme mentioned above [11]. It forms clusters by $O(\log\log n)$ rounds of randomized *leader nomination*, resulting in an expected number of clusters that is within a (large) constant factor of the best static clustering at any time. It can be implemented in a distributed setting.

2. A simple 1-D algorithm

Our algorithm is motivated by the observation that it is easy to maintain a near-optimal unit-interval covering for points moving on the real line. The following rules (suggested by Li Zhang) maintain such a cover. The rules partition the points into sets such that the diameter of each set is at most 1, and the intervals spanned by distinct sets are disjoint.

(0) Initialize the cover greedily with a left-to-right sweep.
(1) Maintain the leftmost and rightmost points in each set, using a kinetic tournament. (Maintaining a total ordering would also work, but would be less efficient.)

(2) If two adjacent sets bump into each other, exchange their outermost points. That is, if L and R are adjacent sets, and the rightmost point of L passes the leftmost point of R, move each of these points to the other set.
(3) If a set becomes too wide (the separation between its leftmost and rightmost points becomes greater than 1), split off its rightmost point into a singleton set, and check whether rule (4) applies.
(4) If two adjacent sets get narrow and close enough together that both fit in a unit interval, merge the two sets into one.

This maintenance scheme is efficient: every change in the sets can be charged to a crossing of two points, or to a crossing of one point and a unit-shifted copy of another. The total number of such events is $O(n^2)$, and in the worst case any constant-factor approximate clustering must change $\Omega(n^2)$ times as the points move [11].

It is also easy to argue that the number of sets is within a constant factor of optimal: every two consecutive sets in the approximation cover an interval of width greater than 1, and so any single set in an optimal covering intersects at most three sets in the approximate covering.

The problem with this approach arises when we try to generalize it to higher dimensions. Given points moving in the plane, we would like to project the points onto the x-axis and maintain a partition into less-than-unit-width slabs, then partition the points in each of these slabs into sets based on their y-coordinates.

Consider what happens to an instance of y-interval maintenance when the x-sets change. Rules (2) and (3) cause single-point insertions and deletions in the y-sets. If a point is inserted/deleted in the middle of an existing set, we simply add it into/remove it from the min/max y-tournaments. If a point is inserted outside any set, create a singleton set and apply rule (4). If a point is deleted on the boundary of an existing set, re-evaluate the set's min/max point and check whether rule (4) applies. See Fig. 2, left side.

Unfortunately, rule (4) causes the scheme to break down, because it changes sets extensively. If two x-sets merge because of rule (4), then the y-intervals of the combined set have to be rebuilt completely—if we simply overlaid the y-intervals of the two x-sets, the invariants of the intervals would not hold at all. See Fig. 2, right side.

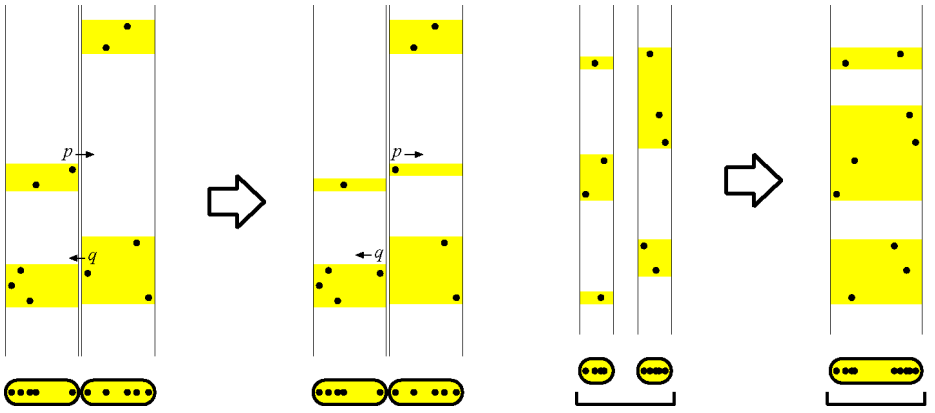


Fig. 2. The one-dimensional scheme can be applied hierarchically if O(1) points move between sets (left), but it breaks down when two x-sets merge (right).

3. Smooth maintenance

To avoid the expensive updates of the scheme outlined above, we develop a *smooth maintenance* scheme for one-dimensional covering. We replace rule (4) by more complex rules, such that whenever the x-covering has to be updated, O(1) points move between sets. This means that the covering algorithm of the previous section can be used for the y-intervals, since that structure can be maintained under single-point insertions/deletions. In fact, by extending the smooth maintenance scheme to handle insertions/deletions, as we do in Section 5, we can use it to maintain unit box coverings in any dimension.

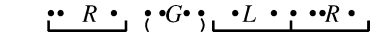
The smooth maintenance scheme uses three kinds of sets:

Right sets. A right set R has an associated unit interval that contains R and no other points. The left end of this interval coincides with the left endpoint of R (and hence the interval extends to the right).

Left sets. A left set L has an associated unit interval that contains L and no other points. The right end of this interval coincides with the *left* endpoint of a neighboring set of type *right* or *gap*. Thus the interval of a left set extends to the left of the set's points.

Gap sets. A gap set does not have a containing unit interval, generally because there is no room for one. Let A and B be the left and right neighbors of a gap set G, resp. A and B are left or right sets—we enforce the condition that no gap set has a gap set as a neighbor. The distance from the left end of B's interval to the left endpoint of G is less than 1; otherwise G would be a right set. In addition, if B is a right set, then the distance between the intervals of A and B is less than 1, or else G would be a left set.

Below is an example of these set types. In this figure, as in those in the remainder of the paper, we use the letters R, L and G to denote the types of sets. We use parentheses to denote the bounds of a gap set, and under-brackets to denote the unit interval bounds of left and right sets.



The initial configuration consists of an optimal number of right sets (only), created by a greedy left-to-right sweep.

To give precise invariants for these sets, we make several definitions that apply to nonempty sets of points on the real line. For a given set S, $lpt(S)$ and $rpt(S)$ are the leftmost and rightmost points of S. The x-coordinate of a point p is denoted by $x(p)$. Each set S has a type $type(S) \in \{\mathbf{R}, \mathbf{L}, \mathbf{G}\}$, and an interval $(\ell(S), r(S))$ based on the type and the points of S. We will keep the intervals of distinct sets interior disjoint, so we can define $\lambda(S)$ to be the set immediately to the left of S, and $\rho(S)$ to be the set immediately to the right of S. Here are the interval definitions:

$$\big(\ell(S), r(S)\big) = \begin{cases} (x(lpt(S)), x(lpt(S)) + 1) & \text{if } type(S) = \mathbf{R}, \\ (\ell(\rho(S)) - 1, \ell(\rho(S))) & \text{if } type(S) = \mathbf{L}, \\ (x(lpt(S)), x(rpt(S))) & \text{if } type(S) = \mathbf{G}. \end{cases}$$

We maintain the following five invariants:

(1) (Containment) If $p \in S$, then either $\ell(S) \leqslant x(p) < r(S)$ or $(x(p) = r(S)$ and $type(S) = \mathbf{G})$.

(2) (Disjointness) For all distinct S_1, S_2, either $r(S_1) \leqslant \ell(S_2)$ or $r(S_2) \leqslant \ell(S_1)$. This allows us to define $\lambda(S)$ and $\rho(S)$, and thus restate the invariant more simply as $r(S) \leqslant \ell(\rho(S))$ for all sets S.

(3) (Diversity) If $type(S) \in \{\mathbf{L}, \mathbf{G}\}$, then $type(S) \neq type(\rho(S))$.
(4) ($\mathbf{G} \not\to \mathbf{R}$) If $type(S) = \mathbf{G}$, then $\ell(\rho(S)) - \ell(S) < 1$.
(5) ($\mathbf{G} \not\to \mathbf{L}$) If $type(S) = \mathbf{G}$, $type(\lambda(S)) = \mathbf{R}$ and $type(\rho(S)) = \mathbf{R}$, then $\ell(\rho(S)) - r(\lambda(S)) < 1$.

Let us define the *diameter* of a set S to be $diam(S) = x(rpt(S)) - x(lpt(S))$. The diameter is no larger than the length of a set's interval, but may be smaller. Invariants 1, 2 and 4 imply that every set has diameter at most 1:

Lemma 1. *For each set S, $diam(S) \leqslant 1$.*

Furthermore, the invariants imply that this scheme has the same approximation factor as the scheme of Section 2.

Lemma 2. *The total number of sets in the smooth maintenance scheme is at most three times as large as the number of sets in an optimal covering by unit intervals.*

Proof. Invariant 3 means that the intervals of any two consecutive sets cover at least unit length. Therefore any unit interval in an optimal covering contains at most one smooth maintenance set in its interior. The left and right endpoints of the interval intersect at most two more sets. □

4. Maintaining the invariants under motion

In this section we give a detailed description of how to restore the invariants when point motions cause them to be violated. We first argue that if Invariant 4 or 5 is violated, but Invariants 1, 2 and 3 hold, and each set has diameter less than 1, then the violated invariant can be restored simply by changing the type of one set, without invalidating any other invariant. Once this is established, we can use this fact as a subroutine in restoring the first three invariants: if we can establish the first three invariants by some combination of point transfers and set type changes, and guarantee that the diameter of each set is less than 1, we can complete the invariant restoration by applying the results for Invariants 4 and 5.

Each of Invariants 1, 2 and 3 is described in its own subsection, and a case analysis breaks down the various circumstances in which the invariant can be violated. In each case, we describe how to restore the invariant and give a figure illustrating the case. We give a proof for each case that the invariants hold after the action is performed. Each proof exploits the unviolated invariants and the conditions of the case to show that each of the five invariants either continues to hold or is restored by the action.

The remainder of this section establishes the following theorem

Theorem 3. *Let $\mathcal{P}$ be a set of points moving continuously on the real line, partitioned into clusters that satisfy the invariants of Section* 3*. If point motion causes an invariant to fail, the invariants can be restored by* $O(1)$ *set type changes and point transfers between sets.*

We assume general position in this analysis. That is, at most one pair of points has separation 0, 1 or 2 at any instant. In Section 7 we discuss how to simulate general position in cases when it does not hold.

4.1. Restoring the "$\mathbf{G} \not\to \mathbf{R}$" and "$\mathbf{G} \not\to \mathbf{L}$" invariants

The invariants are

Invariant 4. *If type*$(S) = \mathbf{G}$, *then* $\ell(\rho(S)) - \ell(S) < 1$.

Invariant 5. *If type*$(S) = \mathbf{G}$, *type*$(\lambda(S)) = \mathbf{R}$ *and type*$(\rho(S)) = \mathbf{R}$, *then* $\ell(\rho(S)) - r(\lambda(S)) < 1$.

We show how to restore each invariant if it is violated:

Lemma 4. *If Invariants* 1, 2 *and* 3 *hold for a collection of sets, but Invariant* 4 *is violated for a set* S *for which* $\mathit{diam}(S) < 1$, *then changing type*(S) *to* **R** *re-establishes all invariants for* S, *without causing any other invariant violations.*

Proof. We argue that each invariant holds for S and its neighbors. (1) Containment holds for S because its interval expands, and for the neighbors because their intervals do not change. (2) Disjointness holds for $\lambda(S)$ because $\ell(S)$ does not change, for S because the invariant violation means there is no overlap between the intervals of S and $\rho(S)$, and for $\rho(S)$ because the intervals of $\rho(S)$ and $\rho(\rho(S))$ do not change. (3) Diversity holds by inspection for $\lambda(S)$ and S, and by continuity for $\rho(S)$. (4), (5) Neither invariant applies to any of the three sets, because by diversity (Invariant 3), neither $\lambda(S)$ nor $\rho(S)$ has type **G**. Changing *type*(S) does not affect invariants for any sets except S and its neighbors, so the lemma is established.

We prove a similar lemma for Invariant 5.

Lemma 5. *If Invariants* 1, 2, 3 *and* 4 *hold for a collection of sets, but Invariant* 5 *is violated for a set* S *for which* $\mathit{diam}(S) < 1$, *then changing type*(S) *to* **L** *re-establishes all invariants for* S, *without causing any other invariant violations.*

Proof. As in the proof of Lemma 4, we argue that all the invariants hold for S and its neighbors. This will establish the lemma, since invariants are not affected for any other set. (1) Containment holds as in the proof of Lemma 4. (2) Disjointness holds for $\lambda(S)$ because the violation of Invariant 5 ensures there is enough room for the expanded interval of S. By definition there is no overlap between the intervals of S and $\rho(S)$, which are a left set and its right neighbor. The intervals of $\rho(S)$ and $\rho(\rho(S))$ do not change, so they remain disjoint. (3) Diversity holds by construction for $\lambda(S)$ and S, and by continuity for $\rho(S)$. (4), (5) Neither invariant applies to any of the three sets, because none has type **G**. □

4.2. Restoring the "Containment" invariant

This invariant is

Invariant 1. *If* $p \in S$, *then either* $\ell(S) \leqslant x(p) < r(S)$ *or* $(x(p) = r(S)$ *and type*$(S) = \mathbf{G})$.

If $type(S) = \mathbf{G}$, then $\ell(S)$ and $r(S)$ are defined to be the extrema of S, and so this invariant cannot be violated for sets of type $\mathbf{G}$. Therefore, we consider only violations when $type(S) \in \{\mathbf{R}, \mathbf{L}\}$.

If $type(S) = \mathbf{L}$, the invariant can be violated in two possible ways: $rpt(S)$ can cross $r(S) = \ell(\rho(S))$, moving to the right into the interval of $\rho(S)$, or $lpt(S)$ can cross $\ell(S)$, moving to the left out of the interval of S.

In the cases for which $type(S) = \mathbf{R}$, for convenience we define $p = rpt(S)$. Thus $x(p) = r(S)$, and p is moving from inside the interval of S to outside (from left of $r(S)$ to right of it).

Fig. 3 presents the algorithm to restore the invariant in a pseudocode similar to C++, with each action illustrated in a small figure.

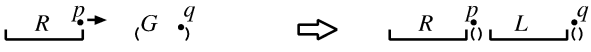
if $(type(S) = \mathbf{L})$ **{**
  if $(x(rpt(S)) = r(S))$
    Case 1A: Put $rpt(S)$ into $\rho(S)$ and put $lpt(\rho(S))$ into S.

  else
    Case 1B: Change $type(S)$ to $\mathbf{R}$.

}
else { // $type(S) = \mathbf{R}$.
  if $(\ell(\rho(S)) - r(S) > 1)$
    Case 1C: Convert p into a singleton set with $type(p) = \mathbf{R}$.

  else if $(type(\rho(S)) \neq \mathbf{G})$
    Case 1D: Convert p into a singleton set with $type(p) = \mathbf{G}$.

  else if $(r(\rho(S)) - r(S) > 1)$
    Case 1E: Let $q = rpt(\rho(S))$. Convert p into a singleton set with $type(p) = \mathbf{G}$; convert $\rho(S) \setminus q$ into a set S' with $type(S') = \mathbf{L}$; convert q into a singleton set with $type(q) = \mathbf{G}$.



  else
    Case 1F: Add p into the $\mathbf{G}$-set $\rho(S)$.

}

Fig. 3. Restoring the invariants after a containment violation.

Appendix A presents the proofs that these actions restore the violated invariant without breaking any others.

4.3. Restoring the "Disjointness" invariant

The invariant is

Invariant 2. *For all distinct S_1, S_2, either $r(S_1) \leqslant \ell(S_2)$ or $r(S_2) \leqslant \ell(S_1)$. This allows us to define $\lambda(S)$ and $\rho(S)$, and thus restate the invariant more simply as $r(S) \leqslant \ell(\rho(S))$ for all sets S.*

If $type(S) = \mathbf{L}$, then $r(S)$ is defined to be equal to $\ell(\rho(S))$, and so this invariant cannot be violated for sets of type **L**. Therefore, we consider only violations when $type(S) \in \{\mathbf{R}, \mathbf{G}\}$.

At the instant in which the invariant is about to be violated, $r(S)$ is equal to $\ell(\rho(S))$, and is about to become greater than it. We describe how to restore the invariant in several cases, depending on the types of S and its neighbors, and on the distances from S's interval to other sets' intervals. Fig. 4 gives an overview of the cases in which $type(S) = \mathbf{R}$, and Fig. 5 considers the cases in which $type(S) = \mathbf{G}$. The detailed correctness proofs for both figures appear in Appendix B.

4.4. Restoring the "Diversity" invariant

The invariant is

Invariant 3. *If $type(S) \in \{\mathbf{L}, \mathbf{G}\}$, then $type(S) \neq type(\rho(S))$.*

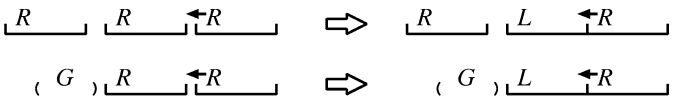
Because this is a combinatorial invariant, not a geometric invariant, point motions cannot make it fail. Only explicit set type changes or neighbor changes can invalidate the invariant. We ensure that the invariant is maintained during the updates needed to restore the other invariants.

5. Dynamic maintenance

Using the smooth maintenance scheme described above, in which the five invariants are maintained as the points move, we can maintain clusters in the plane. We use the smooth maintenance scheme to maintain clusters of at most unit width on the x-coordinates of the moving points, and then use the simple scheme of Section 2 to maintain clusters on the y-coordinates within each x-cluster. Because each change to the x-clusters transfers at most O(1) points between clusters, the simple 1-D scheme is sufficient to maintain the y-clusters, since it supports insertions and deletions.

In this section we show how to extend the smooth maintenance scheme to support insertions and deletions. This enables us to use smooth maintenance at all levels of the hierarchy in a d-dimensional covering of points by unit boxes. Each motion event at one level of the hierarchy creates O(1) dynamic changes to the clusterings at the next level of the hierarchy, and so on down to the leaves of the hierarchical data structure. Because each level of the clustering is within a factor of three of optimal, we can extend Lemma 2 to $\mathbb{R}^d$.

// Define $S' = \rho(S)$, $S'' = \rho(S')$, and $S''' = \rho(S'')$.
// Thus the sequence of four sets starting with S and
// extending to its right is S, S', S'', S'''.
if $(type(\rho(S)) = \mathbf{R})$ **{**
  if $(type(\lambda(S)) \neq \mathbf{L})$
    Case 2A: Change $type(S)$ to **L**.

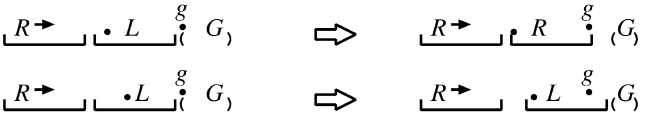


  else
    Case 2B: Change $type(S)$ to **G**.

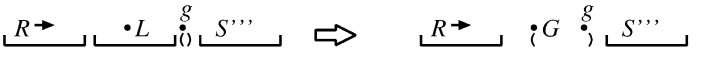
}
else if $(type(\rho(S)) = \mathbf{G})$
  Case 2C: Move $lpt(\rho(S))$ into S. If this makes $\rho(S)$ empty, delete $\rho(S)$.

else { // $type(\rho(S)) = \mathbf{L}$.
  if $(type(S'') = \mathbf{R})$
    Case 2D: Change $type(S')$ to **G**.

  else if $(|S''| > 1)$ // $type(S'') = \mathbf{G}$)
    Case 2E: Move $g = lpt(S'')$ from S'' into S'. If $\ell(S'' \setminus g) - \ell(S') > 1$, then set $type(S' \cup g)$ to **R**, else set it to **L**.



  else
    Case 2F: Combine $S' \cup S''$ into a single set of type **G**.



}

Fig. 4. Restoring the invariants after a disjointness violation, when $type(S) = \mathbf{R}$.

Lemma 6. *The smooth maintenance clustering covers a collection of points moving in $\mathbb{R}^d$ using at most 3^d times as many unit boxes as an optimal static covering.*

// Define $S' = \rho(S)$ and $S'' = \rho(S')$.
if $(type(S') = \mathbf{R})$
Case 2G: Let $p = rpt(S)$ and $q = lpt(S')$. Remove p from S and put it into S'; remove q from S' and put it into S.

else if $(|S| > 1)$ // $type(S') = \mathbf{L}$
Case 2H: Remove $rpt(S)$ from S and put it into S'.

else if $(type(\lambda(S)) = \mathbf{R})$ // $|S| = 1$
Case 2I: Combine $S \cup S'$ into a single set C with $type(C) = \mathbf{L}$.

else if $(type(S'') = \mathbf{R})$ // $type(\lambda(S)) = \mathbf{L}$
Case 2J: Combine $S \cup S'$ into a single set C with $type(C) = \mathbf{G}$.

else // $type(S'') = \mathbf{G}$
Case 2K: Let $p = lpt(S'')$. Remove p from S'', and delete S'' if this makes it empty. Combine $S \cup S' \cup p$ into a single set C with $type(C) = \mathbf{R}$.

Fig. 5. Restoring the invariants after a disjointness violation, when $type(S) = \mathbf{G}$.

Sections 5.1 and 5.2 cover insertions and deletions separately, thereby establishing the following theorem.

Theorem 7. *Let $\mathcal{P}$ be a set of points on the real line, partitioned into clusters that satisfy the invariants of Section* 3. *If a point is inserted into or deleted from $\mathcal{P}$, the invariants can be restored by* O(1) *set type changes and point transfers between sets.*

5.1. Insertions

We first consider how to reestablish the invariants when a point p is inserted into an existing collection of smooth maintenance sets. As in Section 4, we will assume general position: the inserted point does not lie at distance 0, 1 or 2 from any existing point, and no invariant is at the point of failing when p is inserted. Fig. 6 analyzes the insertion into cases, and the detailed proofs of correctness appear in Appendix C.

if ($\exists S$ such that $\ell(S) < x(p) < r(S)$)
Case Insert-A: Add p to S.

// We denote the sets left and right of p by $\lambda(p)$ and $\rho(p)$.
else if ($\ell(\rho(p)) - x(p) > 1$)
Case Insert-B: Make p into a singleton set S with $type(S) = \mathbf{R}$.

else if ($type(\lambda(p)) = \mathbf{G}$)
Case Insert-C: Add p into $\lambda(p)$.

else if ($type(\rho(p)) \neq \mathbf{G}$)
Case Insert-D: Make p into a singleton set S with $type(S) = \mathbf{G}$.

else if ($r(\rho(p)) - x(p) < 1$)
Case Insert-E: Add p into $\rho(p)$.

else // $r(\rho(p)) - x(p) > 1$
Case Insert-F: Let $q = rpt(\rho(p))$. Remove q from $\rho(p)$ and make it into a singleton **G**-set. Change $type(\rho(p) \setminus q)$ to **L**. Make p into a singleton **G**-set.

Fig. 6. Restoring the invariants after a point p is inserted.

5.2. Deletions

Deletions are harder to handle than insertions. We describe how to cope with deletions using a combination of lemmas (used as subroutines) and the sort of case analysis used in the previous sections.

When a point is deleted, if it is the left point of its containing set, then its set's interval may slide to the right, and the type of the set may change. This may affect the invariants for sets to the left and right of the containing set. We first argue that the invariants for sets to the left can be restored.

Lemma 8. *Given a set S, if $type(\rho(S))$ changes consistent with Invariant 3 and $\ell(\rho(S))$ increases discontinuously, then all invariants can be restored for S and all sets left of S with at most two set type changes.*

Proof. Suppose that $type(S) = \mathbf{R}$. The change to $\ell(\rho(S))$ does not violate disjointness (Invariant 2), and none of the other invariants is violated for S, either. Set S does not need to change, and therefore no sets to its left need to change.

Suppose that $type(S) = \mathbf{G}$. By assumption $type(\rho(S)) \in \{\mathbf{R}, \mathbf{L}\}$. Containment, disjointness, and diversity are not violated for S, but Invariant 4 or 5 might be. If either is violated, we apply Lemma 4 or 5 to correct it. No change to $\lambda(S)$ is needed, and hence the invariants hold for all sets to its left.

Suppose that $type(S) = \mathbf{L}$. By assumption $type(\rho(S))$ and $type(\lambda(S))$ belong to $\{\mathbf{R}, \mathbf{G}\}$. Disjointness and diversity are not violated for S, but containment might be if the interval of S slides too far to the right. Thus if $\ell(\rho(S)) - x(lpt(S))$ increases to > 1, then we change $type(S)$ to $\mathbf{R}$. The left endpoint $\ell(S)$ increases discontinuously to either $\ell(\rho(S)) - 1$ or to $x(lpt(S))$, and we apply one of the first two cases to $\lambda(S)$, since $type(\lambda(S)) \in \{\mathbf{R}, \mathbf{G}\}$. □

In several cases of point deletion, we will encounter a situation in which the right neighbor of a **G**-set S has been modified—either a point has been deleted, or the set's type no longer satisfies the invariants. The key property is that there is a unit interval to the right of $r(S)$ that contains all the points of $\rho(S)$ and no other points. The points of $\rho(S)$ belonged to a single set before the delete operation. We will call a set of points surrounded by an otherwise empty unit interval a *protected set*, because the unit interval "protects" it from other sets. The type of a protected set is not determined, and may be set/modified in the course of restoring invariants.

Lemma 9. *Let S be a set with $type(S) = \mathbf{G}$ and diameter less than 1, let $\rho(S)$ be a protected set, and suppose $type(\rho(\rho(S))) \neq \mathbf{G}$. Suppose that all sets to the left of S satisfy the invariants, as do all sets to the right of $\rho(S)$. Then S and $\rho(S)$ can be transformed with $O(1)$ point transfers and set type changes into at most three sets so that all sets satisfy the invariants.*

Proof. Note that because all invariants hold for sets to the left of S, $type(\lambda(S)) \neq \mathbf{G}$, by the diversity invariant.

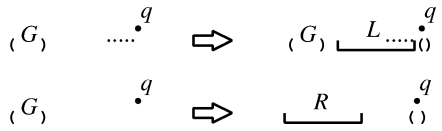
Let $p = rpt(S)$, and let $q = rpt(\rho(S))$. If $x(q) - x(p) < 1$, then we split off p from S to make $S'' = \{p \cup \rho(S)\}$ into a new set with $type(S'') = \mathbf{R}$.

Denote the reduced set $S \setminus p$ by S'. If $S = \{p\}$, then $S' = \emptyset$ is deleted.

We now argue that the invariants hold for this case. (1) Containment holds for S'' because $x(q) - x(p) < 1$; it holds for S' because $type(S') = \mathbf{G}$. (2) Disjointness holds by construction for S', and for S'' because $\rho(S)$ is protected: $\ell(\rho(\rho(S))) - x(p) > 1$. (3) Diversity holds for $\lambda(S')$, S' and S'' by construction. (4), (5) Because $S' \subset S$, Lemma 1 implies that $diam(S') < 1$. We apply Lemmas 4 and 5 to S' to restore Invariants 4 and 5.

Now suppose that $x(q) - x(p) > 1$. We split off q from $\rho(S)$ to form a singleton set $S'' = \{q\}$ and a reduced set $S' = \{\rho(S) \setminus q\}$. If $\rho(S) = \{q\}$, then $S' = \emptyset$ is deleted.

We set $type(S'') = \mathbf{G}$. If $S' \neq \emptyset$, we set $type(S') = \mathbf{L}$. If $S' = \emptyset$, we set $type(S) = \mathbf{R}$; otherwise $type(S)$ remains **G**.



We now establish the invariants for this case. (1), (2) Containment and disjointness hold by construction. (3) Diversity also holds by construction, since $type(\lambda(S)) \neq \mathbf{G}$ and $type(\rho(\rho(S))) \neq \mathbf{G}$. (4), (5) $diam(S) < 1$ by assumption, and $diam(S'') = 0$. Therefore we can apply Lemmas 4 and 5 to restore Invariants 4 and 5. □

Armed with these lemmas, we tackle the problem of invariant restoration after a deletion. Let p be the deleted point, with $p \in S$. We deal first with an easy case: $|S| > 1$, and either $type(S) \neq \mathbf{R}$ or $p \neq lpt(S)$. See Fig. 7. The proofs for this case appear in Appendix D.

We now consider the remaining cases in which $|S| > 1$, namely $type(S) = \mathbf{R}$ and $p = lpt(S)$. We restore the invariants in two steps. We first restore the invariants for S, $\rho(S)$, and all sets to the right. This may cause invariant violations for sets to the left of S, so we then restore those invariants.

Fig. 8 shows how to restore the invariants to the right of S; the correctness proofs appear in Appendix D.

There may be diversity violations or spacing violations with the sets to the left of S. Some of these can be restored using Lemma 8, and some require more significant action. Fig. 9 presents the code for these cases. The corresponding proofs appear in Appendix D.

Now we consider the cases in which $|S| = 1$, and so deleting p causes S to cease to exist. If $type(S) \in \{\mathbf{L}, \mathbf{R}\}$, we apply Lemma 8 to $\lambda(S)$. The type of $\lambda(S)$ will become $\mathbf{R}$, if it was not already, because the empty unit interval formerly occupied by S provides room. Thus $\lambda(S)$ satisfies all the invariants. The only invariant that might be violated for $\rho(S)$ is Invariant 5, and it can be restored using Lemma 5.

The pseudocode in Fig. 10 handles the cases in which $|S| = 1$ and $type(S) = \mathbf{G}$. The corresponding proofs appear in Appendix D.

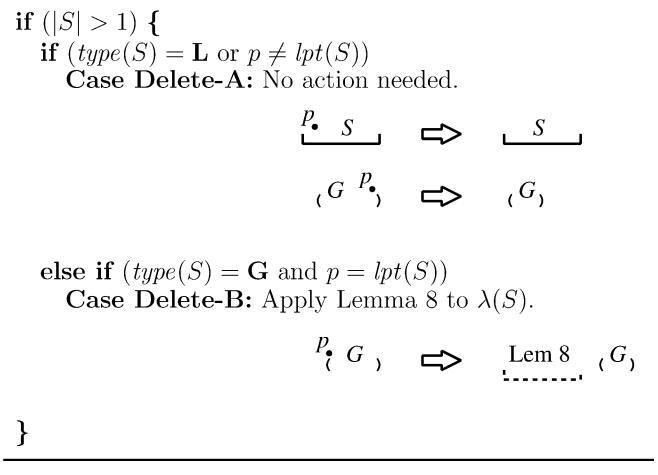
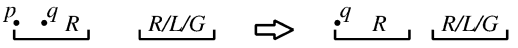


Fig. 7. Easy cases of point deletion. Point $p \in S$ is the deleted point.

// Assume $p = lpt(S)$, $|S| > 1$, and $type(S) = \mathbf{R}$.
// Define $S' = S \setminus p$ and $q = lpt(S')$.
if $(\ell(\rho(S)) - x(q) > 1)$
Case Delete-C: Set $type(S')$ to $\mathbf{R}$.



else
Case Delete-D: If $type(\rho(S)) \neq \mathbf{L}$, set $type(S')$ to $\mathbf{L}$; otherwise, set $type(S')$ to $\mathbf{G}$.

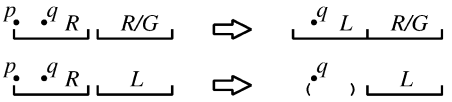
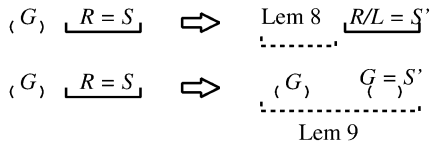


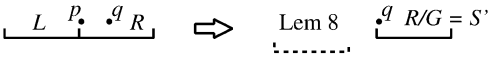
Fig. 8. Restoring the invariants for sets to the right after point p is deleted from a non-singleton set S.

if $(type(\lambda(S)) = \mathbf{R})$
Case Delete-E: No action needed.

else if $(type(\lambda(S)) = \mathbf{G})$
Case Delete-F: If $type(S') \neq \mathbf{G}$ then apply Lemma 8 to $\lambda(S)$. Otherwise ($type(S') = \mathbf{G}$), apply Lemma 9 to $\lambda(S)$ and S'.



else if $(type(S') \neq \mathbf{L})$ // $type(\lambda(S)) = \mathbf{L}$
Case Delete-G: Apply Lemma 8.



else if $(type(\lambda(\lambda(S))) = \mathbf{R})$
Case Delete-H: Change $type(\lambda(S))$ to $\mathbf{G}$.

else // $type(\lambda(\lambda(S))) = \mathbf{G}$
Case Delete-I: Apply Lemma 9 to $\lambda(\lambda(S))$ and $\lambda(S)$.

Fig. 9. Restoring the invariants for sets to the left of S after point p is deleted from a non-singleton set S, leaving $S' = S \setminus p$ and $q = lpt(S')$.

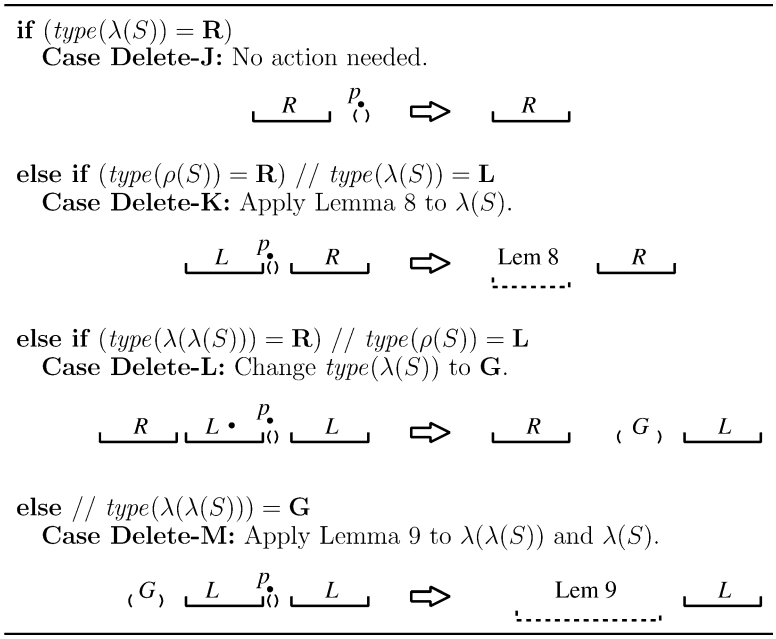


Fig. 10. Restoring the invariants after a singleton **G**-set is deleted.

6. Kinetic data structures

We use kinetic tournaments [12] to maintain $lpt(S)$ and $rpt(S)$. Each of the invariants can be converted into a KDS certificate that fails when the separation between two points is 0, 1 or 2. Thus the KDS is efficient, because there are $O(n^2)$ such separation events for points in pseudo-algebraic motion, and the worst-case number of changes to a kinetic covering with at most a constant factor more sets than the optimum is $\Omega(n^2)$ [11]. The KDS is compact, because it uses only $O(n)$ space. It is local—each point participates in $O(1)$ invariant certificates and $O(\log n)$ kinetic tournament certificates. Section 4 shows that each invariant certificate failure can be corrected by $O(1)$ set type changes and transfers of points between sets (leading to $O(\log n)$ certificate changes and $O(\log^2 n)$ total time per failure), and hence the structure is responsive.

Section 5 shows that the KDS can be made dynamic, at an update cost of $O(1)$ set type changes and point transfers per insertion or deletion. Therefore it can be applied recursively to maintain a unit box covering for points in any dimension $\mathbb{R}^d$, with a worst-case approximation factor of 3^d.

In practice this structure may perform better than the $O(n^2)$ worst-case bound. If there are many points, then we are not maintaining their total order in x and y. Instead, we are maintaining extrema of sets, which means that we don't have to track every exchange internal to the sets. That advantage applies most strongly to the top level of the data structure, but at the lower level, we may have fewer events because we are working with smaller subsets of the points.

7. General position and simulation of simplicity

Our algorithms for invariant restoration assume that at most one invariant fails at a time. When this assumption does not hold, we can use *Simulation of Simplicity* (SOS) [19] to process failures in a consistent order.

If the position of point i is $x_i(t)$, SOS treats it as though the position were $x_i(t) + \varepsilon^i$ for some infinitesimal $\varepsilon > 0$. SOS gives a consistent failure order to simultaneously-failing KDS certificates.

SOS is equivalent to lexicographic tie-breaking. A certificate involving indices i and j, with $i < j$, takes one of two forms:

$$x_i(t) - x_j(t) + m < 0,$$
$$-x_i(t) + x_j(t) + m < 0$$

for $m \in \{0, \pm 1, \pm 2\}$. Each certificate has a *key* associated with it. In the first case the key is $(n-i,\ j-n)$, and in the second case it is $(i-n, n-j)$, where n is the total number of points and the points are numbered from 0 to $n-1$. We order the certificates that fail at time t by largest-to-smallest lexicographic order on their keys. The following lemma shows that this lexicographic order gives a consistent, nondegenerate failure order.

Lemma 10. *The order of certificate failures given by lexicographic order is the same as the order given by Simulation of Simplicity.*

Proof. Consider two certificates, with signs represented by subscripted "s"s.

$$s_i x_i(t) + s_j x_j(t) + m < 0, \tag{1}$$
$$s_k x_k(t) + s_l x_l(t) + m' < 0. \tag{2}$$

Here $i < j$, $k < l$, $s_i = -s_j$, $s_k = -s_l$, and $s_i, s_k \in \{-1, 1\}$. Suppose both certificates fail at time t. Assume that $i \neq k$ or $j \neq l$, since otherwise the two certificates are the same. SOS orders the certificates' ε-extended expressions from larger to smaller. The one closer to zero—closer to failure—is performed first. Certificate (1) is performed before (2) if and only if

$$s_i \varepsilon^i + s_j \varepsilon^j > s_k \varepsilon^k + s_l \varepsilon^l.$$

(We have cancelled out the terms that are equal to zero at time t.) We can rewrite this condition programmatically as a conditional expression.

if $(i < k)$
  return $(s_i > 0)$
else if $(k < i)$
  return $(s_k < 0)$
else if $(s_i \neq s_k)$
  return $(s_i > s_k)$
else if $(j < l)$
  return $(s_j > 0)$ // $\equiv s_i < 0$
else // $(l < j)$
  return $(s_l < 0)$ // $\equiv s_k > 0$

The lexicographic comparison takes four different forms, depending on the signs s_i and s_k.

	Comparison	Equivalent
$s_i > 0, s_k > 0$:	$(n-i, j-n) > (n-k, l-n)$	$(k-i, j-l) > (0,0)$
$s_i > 0, s_k < 0$:	$(n-i, j-n) > (k-n, n-l)$	**true**
$s_i < 0, s_k > 0$:	$(i-n, n-j) > (n-k, l-n)$	**false**
$s_i < 0, s_k < 0$:	$(i-n, n-j) > (k-n, n-l)$	$(i-k, l-j) > (0,0)$

It is not difficult to verify that in each of these four cases, the result of the lexicographic comparison matches the result from the SOS conditional expression. □

8. Extensions

One possible enhancement of our approach is to combine smooth maintenance with approximate extents. Note that a covering approximates a set of points, but smooth maintenance *exactly* maintains an approximate covering. If we allow the extrema of each set to be inaccurate by some small amount, it is possible that we could maintain a covering that would be nearly as good, but with fewer KDS events. Agarwal and Har-Peled [20] showed that one-dimensional approximate extrema can be maintained more cheaply than exact extrema. It may be possible to use approximate extrema to improve the performance of the smooth maintenance scheme, at least in practice. (The smooth maintenance scheme is optimal in the worst case, because n points in linear motion can force $\Theta(n^2)$ cluster changes, even if approximate extrema are used; nevertheless, approximate extrema may still be of practical use.)

An intriguing direction for future work is smooth maintenance without dimensional reduction. For example, we would like to have a smooth maintenance scheme for two-dimensional clusters that does not project the points to the x and y axes separately. Given the complexity of our one-dimensional solution, a two-dimensional scheme would require a significant simplifying breakthrough.

Acknowledgements

This research began as an offshoot from the work I did with Jie Gao, Leo Guibas, Li Zhang and An Zhu on randomized maintenance of clusters for moving points [11]. Thanks to all of them for the discussions that inspired this work.

Appendix A. Proofs for "Containment" restoration

Here are the proofs that the algorithm of Section 4.2 restores the violated containment invariant without breaking any others.

Each proof covers one of the program's cases in a stylized format. The first line gives the condition to which the case applies. Next comes the action required to correct the invariant violation, along with a figure depicting the setup and the action. The final block gives a terse proof that each of Invariants 1, 2 and 3 is restored or continues to hold after the action, and that all affected **G**-sets have diameter

less than 1. (General position and Lemma 1 imply that any pre-existing **G**-set S has $diam(S) < 1$.) Any violations of Invariants 4 and 5 that remain after these changes can be corrected by set type changes using Lemmas 4 and 5.

Case 1A:
Condition: $type(S) = \mathbf{L}$, $rpt(S)$ crosses $r(S) = \ell(\rho(S))$.
Action: Put $rpt(S)$ into $\rho(S)$ and put $lpt(\rho(S))$ into S.

Proof. (1) Restored for S and $\rho(S)$. (2), (3), (4) and (5) continue to hold, because no intervals or set types change.

Case 1B:
Condition: $type(S) = \mathbf{L}$, $lpt(S)$ crosses $\ell(S) = \ell(\rho(S)) - 1$.
Action: Change $type(S)$ to **R**.

Proof. (1), (2) hold because no intervals change. (3) Ceases to matter for S; holds *a fortiori* for $\lambda(S)$.

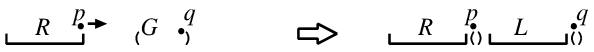
Case 1C:
Condition: $type(S) = \mathbf{R}$, $\ell(\rho(S)) - r(S) > 1$.
Action: Convert p into a singleton set with $type(p) = \mathbf{R}$.

Proof. (1) Restored for S, trivial for p. (2) Follows for (S, p) from $x(p) = r(S)$. Follows for $(p, \rho(S))$ from the condition. (3) Does not apply.

Case 1D:
Condition: $type(S) = \mathbf{R}$, $\ell(\rho(S)) - r(S) < 1$ (by general position), $type(\rho(S)) \in \{\mathbf{R}, \mathbf{L}\}$.
Action: Convert p into a singleton set with $type(p) = \mathbf{G}$.

Proof. (1) Restored for S, trivial for p. (2) Because $r(S) \leqslant \ell(\rho(S))$ originally, after the action we have $r(S) \leqslant \ell(p) = r(p) \leqslant \ell(\rho(S))$. (3) Follows from the condition.

Case 1E:
Condition: $type(S) = \mathbf{R}$, $\ell(\rho(S)) - r(S) < 1$ (by general position), $type(\rho(S)) = \mathbf{G}$, $r(\rho(S)) - r(S) > 1$.
Action: Let $q = rpt(\rho(S))$. Convert p into a singleton set with $type(p) = \mathbf{G}$; convert $\rho(S) \setminus q$ into a set S' with $type(S') = \mathbf{L}$; convert q into a singleton set with $type(q) = \mathbf{G}$.



Proof. (1) Holds trivially for the singleton **G**-sets p and q. Holds for S' because it previously held for $\rho(S)$, and because of Lemma 1. (2) Holds for S' and q by construction; holds for the singleton set p by the condition: $x(q) - x(p) = r(\rho(S)) - r(S) > 1$. (3) Holds for p and S' by construction; holds for q because it previously held for $\rho(S)$.

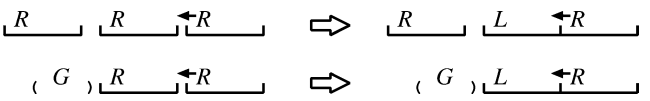
Case 1F:
Condition: $type(S) = \mathbf{R}$, $type(\rho(S)) = \mathbf{G}$, $r(\rho(S)) - r(S) < 1$ (by general position).
Action: Add p into the **G**-set $\rho(S)$.

Proof. (1) Restored for S, trivial for $p \cup \rho(S)$. (2) Easy; the intervals of S and $p \cup \rho(S)$ overlap only at $x(p)$. (3) Continues to hold unchanged.

Appendix B. Proofs for "Disjointness" restoration

Here are the proofs of correctness for the algorithm of Fig. 4.

Case 2A:
Condition: $type(S) = \mathbf{R}$, $type(\rho(S)) = \mathbf{R}$, $type(\lambda(S)) \in \{\mathbf{R}, \mathbf{G}\}$.
Action: Change $type(S)$ to **L**.



Proof. (1) Preserved because $\ell(S)$ moves infinitesimally to the left of $lpt(S)$, and $r(S) > x(rpt(S))$. (2) Restored. (3) Holds for S and $\lambda(S)$ by the condition.

Case 2B:
Condition: $type(S) = \mathbf{R}$, $type(\rho(S)) = \mathbf{R}$, $type(\lambda(S)) = \mathbf{L}$.
Action: Change $type(S)$ to **G**.

Proof. (1) Always true for **G**-sets. (2) Restored. (3) Holds for S and $\lambda(S)$ by the condition. (4), (5) $diam(S) < 1$ by general position ($x(lpt(S)) + 1 \neq x(rpt(S))$), so Lemmas 4 and 5 apply.

Case 2C:
Condition: $type(S) = \mathbf{R}$, $type(\rho(S)) = \mathbf{G}$.
Action: Move $lpt(\rho(S))$ into S. If this makes $\rho(S)$ empty, delete $\rho(S)$.

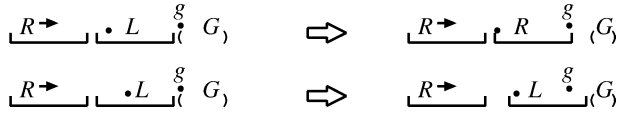
Proof. (1) Continues to hold for S and $\rho(S)$. (2) Restored. (3) Either unchanged, or vacuous if $\rho(S)$ is deleted.

In the following three cases we need to refer to sets that are distance two and three to the right of S. To abbreviate and simplify the notation, we define $S' = \rho(S)$, $S'' = \rho(S')$, and $S''' = \rho(S'')$. Thus the sequence of four sets starting with S and extending to its right is S, S', S'', S'''.

Case 2D:
Condition: $type(S) = \mathbf{R}$, $type(S') = \mathbf{L}$, $type(S'') = \mathbf{R}$.
Action: Change $type(S')$ to **G**.

Proof. (1) Always true for **G**-sets. (2) Restored. (3) By observation. (4), (5) $diam(S') < 1$ by general position ($x(lpt(S')) + 1 \neq x(rpt(S'))$), so Lemmas 4 and 5 apply.

Case 2E:
Condition: $type(S) = \mathbf{R}$, $type(S') = \mathbf{L}$, $type(S'') = \mathbf{G}$, $|S''| > 1$.
Action: Move $g = lpt(S'')$ from S'' into S'. If $\ell(S'' \setminus g) - \ell(S') > 1$, then set $type(S' \cup g)$ to **R**, else set it to **L**.



Proof. (1) The unit interval of S' slides to the right, but continues to contain all the points of $S' \cup g$. (Previously g was on the boundary of the interval; after the change it is inside.) (2) The invariant is restored for S. $type(S' \cup g)$ is chosen to preserve the invariant for $S' \cup g$. (3) By construction.

Case 2F:
Condition: $type(S) = \mathbf{R}$, $type(S') = \mathbf{L}$, $type(S'') = \mathbf{G}$, $|S''| = 1$.
Action: Combine $S' \cup S''$ into a single set of type **G**.

Proof. Define $C = S' \cup S''$. (1) C is contained in a unit interval, and therefore the interval of C contains its points. (2) Restored, since the interval of C lies to the right of the original interval of S', and $r(C) = r(S'')$. (3) By construction. (4), (5) $diam(C) = diam(S') < 1$ by general position ($x(lpt(S')) + 1 \neq x(rpt(S'))$), so Lemmas 4 and 5 apply.

We now consider the cases in which $type(S) = \mathbf{G}$. In some of the cases we need to refer to sets that are distance one and two to the right of S; as above, we define $S' = \rho(S)$, $S'' = \rho(S')$. Fig. 5 gives an overview of these cases.

Case 2G:
Condition: $type(S) = \mathbf{G}$, $type(S') = \mathbf{R}$.
Action: Let $p = rpt(S)$ and $q = lpt(S')$. Remove p from S and put it into S'; remove q from S' and put it into S.

Proof. (1) Always holds for **G**-sets such as S; holds *a fortiori* for S', since its interval moves to the right, without dropping any points on the left side. (2) Restored. (3) Continues to hold.

Case 2H:
Condition: $type(S) = \mathbf{G}$, $type(S') = \mathbf{L}$, $|S| > 1$.
Action: Remove $rpt(S)$ from S and put it into S'.

Proof. (1) The transferred point is inside the interval of S', so the invariant holds for S'; it always holds for the **G**-set S. (2) Restored. (3) No change.

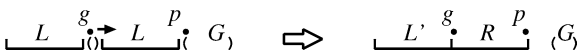
Case 2I:
Condition: $type(S) = \mathbf{G}$, $type(S') = \mathbf{L}$, $|S| = 1$, $type(\lambda(S)) = \mathbf{R}$.
Action: Combine $S \cup S'$ into a single set C with $type(C) = \mathbf{L}$.

Proof. (1) Holds because the interval of C is identical to that of S', and contains S' plus the single point of S. (2) The intervals stay the same, except that the one for S disappears. (3) By construction.

Case 2J:
Condition: $type(S) = \mathbf{G}$, $type(S') = \mathbf{L}$, $|S| = 1$, $type(\lambda(S)) = \mathbf{L}$, $type(S'') = \mathbf{R}$.
Action: Combine $S \cup S'$ into a single set C with $type(C) = \mathbf{G}$.

Proof. (1) The only changed interval is that of C, which is a **G**-set. The invariant always holds for **G**-sets. (2) No interval changes, except that of C, which is smaller than that of S'. (3) By construction. (4), (5) Because $\ell(S'') - \ell(S)$ is changing from > 1 to < 1, $diam(C) < 1$ after the change.

Case 2K:
Condition: $type(S) = \mathbf{G}$, $type(S') = \mathbf{L}$, $|S| = 1$, $type(\lambda(S)) = \mathbf{L}$, $type(S'') = \mathbf{G}$.
Action: Let $p = lpt(S'')$. Remove p from S'', and delete S'' if this makes it empty. Combine $S \cup S' \cup p$ into a single set C with $type(C) = \mathbf{R}$.



Proof. (1) Holds for C because at the instant that Invariant 2 is violated, the combined set C has width 1 and is shrinking. Continues to hold for S'', because it is a **G**-set. (2) The interval of $S'' \setminus p$ is right of the interval of C, which does not extend left of the original position of S. (3) By construction.

Appendix C. Proofs of invariant restoration after an insertion

Here are the proofs that the actions described in Fig. 6 restore the invariants after a point p is inserted. We show that in each case the actions restore the first three invariants, and that any **G**-set that changes has diameter less than 1, so that Lemmas 4 and 5 can be used to restore the last two invariants.

Case Insert-A:
Condition: There exists a set S such that $\ell(S) < x(p) < r(S)$.
Action: Add p to S.

Proof. (1) Holds by the condition. (2), (3), (4) and (5) Hold because no intervals or set types change.

If p does not lie in the interior of some existing set's interval, then we can define the sets left and right of p, which we denote $\lambda(p)$ and $\rho(p)$.

Case Insert-B:
Condition: $\ell(\rho(p)) - x(p) > 1$.
Action: Make p into a singleton set S with $type(S) = \mathbf{R}$.

Proof. (1) No change for sets not equal to S; trivial for S, which is a singleton **R**-set. (2) The condition ensures the interval of S is disjoint from the other set intervals. (3) Trivial.

Case Insert-C:
Condition: $type(\lambda(p)) = \mathbf{G}$.
Action: Add p into $\lambda(p)$.

Proof. (1) Trivial for **G**-sets. (2) By the definition of $\lambda(p)$, no other set intersects the extension to $\lambda(p)$'s interval. (3) No change to set types. (4), (5) Because Invariant 4 previously held for $\lambda(p)$, we have $\ell(\rho(p)) - \ell(\lambda(p)) < 1$, and therefore $diam(\lambda(p) \cup p) < 1$.

Case Insert-D:
Condition: $type(\lambda(p)) = \mathbf{R}$ and $type(\rho(p)) \neq \mathbf{G}$.
Action: Make p into a singleton set S with $type(S) = \mathbf{G}$.

Proof. (1) Trivial for **G**-sets. (2) p is not inside an existing interval, so S does not violate the invariant. (3) By construction. (4), (5) A singleton **G**-set has zero diameter.

Case Insert-E:
Condition: $type(\lambda(p)) = \mathbf{R}$, $type(\rho(p)) = \mathbf{G}$ and $r(\rho(p)) - x(p) < 1$.
Action: Add p into $\rho(p)$.

Proof. (1) Trivial for **G**-sets. (2) From the condition. (3) No change to set types. (4), (5) By the condition, $diam(p \cup \rho(p)) < 1$.

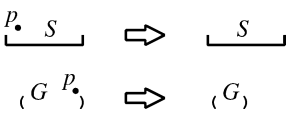
Case Insert-F:
Condition: $\ell(\rho(p)) - x(p) < 1$, $type(\lambda(p)) = \mathbf{R}$, $type(\rho(p)) = \mathbf{G}$ and $r(\rho(p)) - x(p) > 1$.
Action: Let $q = rpt(\rho(p))$. Remove q from $\rho(p)$ and make it into a singleton **G**-set. Change $type(\rho(p) \setminus q)$ to **L**. Make p into a singleton **G**-set.

Proof. Let S be the new **L**-set. (1) Trivial for **G**-sets; by construction and Lemma 1 the unit interval of S contains all the points of $\rho(p)$ except q. (2) Condition implies that S is interior-disjoint from the singleton **G**-sets, which are disjoint from their neighbors. (3) By inspection, and because it previously held. (4), (5) Both singleton **G**-sets have zero diameter.

Appendix D. Proofs of invariant restoration after a deletion

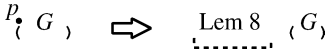
Here are the correctness proofs for the easy cases of point deletion from Fig. 7. Point p is deleted from a set S, with $|S| > 1$ and either $type(S) \neq \mathbf{R}$ or $p \neq lpt(S)$.

Case Delete-A:
Condition: $|S| > 1$, $type(S) = \mathbf{L}$ or $p \neq lpt(S)$.
Action: No action needed.



Proof. No intervals change if $type(S) \in \{\mathbf{R}, \mathbf{L}\}$, or if $type(S) = \mathbf{G}$ and $p \neq rpt(S)$. In this case no invariants can be violated. If $type(S) = \mathbf{G}$ and $p = rpt(S)$, $r(S)$ changes, but since none of the invariants depends on $r(S)$, they all continue to hold.

Case Delete-B:
Condition: $|S| > 1$, $p = lpt(S)$, $type(S) = \mathbf{G}$.
Action: Apply Lemma 8 to $\lambda(S)$.



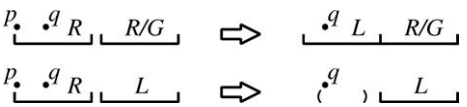
Proof. (1), (2) and (3) Unaffected by changes to $\ell(S)$. (4), (5) $diam(S)$ decreases, so Lemmas 4 and 5 can be applied to restore Invariants 4 and 5 if necessary.

Here are the proofs for the algorithm of Fig. 8, which restores invariants for S and sets to its right, assuming $|S| > 1$, $type(S) = \mathbf{R}$ and $p = lpt(S)$. We define $S' = S \setminus p$ and $q = lpt(S')$.

Case Delete-C:
Condition: $p = lpt(S)$, $|S| > 1$, $type(S) = \mathbf{R}$, and $\ell(\rho(S)) - x(q) > 1$.
Action: Set $type(S')$ to **R**.

Proof. (1) Previously held for S. Holds *a fortiori* for S', since $S' \subset S$. (2) By the condition. (3) No change. (4), (5) $diam(\rho(S))$ is unchanged, so Lemmas 4 and 5 can be applied if necessary.

Case Delete-D:
Condition: $p = lpt(S)$, $|S| > 1$, $type(S) = \mathbf{R}$ and $\ell(\rho(S)) - x(q) < 1$.
Action: If $type(\rho(S)) \neq \mathbf{L}$, set $type(S')$ to **L**; otherwise, set $type(S')$ to **G**.



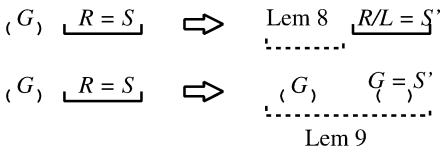
Proof. (1) Trivial if $type(S') = \mathbf{G}$; by the condition otherwise. (2) No violation for $\lambda(S)$, since $\ell(S) < \ell(S')$. No violation for S'—if $type(S') = \mathbf{G}$, its interval is contained in that of S; if $type(S') = \mathbf{L}$, the invariant holds by the definition of **L**-sets. (3) By inspection for S'. Violations for $\lambda(S)$ will be corrected later. (4), (5) $diam(\rho(S')) < 1$, so Lemmas 4 and 5 can be applied if necessary.

Here are the proofs for the algorithm of Fig. 9, which restores invariants for sets left of $S' = S \setminus p$, after p is deleted from a non-singleton set S.

Case Delete-E:
Condition: $type(\lambda(S)) = \mathbf{R}$.
Action: No action needed.

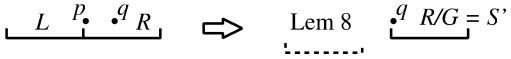
Proof. (1) No change. (2) The distance from $\lambda(S)$ to S' increased. (3) Trivial. (4), (5) Do not apply.

Case Delete-F:
Condition: $type(\lambda(S)) = \mathbf{G}$.
Action: If $type(S') \neq \mathbf{G}$ then apply Lemma 8 to $\lambda(S)$. Otherwise ($type(S') = \mathbf{G}$), apply Lemma 9 to $\lambda(S)$ and S'.



Proof. By Lemma 8 or Lemma 9. (If $type(S') = \mathbf{G}$, the former unit interval of S contains S', so S' is protected.)

Case Delete-G:
Condition: $type(\lambda(S)) = \mathbf{L}$, $type(S') \neq \mathbf{L}$.
Action: Apply Lemma 8.



Proof. By Lemma 8.

Case Delete-H:
Condition: $type(\lambda(S)) = \mathbf{L}$, $type(S') = \mathbf{L}$, $type(\lambda(\lambda(S))) = \mathbf{R}$.
Action: Change $type(\lambda(S))$ to **G**.

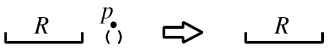
Proof. (1) Trivial for **G**-sets. (2) The interval of a **G**-set is contained in the interval of an **L**-set on the same points. (3) By inspection. (4), (5) $diam(\lambda(S)) < 1$ is unchanged, so Lemmas 4 and 5 can be applied if necessary.

Case Delete-I:
Condition: $type(\lambda(S)) = \mathbf{L}$, $type(S') = \mathbf{L}$, $type(\lambda(\lambda(S))) = \mathbf{G}$.
Action: Apply Lemma 9 to $\lambda(\lambda(S))$ and $\lambda(S)$.

Proof. By Lemma 9.

Here are the proofs that the algorithm of Fig. 10 restores the invariants after a singleton **G**-set is deleted.

Case Delete-J:
Condition: $type(\lambda(S)) = \mathbf{R}$.
Action: No action needed.



Proof. (1), (2) No change. (3) Holds *a fortiori*. (4), (5) Do not apply, since $type(\rho(S)) \neq \mathbf{G}$.

Case Delete-K:
Condition: $type(\lambda(S)) = \mathbf{L}$, $type(\rho(S)) = \mathbf{R}$.
Action: Apply Lemma 8 to $\lambda(S)$.

Proof. By Lemma 8.

Case Delete-L:
Condition: $type(\lambda(S)) = type(\rho(S)) = \mathbf{L}$, $type(\lambda(\lambda(S))) = \mathbf{R}$.
Action: Change $type(\lambda(S))$ to **G**.

Proof. (1) Trivial for **G**-sets. (2) *A fortiori*, because the interval of $\lambda(S)$ shrinks. (3) By inspection. (4), (5) $diam(\lambda(S)) < 1$ is unchanged, so Lemmas 4 and 5 can be applied if necessary.

Case Delete-M:
Condition: $type(\lambda(S)) = type(\rho(S)) = \mathbf{L}$, $type(\lambda(\lambda(S))) = \mathbf{G}$.
Action: Apply Lemma 9 to $\lambda(\lambda(S))$ and $\lambda(S)$.

Proof. By Lemma 9.

References

[1] M. Bern, D. Eppstein, Approximation algorithms for geometric problems, in: D.S. Hochbaum (Ed.), Approximation Algorithms for NP-Hard Problems, PWS Publishing Company, Boston, MA, 1997, pp. 296–345.
[2] R.J. Fowler, M.S. Paterson, S.L. Tanimoto, Optimal packing and covering in the plane are NP-complete, Inform. Process. Lett. 12 (3) (1981) 133–137.
[3] D.S. Hochbaum, W. Maas, Approximation schemes for covering and packing problems in image processing and VLSI, J. ACM 32 (1985) 130–136.
[4] H.B. Hunt, H. Marathe, V. Radhakrishnan, S. Ravi, D. Rosenkrantz, R. Stearns, NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs, J. Algorithms 26 (2) (1998) 238–274.
[5] S. Bespamyatnikh, B. Bhattacharya, D. Kirkpatrick, M. Segal, Mobile facility location, in: 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing & Communications, 2000, pp. 46–53.
[6] S. Basagni, Distributed clustering for ad hoc networks, in: Proc. '99 International Symp. on Parallel Architectures, Algorithms, and Networks (I-SPAN'99), 1999, pp. 310–315.
[7] M. Gerla, J. Tsai, Multicluster, mobile, multimedia radio network, J. Wireless Networks 1 (3) (1995) 255–265.

[8] J. Haartsen, M. Naghshineh, J. Inouye, O. Joeressen, W. Allen, Bluetooth: Vision, goals, and architecture, Mobile Computing and Communications Review 2 (4) (1998) 38–45.
[9] J. Gao, L.J. Guibas, J. Hershberger, L. Zhang, A. Zhu, Geometric spanner for routing in mobile networks, in: ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc), 2001, pp. 45–55.
[10] S. Marti, T.J. Giuli, K. Lai, M. Baker, Mitigating routing misbehavior in mobile ad hoc networks, in: Proc. 6th Annual International Conference on Mobile Computing and Networking, 2000, pp. 255–265.
[11] J. Gao, L.J. Guibas, J. Hershberger, L. Zhang, A. Zhu, Discrete mobile centers, in: Proc. 17th Annu. ACM Sympos. Comput. Geom., 2001, pp. 188–196.
[12] J. Basch, L. Guibas, J. Hershberger, Data structures for mobile data, J. Alg. 31 (1) (1999) 1–28.
[13] S. Har-Peled, Clustering motion, in: Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci., 2001, pp. 84–93.
[14] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The R*-tree: An efficient and robust access method for points and rectangles, in: Proc. ACM SIGMOD Conf. on Management of Data, 1990, pp. 322–331.
[15] G. Kollios, D. Gunopulos, V.J. Tsotras, On indexing mobile objects, in: Proc. ACM Sympos. Principles Database Syst., 1999, pp. 261–272.
[16] S. Šaltenis, C.S. Jensen, S.T. Leutenegger, M.A. Lopez, Indexing the positions of continuously moving objects, in: Proc. ACM SIGMOD Intl. Conf. on Management of Data, 2000, pp. 331–342.
[17] P.K. Agarwal, S. Har-Peled, C.M. Procopiuc, STAR-tree: An efficient self-adjusting index for moving objects, in: Proc. 4th Workshop on Algorithm Engineering and Experiments (ALENEX), 2002, pp. 178–193.
[18] P.K. Agarwal, J. Gao, L.J. Guibas, Kinetic medians and kd-trees, in: Proc. 10th Annu. European Sympos. Algorithms, Lecture Notes in Computer Science, vol. 2461, Springer-Verlag, Berlin, 2002, pp. 5–16.
[19] H. Edelsbrunner, E.P. Mücke, Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms, ACM Trans. Graph. 9 (1) (1990) 66–104.
[20] P.K. Agarwal, S. Har-Peled, Maintaining the approximate extent measures of moving points, in: Proc. 12th ACM-SIAM Sympos. Discrete Algorithms, 2001, pp. 148–157.