Maintenance of a Piercing Set for Intervals with Applications

Matthew J. Katz[1*] Frank Nielsen[2] Michael Segal[3†]

[1]Department of Mathematics and Computer Science
Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel
[2]SONY Computer Science Laboratories Inc., FRL
3-14-13 Higashi Gotanda, Shinagawa-Ku, Tokyo 141-0022, Japan
[3]Department of Computer Science
University of British Columbia, Vancouver, B.C. V6T 1Z4, Canada

Abstract

We show how to efficiently maintain a minimum piercing set for a set $\mathcal{S}$ of intervals on the line, under insertions and deletions to/from $\mathcal{S}$. A linear-size dynamic data structure is presented, which enables us to compute a new minimum piercing set following an insertion or deletion in time $O(c(\mathcal{S}) \log |\mathcal{S}|)$, where $c(\mathcal{S})$ is the size of the new minimum piercing set. We also show how to maintain a piercing set for $\mathcal{S}$ of size at most $(1+\varepsilon)c(\mathcal{S})$, for $0 < \varepsilon \leq 1$, in $\bar{O}(\frac{\log |\mathcal{S}|}{\varepsilon})$ amortized time per update. We then apply these results to obtain efficient solutions to the following three problems: (i) the shooter location problem, (ii) computing a minimum piercing set for arcs on a circle, and (iii) dynamically maintaining a box cover for a d-dimensional point set.

1 Introduction

Let $\mathcal{S} = \{s_1 = [l_1, r_1], \ldots, s_n = [l_n, r_n]\}$ be a set of n intervals on the real line. An *independent subset* of $\mathcal{S}$ is a set of pairwise non-intersecting intervals of $\mathcal{S}$. Let $b(\mathcal{S})$ be the maximum size of an independent subset of $\mathcal{S}$. A *piercing set* for $\mathcal{S}$ is a set $\mathcal{P}$ of points on the real line, such that, for each interval $s_i \in \mathcal{S}$, $s_i \cap \mathcal{P} \neq \emptyset$. Let $c(\mathcal{S})$, the *piercing number* of $\mathcal{S}$, be the size of a minimum piercing set for $\mathcal{S}$. (In graph theory terminology, we are dealing with the *interval graph* defined by $\mathcal{S}$ that is obtained by associating a node with each of the intervals in $\mathcal{S}$, and by drawing edges between nodes whose corresponding intervals intersect. The number $b(\mathcal{S})$ is also called the *packing number* of $\mathcal{S}$, a piercing set is also called a *cut set*, and the number $c(\mathcal{S})$ is also called the *transversal number of* $\mathcal{S}$.)

Clearly $c(\mathcal{S}) \geq b(\mathcal{S})$, since $b(\mathcal{S})$ piercing points are needed in order to pierce all intervals in a maximum independent subset of $\mathcal{S}$. It is not difficult to see though that $b(\mathcal{S})$ piercing points are also sufficient in order to pierce all intervals in $\mathcal{S}$, thus $c(\mathcal{S}) = b(\mathcal{S})$, and a minimum piercing set for $\mathcal{S}$ can be found in time $O(n \log c(\mathcal{S}))$ (see [11]).

In this paper we deal with the problem where the set of intervals $\mathcal{S}$ is dynamic (i.e., from time to time a new interval is inserted into $\mathcal{S}$ or an interval is deleted from $\mathcal{S}$), and we wish to efficiently maintain a minimum (or nearly minimum) piercing set for $\mathcal{S}$. Assuming the size of $\mathcal{S}$

*Supported by the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities, and by an Intel research grant.

†Supported by the Pacific Institute for Mathematical Studies and by the NSERC research grant.

never exceeds n, we present two solutions: an exact solution and an approximate solution (which is based on the exact solution). In the exact solution, a new minimum piercing set for $\mathcal{S}$ is computed from the current minimum piercing set, following an insertion/deletion of an interval to/from $\mathcal{S}$. The computation time is $O(c \log n)$, where c is the size of the new minimum piercing set (which differs from the size of the current set by at most 1). More precisely, a linear-size data structure, representing the current set of intervals and its minimum piercing set, is used to compute the new minimum piercing set, following an insertion/deletion of an interval. The data structure is updated during the computation.

In the approximate solution, a piercing set for $\mathcal{S}$ of size at most $(1+\varepsilon)c(\mathcal{S})$ is maintained, for a given approximation factor ε, $0 < \varepsilon \leq 1$. The amortized cost of an update (for any sequence of updates following a preprocessing stage that requires $O(n \log n)$ time) is $\bar{O}(\frac{\log n}{\varepsilon})$. (Notice that the update cost varies from $O(\log n)$, for $\varepsilon = 1$ (i.e., a 2-approximation), to $O(c(\mathcal{S}) \log n)$, for $\varepsilon < 1/c(\mathcal{S})$ (i.e., an exact solution).) Both the exact and approximate solutions are presented in Section 2. In Section 3, we apply the above exact and approximate solutions to obtain efficient solutions to the problems below.

The shooter location problem. Given a set of n disjoint segments in the plane, find a location p in the plane, for which the number of shots needed to hit all segments is minimal, where a shot is a ray emanating from p. This problem was first introduced by Nandy et al. [10], who observed that solving the problem for a given location p is equivalent to finding a minimum piercing set for a set of n arcs on a circle. The latter problem can be solved in time $O(n \log n)$, see below. They also presented an $O(n^3)$-time algorithm for the case where the shooter is allowed to move along a given line, and left open the general problem. Wang and Zhu [15] obtained an $O(n^5 \log n)$-time solution for the general problem. They also gave an $O(n^5)$-time algorithm for computing a 2-approximation, that is, a location for which the number of required shots is at most twice the optimal number of shots. Recently, Chaudhuri and Nandy [1] presented an improved solution for the general problem; its worst-case running time is $O(n^5)$, but it is expected to perform better in practice. Actually, the general problem can be solved by applying the solution for a shooter on a line to the $O(n^2)$ lines defined by the endpoints of the segments, thus obtaining an alternative $O(n^5)$-time solution.

Here we obtain an $O(\frac{1}{\varepsilon} n^4 \log n)$-time algorithm for computing a $(1+\varepsilon)$-approximation for the general problem (with possibly one extra shot), significantly improving the $O(n^5)$ 2-approximation of [15]. We can also find a location for which the number of shots is at most $r^* + 1$, where r^* is the optimal number of shots, in (output-sensitive) $O(n^4 r^* \log n)$ time. Finally, we describe another, more complicated, method for computing a $(1+\varepsilon)$-approximation. This method uses cuttings to compute a $(1+\varepsilon)$-approximation in $O(\frac{1}{\varepsilon^3} \frac{n^4 \log n}{r^*})$ time, thus breaking the $O(n^4)$ barrier for most values of r^*, assuming ε is a constant.

A minimum piercing set for arcs on a circle. Let $\mathcal{A} = \{a_1, ..., a_n\}$ be a set of n arcs on the unit circle centered at the origin. As for intervals on the line, a set $\mathcal{P}$ of points on the circle is a *piercing set* for $\mathcal{A}$, if for each arc $a_i \in \mathcal{A}$, $a_i \cap \mathcal{P} \neq \emptyset$. We wish to compute a minimum piercing set for $\mathcal{A}$. (In graph theory terminology, we are dealing with the *circular-arc graph* for $\mathcal{A}$ that is obtained by associating a node with each of the circular arcs in $\mathcal{A}$, and by drawing edges between nodes whose corresponding circular arcs intersect. We denote this graph by $G(\mathcal{A})$.)

Observe that now, unlike in the case of intervals on the line, the packing number $b(\mathcal{A})$ and the piercing number $c(\mathcal{A})$ may differ. (Assume, for example, that $\mathcal{A}$ consists of three arcs, each of length $2\pi/3$, that together cover the circle, then $b(\mathcal{A}) = 1$ while $c(\mathcal{A}) = 2$.) However, it is easy to see (Claim 8) that in this case either $c(\mathcal{A}) = b(\mathcal{A})$ or $c(\mathcal{A}) = b(\mathcal{A}) + 1$.

A point p on the unit circle induces a clique $\{a_i \in \mathcal{A} \mid p \in a_i\}$ of the graph $G(\mathcal{A})$. Notice that $G(\mathcal{A})$ might also have cliques whose arcs do not share a point (as in the example above). Cliques of the former type are called *linear* cliques. Assume we wish to find a minimum number of cliques of

$G(\mathcal{A})$ whose union is $\mathcal{A}$ (the clique covering problem). Hsu and Tsai [7] and Rao and Rangan [12] showed that if $\mathcal{A}$ itself is not a clique, then it suffices to consider only linear cliques. Thus, if $\mathcal{A}$ is not a clique, the problem of finding a minimum piercing set for $\mathcal{A}$ is essentially equivalent to the problem of finding a minimum number of cliques of $G(\mathcal{A})$ whose union is $\mathcal{A}$.

Golumbic and Hammer [5], Hsu and Tsai [7], Lee et al. [8], and Masuda and Nakajima [9] gave $O(n \log n)$-time algorithms for computing a maximum independent set of a circular-arc graph with n arcs. Gupta et al. [6] gave an $\Omega(n \log n)$ lower bound for this problem (actually, for the simpler problem of computing a maximum independent set of an interval graph with n intervals). Lee et al. [8] gave an $O(n \log n)$-time algorithm for the minimum cut set (i.e., piercing set) problem together with an application to a facility location problem, and Hsu and Tsai [7] gave an $O(n \log n)$-time algorithm for the minimum number of cliques problem. More recently, Tsai and Lee [14] investigated the problem of finding k best cuts (i.e., k cuts for which the number of different arcs that are cut is maximal). They showed how this problem is related to a facility location problem. Daniels and Milenkovic [3] use piercing sets (which they call *hitting sets*) in connection with generating layouts for the clothing industry.

We provide yet another optimal $\Theta(n \log n)$-time algorithm for computing a minimum piercing set for $\mathcal{A}$. We believe that our algorithm is (at least conceptually) simpler than the previous algorithms. Moreover, we can maintain a piercing set for $\mathcal{A}$ of size at most $(1+\varepsilon)c(\mathcal{A})+1$ in amortized update time $\bar{O}(\frac{\log n}{\varepsilon})$.

Maintenance of a box cover. Let $\mathcal{Q}$ be a set of n points in $\mathbb{R}^d$. A *cover* for $\mathcal{Q}$ is a set of (axis-parallel) unit hypercubes whose union contains $\mathcal{Q}$. The problem of computing a minimum cover is known to be NP-complete [4], and is dual to the following piercing problem. Given a set $\mathcal{B}$ of n unit hypercubes in $\mathbb{R}^d$, compute a minimum piercing set for $\mathcal{B}$. We present several efficient algorithms for dynamically maintaining a small piercing set for a set of arbitrary (axis-parallel) boxes in $\mathbb{R}^d$. We obtain an $O(c^* \log^d n)$ update-time algorithm for maintaining a piercing set of size c for arbitrary boxes, where $c \leq (1+\log_2 n)^{d-1} c^*$ and c^* denotes the optimal size, and an $O(2^{d-1} c^* \log n)$ update-time algorithm for maintaining a piercing set of size c for congruent boxes, where $c \leq 2^{d-1} c^*$. We can also obtain (in both cases) a trade-off between the update time and the approximation factor. These algorithms are based both on our dynamic data structures for intervals on the line, and on ideas from [11].

2 Maintenance of a Piercing Set for Intervals

2.1 Exact maintenance

Let $\mathcal{S}$ be a set of $m \leq n$ intervals on the line. We assume that from time to time a new interval is added to $\mathcal{S}$ or an existing interval is removed from $\mathcal{S}$. However, we require that at any moment $|\mathcal{S}| \leq n$. We show how to maintain a minimum piercing set for $\mathcal{S}$ under insertions and deletions in $O(c \log n)$ time, where c is the size of the new piercing set. We actually maintain a certain minimum piercing set which we call the *right-to-left piercing set* and which is defined as follows. Find the rightmost among the left endpoints of the intervals in $\mathcal{S}$. Let $s \in \mathcal{S}$ be the interval to which this endpoint belongs. Clearly the best location for a piercing point p in s is at its left endpoint. Remove all intervals that are pierced by p and reiterate. In this way we obtain a minimum piercing set for $\mathcal{S}$. The right-to-left piercing set can be computed easily in $O(n \log n)$ time. (Actually it can be computed in $O(n \log c(\mathcal{S}))$ time, see [11]). Initially, we compute the right-to-left piercing set $\mathcal{P}$ of $\mathcal{S}$.

We now construct a data structure of size $O(n)$ that will allow us to update the right-to-left piercing set within the claimed bound. For each piercing point $p \in \mathcal{P}$, let $\mathcal{S}_p$ be the subset of

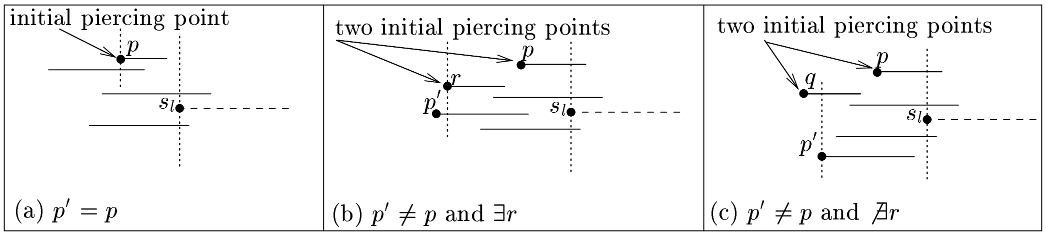

Figure 1: Three different cases that may occur during the insertion process. The dashed segment is the one being inserted.

intervals of $\mathcal{S}$ that were pierced by p during the right-to-left piercing process. These subsets are computed during the computation of $\mathcal{P}$. Notice that an interval $s \in \mathcal{S}$ is associated with the rightmost piercing point of $\mathcal{P}$ that lies in it. Construct a balanced binary search tree $\mathcal{T}$ on the piercing points in $\mathcal{P}$. For each node v in $\mathcal{T}$ representing a piercing point p, construct a balanced binary search tree $\mathcal{T}_p$ on the right endpoints of the intervals in $\mathcal{S}_p$, and let v point to the root of $\mathcal{T}_p$. With each node w in $\mathcal{T}_p$ we store the point l_w which is the rightmost among the left endpoints corresponding to the right endpoints in the subtree rooted at w. Notice that $l_{\text{root}} = p$. The overall construction time is $O(n \log n)$, and the resulting data structure is of size $O(n)$. We now describe the updating procedures for insertion and deletion of an interval.

2.1.1 Insertion

Let $s = [s_l, s_r]$ be a new segment to be added to $\mathcal{S}$. We first check, using the tree $\mathcal{T}$, whether s is already pierced by the current piercing set $\mathcal{P}$. If it is, then $\mathcal{P}$ is also the right-to-left piercing set of $\mathcal{S} \cup \{s\}$. We insert s into the tree $\mathcal{T}_p$, where p is the rightmost point in $\mathcal{P}$ that lies in s, and update the values l_w in the relevant nodes of $\mathcal{T}_p$. All these operations can be done in $O(\log n)$ time.

Assume now that $s \cap \mathcal{P} = \emptyset$. Notice that all the piercing points of $\mathcal{P}$ that lie to the right of s are also present in the right-to-left piercing set of $\mathcal{S} \cup \{s\}$ and their corresponding trees do not change. We first insert s_l as a new piercing point to the main tree $\mathcal{T}$. Next we need to create its corresponding tree $\mathcal{T}_{s_l}$. $\mathcal{T}_{s_l}$ should consist of the new segment s together with all segments in $\mathcal{S}$ that are pierced by s_l, but not by any other piercing point to the right of s_l. All these segments, however, must belong to $\mathcal{S}_p$, where p is the rightmost piercing point to the left of s_l. So we locate p in $O(\log n)$ time using $\mathcal{T}$, and search in $\mathcal{T}_p$ in $O(\log n)$ time for the leftmost right endpoint e that lies to the right of s_l. All the intervals in $\mathcal{T}_p$ whose right endpoint is to the right of e, including e, should be removed from $\mathcal{T}_p$ and added to $\mathcal{T}_{s_l}$. We must also update the values l_w in the relevant nodes of both trees. Below, we describe how to perform this transfer and update in a more general setting.

It is possible that the interval defining the point p has been transfered to $\mathcal{T}_{s_l}$. Let p' be the value that is currently stored in the root of $\mathcal{T}_p$, i.e., $l_{\text{root}} = p'$. (If p' does not exist, i.e., if $\mathcal{T}_p$ is empty, we simply delete p from $\mathcal{T}$ and stop.) If $p' = p$, then we are done, otherwise the interval defining p has been transfered and we replace the piercing point p by p' (see Figure 1).

We now have to check whether there is a piercing point (perhaps several of them) in $\mathcal{T}$ that lies to the right of p' and to the left of s_l. If the answer is positive, we consider the rightmost piercing point r in $\mathcal{T}$ that lies between p' and s_l. All right endpoints of the intervals that are currently stored in $\mathcal{T}_r$ are to the left of all right endpoints of the intervals currently stored in $\mathcal{T}_{p'}$. Thus, we can remove the point p' from $\mathcal{T}$ and transfer the intervals in the tree $\mathcal{T}_{p'}$ to the tree $\mathcal{T}_r$, by applying the *join* operation described below. We update the values l_w in $\mathcal{T}_r$ and stop. Otherwise, if the answer is negative, we need to locate the piercing point q that lies immediately to the left of p', and transfer the intervals of $\mathcal{T}_q$ that are pierced by p' to $\mathcal{T}_{p'}$. As before we search in $\mathcal{T}_q$ for the leftmost right endpoint e that lies to the right of p'. We need to transfer the intervals in $\mathcal{T}_q$ whose right endpoint is to the right of e, including e, to $\mathcal{T}_{p'}$. Observe that if s' is an interval in $\mathcal{T}_q$ whose right endpoint e' is to the right of e, including e, then e' lies to the left of all right endpoints in $\mathcal{T}_{p'}$, since otherwise $p \in s'$ and s' should already be in $\mathcal{T}_{p'}$ (which was obtained from $\mathcal{T}_p$). This property allows us to apply the standard *split* and *join* operations, see below, for first removing the intervals whose right endpoint e' is to the right of e, including e, from $\mathcal{T}_q$ (split) and then adding them to $\mathcal{T}_{p'}$ (join) in $O(\log n)$ time. We update the values l_w in both trees (see Figure 2). We continue in this way until we either reach a step in which the piercing point does not change (Figure 1(a)), or the case of Figure 1(b) occurs, or there are no more piercing points to the left of the piercing point. Clearly the whole insertion process takes only $O(c \log n)$ time, i.e., $O(\log n)$-time for the at most c cascading steps. A more careful analysis yields $O(c \log \frac{n}{c})$. (We apply Hölder's inequality to $\sum_{i=1}^{c} \log n_i$, where $\sum_i n_i = n$.)

2.1.2 Deletion

Let $s = [s_l, s_r]$ be an interval to be deleted from $\mathcal{S}$. We locate the rightmost piercing point p of $\mathcal{P}$ that lies in s. We distinguish between two cases. If $p \neq s_l$, then we remove s from $\mathcal{T}_p$, update the necessary l_w values and stop. This can be done in $O(\log n)$ time. The more difficult case is when $p = s_l$. In this case, we first remove s from $\mathcal{T}_p$ and update the necessary l_w values. We then replace p (in $\mathcal{T}$) by the value p' that is stored in the root of $\mathcal{T}_p$, which now becomes $\mathcal{T}_{p'}$. (If p' does not exist, we simply delete p from $\mathcal{T}$ and stop.) We proceed as described in the insertion procedure, that is, we either locate the rightmost piercing point r which lies to the right of p' and to the left of s_l (if such a point exists) and transfer the intervals of $\mathcal{T}_{p'}$ to $\mathcal{T}_r$ thus removing p', or we locate the piercing point q that lies immediately to the left of p', and transfer the intervals of $\mathcal{T}_q$ that are pierced by p' to $\mathcal{T}_{p'}$, and so on. The overall time spent on a deletion operation is thus $O(c \log n)$.

Theorem 1 *Let $\mathcal{S}$ be a set of intervals on a line, and assume that the size of $\mathcal{S}$ never exceeds n. It is possible to construct, in time $O(n \log n)$, a data structure of size $O(n)$, that enables us to maintain a minimum piercing set for $\mathcal{S}$, under insertions and deletions of intervals to/from $\mathcal{S}$, in time $O(c \log \frac{n}{c})$ per update, where c is the size of the current minimum piercing set for $\mathcal{S}$.*

Notice that we can use the data-structure above to maintain a maximum independent subset of $\mathcal{S}$. (The subset of intervals corresponding to the points of the minimum piercing set, i.e., the intervals whose left endpoint is a piercing point, is such a subset.)

Theorem 2 *Let $\mathcal{S}$ be a set of intervals on a line, and assume that the size of $\mathcal{S}$ never exceeds n. It is possible to construct, in time $O(n \log n)$, a data structure of size $O(n)$, that enables us to maintain a maximum independent set of $\mathcal{S}$, under insertions and deletions of intervals to/from $\mathcal{S}$, in time $O(b \log \frac{n}{b})$ per update, where b is the size of the current maximum independent set of $\mathcal{S}$.*

2.1.3 Joining and splitting trees

We now describe how to implement the split and join operations that are used by the algorithms for insertion and deletion above.

Joining trees. Let A_1 and A_2 be two sets of keys, such that all the keys in A_1 are smaller than i, and all the keys in $\mathcal{A}_2$ are greater than i, for some key i. Let $\mathcal{T}_{A_1}$ and $\mathcal{T}_{A_2}$ be the balanced binary search (red-black) trees for the sets A_1 and A_2, respectively. The *join* operation $join(A_1, i, A_2)$,

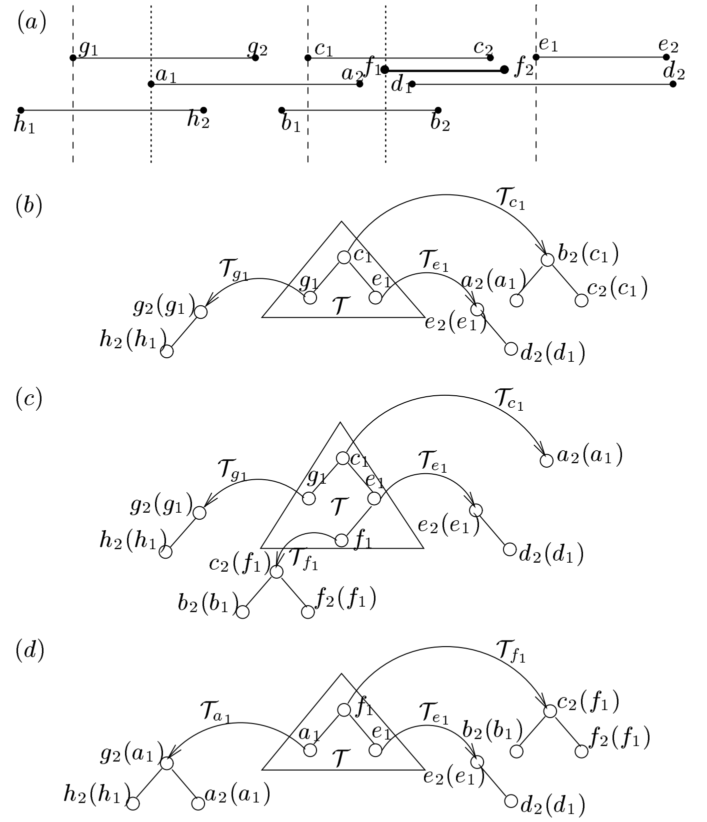

Figure 2: Inserting a new interval. (a) The initial interval set together with the new interval $[f_1, f_2]$; the initial piercing set is $\mathcal{P} = \{e_1, c_1, g_1\}$. (b) The initial data structure. (c) $[f_1, f_2] \cap \mathcal{P} = \emptyset$ and therefore f_1 is added to $\mathcal{T}$ and $\mathcal{T}_{f_1}$ is created. (d) c_1 is replaced by a_1, the value at the root of $\mathcal{T}_{c_1}$ (which now becomes $\mathcal{T}_{a_1}$); finally g_1 is removed from $\mathcal{T}$ since $\mathcal{T}_{g_1}$ is empty; the new piercing set is $\{e_1, f_1, a_1\}$.

described by Tarjan [13], takes $\mathcal{T}_{A_1}$, the key i, and $\mathcal{T}_{A_2}$, and returns the balanced binary search tree $\mathcal{T}_{(A_1 \cup \{i\} \cup A_2)}$ for the set $A_1 \cup \{i\} \cup A_2$. In our case, i stands for the smallest value in the tree $\mathcal{T}_{p'}$. The cost of Tarjan's join operation is $O(\log n)$. Moreover, within the same time bound we can update the values l_w wherever needed.

Splitting trees. Let A be a set of keys, i some key that belongs to A, and $\mathcal{T}_A$ a balanced binary search (red-black) tree for A. The *split* operation *split*(A, i), described in [13], takes $\mathcal{T}_A$ and i and returns two balanced binary search trees: $\mathcal{T}_{A_1}$ for all members of A that are smaller than i, and $\mathcal{T}_{A_2}$ for all members of A that are greater than i. In our case, i stands for the right endpoint e in the description of the algorithms for insertion and deletion. The cost of Tarjan's split operation is $O(\log n)$. Moreover, within the same time bound we can update the values l_w wherever needed.

2.2 Approximate maintenance

We now show how to maintain a piercing set $\mathcal{P}'$ for $\mathcal{S}$, where $\mathcal{S}$ is as above, such that $|\mathcal{P}'| \leq (1 + \varepsilon)c(\mathcal{S})$, for any fixed $0 < \varepsilon \leq 1$. The amortized cost per update is $\bar{O}(\frac{\log n}{\varepsilon})$, for any sequence of insertions and deletions, which begins immediately after a preprocessing stage in which the right-to-left (minimum) piercing set $\mathcal{P}$, $|\mathcal{P}| = c_0$, for $\mathcal{S}$ is computed and some additional work, that does not affect the time bound for this stage, is done. (Of course, we continue to assume that at any time $|\mathcal{S}| \leq n$.)

The key idea is to avoid long cascades by fixing stopping points, which are points in $\mathcal{P}'$, such that, a cascade cannot continue beyond a stopping point. Initially, we set $\mathcal{P}' = \mathcal{P} = (p_1, \ldots, p_{c_0})$, and $p_1, p_{1+\lceil\frac{2}{\varepsilon}\rceil}, p_{1+2\lceil\frac{2}{\varepsilon}\rceil}, \ldots$ are the stopping points. The stopping points partition the sequence of piercing points into at most $\lceil\frac{\varepsilon}{2}c_0\rceil$ groups, each of size at most $\lceil\frac{2}{\varepsilon}\rceil$. (The first group begins with p_{c_0} and ends with the first stopping point from the right, the second group begins with the point immediately to the left of this stopping point and ends with the second stopping point from the right, and so on.) Roughly, at any time, each of the groups consists of the right-to-left piercing set for the subset of intervals associated with the points in the group. An insertion or a deletion of an interval can only affect a single group, which now has to adapt to the change in the subset of intervals associated with its points.

A stopping point is never deleted (in between clean-up stages, see below), even if it is not needed as a piercing point any more. One can think of a stopping point as a degenerate (dummy) interval. But, whenever the size of a group reaches twice its initial size, i.e. $2\lceil\frac{2}{\varepsilon}\rceil$, it is split into two, by making the point in position $\lceil\frac{2}{\varepsilon}\rceil$ in the group a new stopping point. This guarantees an update cost of $\bar{O}(\frac{\log n}{\varepsilon})$ time.

In this way, we can ensure for a while that $\mathcal{P}'$ is a $(1+\varepsilon)$-approximation. However, after performing a sequence of $\frac{\varepsilon}{4}c_0$ insertions and deletions, we need to perform a clean-up stage (see below), in which we reset $\mathcal{P}'$ to the current right-to-left piercing set of $\mathcal{S}$. This stage requires $O(c_0 \log n)$ time, which is divided among the updates in the sequence. Below, we describe the insertion and deletion operations and then analyze our approximation scheme.

2.2.1 Insertion

Let $s = [s_l, s_r]$ be a new interval to be added to $\mathcal{S}$. We check in $O(\log n)$ time whether s is already pierced by a point in $\mathcal{P}'$. If yes, we insert s in $O(\log n)$ time, associating it with the rightmost point in $\mathcal{P}'$ that lies in it, as in the exact scheme. If not, we add s_l as a new piercing point to $\mathcal{P}'$, and begin the iterative process (which we call a cascade) that was described in Section 2.1.1. This process can either end naturally, before the group's stopping point is encountered, or artificially, upon reaching this stopping point. The number of points in the group may increase by 1, and if it has reached $2\lceil\frac{2}{\varepsilon}\rceil$, we split it into two equal size groups by making the point in position $\lceil\frac{2}{\varepsilon}\rceil$ in the group a new stopping point. The length of the cascade is thus less than $2\lceil\frac{2}{\varepsilon}\rceil$, and therefore the cost of an insertion is $O(\frac{\log n}{\varepsilon})$.

2.2.2 Deletion

Let $s = [s_l, s_r]$ be an interval to be deleted from $\mathcal{S}$, and let p be the rightmost point in $\mathcal{P}'$ that lies in s. If $p \neq s_l$, we simply remove the segment s in $O(\log n)$ time from p's tree, as in the exact scheme. If, however, $p = s_l$, we begin the iterative process described in Section 2.1.2, which either stops naturally, or when the group's stopping point is encountered. The cost of a deletion is thus $O(\frac{\log n}{\varepsilon})$. In both cases, if p is a stopping point, we simply remove s without replacing or deleting p, even if p's tree is empty.

2.2.3 The clean-up stage

In order to ensure that we remain with a $(1+\varepsilon)$-approximation after each update, we need to perform a clean-up stage following a sequence of $\frac{\varepsilon}{4}c_0$ updates. The clean-up stage brings us back to the initial state, where $\mathcal{P}'$ is the right-to-left piercing set for $\mathcal{S}$, and the stopping points are the points of $\mathcal{P}'$ in position $1, 1+\lceil\frac{2}{\varepsilon}\rceil, 1+2\lceil\frac{2}{\varepsilon}\rceil, \ldots$ The clean-up requires only $O(c_0 \log n)$ time (unlike the initial preprocessing stage which requires $O(n \log n)$ time), so if we divide it over the last sequence of updates, we obtain the claimed $\bar{O}(\frac{\log n}{\varepsilon})$ amortized cost per update.

The situation just before the clean-up is that each interval is stored with the rightmost point in $\mathcal{P}'$ that lies in it. However, there may be piercing points (among the stoppers) whose corresponding set of intervals is empty, and there may be piercing points (among the stoppers) for which the value l_{root} at the root of their tree is different from the piercing point itself.

In the clean-up stage we perform a right-to-left traversal, beginning at the rightmost stopper in $\mathcal{P}'$. During the traversal the various cases which are described in Section 2.1.1 occur, and we handle them accordingly.

If p is of the first type above, then we delete it, and jump to the next stopper q. Otherwise, let p' be the value stored at the root of p's tree. If $p' = p$, then we jump to q, and if $p' \neq p$, then we proceed as follows. If p' is to the left of r, the point immediately to the left of p, then we transfer the intervals in p's tree to r's tree, delete p, and jump to q. Otherwise, we replace p with p', and start a cascade as in Section 2.1.1. We then jump to the first stopper following the cascade.

At the end of this process $\mathcal{P}'$ is again the right-to-left minimum piercing set for $\mathcal{S}$ and we update the value of c_0. The whole process requires only $O(c_0 \log n)$ time.

2.2.4 The analysis

We have to show that $\mathcal{P}'$ is a $(1+\varepsilon)$-approximation after each update. At time t (i.e., after the t'th update), the size c_t of the minimum piercing set and the size c'_t of $\mathcal{P}'$ are surely in between $c_0 - \frac{\varepsilon}{4}c_0$ and $c_0 + \frac{\varepsilon}{4}c_0$. Thus, even in the worst case, where c_t is equal to the minimum value and c'_t is equal to the maximum value, we have

$$c'_t = (1+\frac{\varepsilon}{4})c_0 \leq (1+\frac{3\varepsilon}{4}-\frac{\varepsilon^2}{4})c_0 = (1+\varepsilon)c_t \ ,$$

so $\mathcal{P}'$ is indeed a $(1+\varepsilon)$-approximation.

We obtain the following theorem:

Theorem 3 *For any $0 < \varepsilon \leq 1$, we can maintain a $(1+\varepsilon)$-approximation of a minimum piercing set for $\mathcal{S}$ in amortized update time $\bar{O}(\frac{\log n}{\varepsilon})$.*

As a corollary, we obtain the following theorem concerning the size $b(\mathcal{S})$ of a maximum independent subset of $\mathcal{S}$.

Theorem 4 *For any $0 < \varepsilon \leq 1$, we can maintain a $(1+\varepsilon)$-approximation of the size $b(\mathcal{S})$ of a maximum independent subset of $\mathcal{S}$ in amortized update time $\bar{O}(\frac{\log n}{\varepsilon})$. (That is, at time t, $\frac{c'_t}{1+\varepsilon} \leq b(\mathcal{S}) \leq c'_t$.)*

3 Applications

In this section we present the three applications that were mentioned in Section 1. See Section 1 for a survey of related previous results.

3.1 Shooter location problem

In the *Shooter Location Problem* (SLP for short), we are given a set $\mathcal{S} = \{s_1, \ldots, s_n\}$ of n disjoint segments in the plane, and we seek a point p from which the number of shots needed to hit all segments in $\mathcal{S}$ is minimal, where a shot is a ray emanating from p.

A $(1+\varepsilon)$-approximation. Let $\mathcal{L}$ be the set of $O(n^2)$ lines defined by the endpoints of the segments in $\mathcal{S}$. Consider any cell f of the arrangement $\mathcal{A}(\mathcal{L})$, and let p be a point in the interior of f. The number of shots from p needed to hit all segments in $\mathcal{S}$ is equal to the size of a minimum piercing set for the set of circular arcs obtained by projecting each of the segments in $\mathcal{S}$ on a circle enclosing all the segments in $\mathcal{S}$ and centered at p. For any other point p' in the interior of f, the number of shots from p' is equal to the number of shots from p, since the circular-arc graphs for p and for p' are identical. Moving from one cell of $\mathcal{A}(\mathcal{L})$ to an adjacent cell corresponds to a swap in the locations of two adjacent arc endpoints.

We traverse the arrangement $\mathcal{A}(\mathcal{L})$, dynamically maintaining an approximation of the minimum number of rays required to intersect all the segments from a point in the current cell. At each cell of $\mathcal{A}(\mathcal{L})$, we shoot a vertical ray directed upwards, allowing us to deal with the interval graph obtained by unrolling the cell's circular-arc graph (after removing the arcs that are intersected by the vertical ray). We use the data structure of Section 2.2 to maintain in amortized time $\bar{O}(\frac{\log n}{\varepsilon})$ a $(1+\varepsilon)$-approximation of the size of the minimum piercing set for this interval graph. At the end, we choose the cell for which the number computed is the smallest. (Actually, this scheme will also work for segments that are not necessarily disjoint.)

Theorem 5 *For any fixed $0 < \varepsilon \leq 1$, a $(1+\varepsilon)$-approximation (with possibly one extra shot) for the shooter location problem can be found in $O(\frac{1}{\varepsilon} n^4 \log n)$ time.*

Towards an exact solution. We showed how to obtain a $(1+\varepsilon)$-approximation, that is, how to find a number r such that $r^* \leq r \leq (1+\varepsilon)r^* + 1$, where r^* is the optimal number of shots. Therefore, if $\varepsilon r^* < 1$, we obtain a location for which the number of rays is either optimal or optimal plus one. Since we need to choose $\varepsilon < \frac{1}{r^*}$ without knowing r^*, we first run the algorithm with, say, $\varepsilon = 1$, and obtain a number of rays $r^* \leq r' \leq 2r^* + 1$. Then we choose $\varepsilon = \frac{1}{r'} < \frac{1}{r^*}$ and run the algorithm to obtain the optimal, or optimal with one extra shot, solution in $\tilde{O}(n^4 r^* \log n)$ output-sensitive time (single bootstrapping).

Theorem 6 *The optimal number of shots r^* (with possibly one extra shot) of the shooter location problem can be computed in $O(n^4 r^* \log n)$ time.*

Avoiding the complete arrangement traversal. We now describe another method for obtaining a $(1+\varepsilon)$-approximation, which is often more efficient than the method described above. Let $\mathcal{L}'$ be a $\frac{1}{c}$-cutting of $\mathcal{L}$ (see [2]). That is, $\mathcal{L}'$ is a set of $O(c)$ lines, and each cell of the (vertical decomposition of the) arrangement $\mathcal{A}(\mathcal{L}')$ is cut by at most $\frac{|\mathcal{L}|}{c} \leq \frac{2n^2}{c}$ lines of $\mathcal{L}$. For each of these lines l, we dynamically compute a $(1+\delta)$-approximation for a shooter moving along l (in the original environment S). The total computation time is $O(n^2 c \frac{\log n}{\delta})$. Let r_{min} be the best score obtained during the computation. We have $r^* \leq r_{min} \leq (1+\delta)(r^* + \frac{2n^2}{c})$. (The right inequality holds since if $C \in \mathcal{A}(\mathcal{L})$ is the cell from which only r^* shots are needed, then there exists a cell $C' \in \mathcal{A}(\mathcal{L})$ that is supported by a line in $\mathcal{L}'$, such that, C can be reached from C' by passing through at most $\frac{2n^2}{c}$ cells of $\mathcal{A}(\mathcal{L})$.) By setting $c = \frac{2n^2}{\gamma r^*}$, for some $0 < \gamma < 1$, we obtain $r^* \leq r_{min} \leq (1+\delta)(1+\gamma)r^*$ in $O(\frac{1}{\delta\gamma} \frac{n^4 \log n}{r^*})$ time. We choose $\delta = \gamma = \frac{\varepsilon}{3}$ to ensure a $(1+\varepsilon)$-approximation scheme in $O(\frac{1}{\varepsilon^2} \frac{n^4 \log n}{r^*})$ time.

However, we do not know r^*, the size of the optimal solution, beforehand. We are going to approximate it by r' as follows. We first demonstrate the method for the special case where a 4-approximation is desired, and then present it for the general case.

For a 4-approximation, assume $\delta = \gamma = \frac{1}{4}$, and set $r' \leftarrow \frac{n}{2}$. Let $c = \frac{2n^2}{\gamma r'} = 16n$, and, as above, first compute a $\frac{1}{c}$-cutting of $\mathcal{L}$ and then, for each of the $O(c)$ lines in the cutting, compute a $(1+\delta)$-approximation for a shooter moving along the line. The total computation time is $O(n^3 \log n)$. By taking the minimum score r_{min} along the lines of the cutting, we have $r^* \leq r_{min} \leq (1+\frac{1}{4})(r^* + \frac{n}{8})$. Therefore, if $r_{min} \geq \frac{n}{2}$, then $r^* \geq \frac{11n}{40} \geq \frac{n}{4}$ and we return r_{min} and stop. This gives $\frac{r_{min}}{r^*} \leq \frac{n}{n/4} = 4$. Otherwise, we set $r' \leftarrow \frac{r'}{2}$ and repeat. We continue halving r' until at some stage $r_{min} \geq r'$ (and $r_{min} < 2r'$). At this stage we have $r^* \geq \frac{11r'}{20} \geq \frac{r'}{2}$, and $\frac{r_{min}}{r^*} \leq \frac{2r'}{r'/2} = 4$. The overall cost of this algorithm is bounded by $O(n^4 \log n) \sum_{r'} \frac{1}{r'}$ with $r' = \frac{n}{2^i}$ for $i \leq \log \frac{n}{r^*}$. Thus we end up with a 4-approximation in $O(\frac{n^4 \log n}{r^*})$ time.

For the general case, where a $(1+\varepsilon)$-approximation is desired, we set $r' = \beta n$, for an appropriate $0 < \beta < 1$, as our current estimate of r^*, and let $c = \frac{2n^2}{\gamma r'} = \frac{2n^2}{\gamma \beta n}$. After computing a $\frac{1}{c}$-cutting and r_{min} as before, we have

$$r^* \leq r_{min} \leq (1+\delta)(r^* + \frac{2n^2}{c}) = (1+\delta)(r^* + \gamma\beta n) \ .$$

Now, if $r_{min} \geq r'$ (i.e., if $r_{min} \geq \beta n$), then $(1+\delta)(r^* + \gamma\beta n) \geq \beta n$, which implies that

$$r^* \geq \frac{\beta n(1-(1+\delta)\gamma)}{1+\delta} \ .$$

Thus,

$$\frac{r_{min}}{r^*} \leq \frac{1+\delta}{\beta(1-(1+\delta)\gamma)} \ , \qquad (*)$$

since $r_{min} \leq n$.

If, however, $r_{min} < r'$, we set $r' \leftarrow \beta r'$, and repeat until at some stage $r_{min} \geq r'$. At this stage we have $\beta^i n \leq r_{min} < \beta^{i-1} n$, for some $i \geq 2$, and the ratio between r_{min} and r^* is as in the first stage (Equation $(*)$), this time using $r_{min} < \beta^{i-1} n$.

Therefore, we must pick δ, γ and β such that

$$\frac{1+\delta}{\beta(1-(1+\delta)\gamma)} \leq 1 + \varepsilon \ . \qquad (**)$$

The running time is

$$\frac{n^4 \log n}{\gamma\delta} \Sigma_i \frac{1}{\beta^i n} \ ,$$

with i ranging from 1 to $\log_{\frac{1}{\beta}} \frac{n}{r^*}$. That is,

$$\frac{n^3 \log n}{\gamma\delta} \sum_{i=1}^{\log_{\frac{1}{\beta}} \frac{n}{r^*}} (\frac{1}{\beta})^i \ .$$

But $\sum_{i=1}^{\log_{\frac{1}{\beta}} \frac{n}{r^*}} (\frac{1}{\beta})^i$ is less than $\frac{n}{(1-\beta)r^*}$. Therefore the running time for a $(1+\varepsilon)$-approximation is $O(\frac{n^4 \log n}{\delta\gamma(1-\beta)r^*})$. It is easy to verify that by picking $\gamma = \delta = \frac{\varepsilon}{5}$ and $\beta = 1 - \frac{\varepsilon}{5}$, Equation $(**)$ is satisfied (assuming $\varepsilon \leq 1$), and thus the running time becomes $O(\frac{1}{\varepsilon^3} \frac{n^4 \log n}{r^*})$.

Comparing this method with the first method, we see that this method is more efficient than the first method whenever $r^* \geq \frac{1}{\varepsilon^2}$.

Theorem 7 *A $(1+\varepsilon)$-approximation for the shooter location problem can be found in $O(\frac{1}{\varepsilon^3}\frac{n^4 \log n}{r^*})$ time.*

3.2 Minimum piercing set for circular arcs

Let $\mathcal{A} = \{a_1, \ldots, a_n\}$ be a set of n arcs on the unit circle C centered at the origin. Our goal is to compute a minimum piercing set $\mathcal{P} \subseteq C$ for $\mathcal{A}$.

Let c denote the size of a minimum piercing set for $\mathcal{A}$, and let b denote the maximum size of an independent subset of $\mathcal{A}$, that is, a subset of $\mathcal{A}$ whose arcs are pairwise disjoint. Clearly $c \geq b$, since we need b piercing points in order to pierce all arcs in a maximum independent subset of $\mathcal{A}$. For a set $\mathcal{S}$ of intervals on a line, it is easy to see ([11]) that $b(\mathcal{S})$ piercing points are also sufficient in order to pierce all intervals in $\mathcal{S}$. In our case, however, b piercing points may not be enough. For example, if $\mathcal{A}$ consists of three arcs obtained by cutting the circle C into three parts, then $b = 1$ while $c = 2$. It is easy to see though that the difference between b and c can never exceed 1. Place a piercing point p anywhere on the circle C and remove all arcs that are pierced by p. We can think of the remaining arcs as intervals on a line. The size of a maximum independent subset of these intervals is either b or $b-1$. Thus, in view of the remark above concerning intervals on a line, either $c = b+1$, or $c = b$. Therefore, we have:

Claim 8 *$b \leq c \leq b+1$, and there exists sets of arcs that require $b+1$ piercing points.*

For an arc $a \in \mathcal{A}$, let $f(a)$ be the number of arc endpoints that lie in a, including a's two endpoints. Let a^* be an arc in $\mathcal{A}$ such that $f(a^*) \leq f(a)$ for any other arc $a \in \mathcal{A}$. Clearly $f(a^*) \leq \lfloor \frac{2n}{b} \rfloor$, by the pigeon hole principle. We can find a^* in $O(n \log n)$ time: After sorting the endpoints by their polar angle, one can determine the number of endpoints lying in an arc a in $O(\log n)$ time.

The endpoints that lie in the interior of a^* together with a^*'s two endpoints divide a^* into $O(n/b)$ subarcs. Since a^* must be pierced, we traverse a^* from end to end moving from one subarc to an adjacent subarc. For each of these subarcs, we place in it a piercing point p, and compute a minimum piercing set for the remaining set of arcs that are not pierced by p (which can be viewed as a set of intervals on a line). The subarc whose corresponding minimum piercing set is the smallest, is then chosen as the subarc in which p is eventually placed, and the final piercing set is composed of p and the piercing set that was computed for this subarc. (Of course, if there exists a point of C that is not covered by $\mathcal{A}$, then we can simply treat the set $\mathcal{A}$ as a set of intervals on the line.)

During the traversal, when moving from one subarc to an adjacent subarc we either enter or leave an arc of $\mathcal{A}$. We can therefore use our data structure for maintaining a minimum piercing set for a set $\mathcal{S}$ of intervals on a line (see Section 2.1). Initially $\mathcal{S}$ is obtained from the arcs in $\mathcal{A}$ that are not pierced by a point lying in the first subarc of a^*. We construct our data structure for $\mathcal{S}$ in $O(n \log n)$ time. When moving from one subarc to an adjacent subarc, an interval is either inserted or deleted to/from $\mathcal{S}$. For any subarc of a^*, the number of intervals in $\mathcal{S}$ is at most $n-1$, the size of the minimum piercing set that is computed is at most $b+1$ (by Claim 8), and the computation time is $O(b \log n)$. Since there are $O(n/b)$ subarcs, we conclude that the total running time of our algorithm (for computing a minimum piercing set for $\mathcal{A}$) is $O(n \log n)$.

Theorem 9 *Let $\mathcal{A}$ be a set of n arcs on a circle. It is possible to compute a minimum piercing set for $\mathcal{S}$ in $O(n \log n)$ time.*

Remark: We can apply the approximation scheme of Section 2.2 in order to maintain a small piercing set for $\mathcal{A}$, under insertions and deletions of arcs to/from $\mathcal{A}$. Let p_0 be any point on the circle C. We maintain a $(1+\varepsilon)$ approximation for the set of intervals corresponding to the arcs in $\mathcal{A}$ that are not pierced by p_0. Thus, if $c(\mathcal{A})$ is the piercing number of $\mathcal{A}$, then we can maintain a piercing set for $\mathcal{A}$ of size at most $(1+\varepsilon)c(\mathcal{A})+1$ in amortized $\bar{O}(\frac{\log n}{\varepsilon})$ time per update.

3.3 Maintenance of a box cover

Let $\mathcal{Q}$ be a set of n points in d-space. We wish to compute a minimum cover for $\mathcal{Q}$, consisting of unit (axis-parallel) hypercubes. Dually, we wish to compute a minimum piercing set for a set $\mathcal{Q}^*$ of n d-dimensional unit hypercubes. These problems are referred to in the literature as the BOX COVERING and BOX PIERCING problems. For $d \geq 2$, these problems were shown to be NP-complete by Fowler et al. [4]. We consider the BOX PIERCING problem for arbitrary (axis-parallel) boxes and in a dynamic setting, where from time to time boxes are inserted and deleted from $\mathcal{Q}^*$. We begin with a simple observation, and then dynamize an approximation scheme presented in [11].

Let $\mathcal{S} = \{B_1, ..., B_n\}$ be a set of n boxes (i.e., hyperrectangles) in d-space, where each box is represented as the ordered Cartesian product $B_i = \prod_{j=1}^{d}[x_j^i, X_j^i]$. Let $\mathcal{P}_j^*$ denote a minimum piercing set for the interval set $\mathcal{S}_j = \{(x_j^i, X_j^i) |\ i \in [1..n]\ \}$. Clearly $\mathcal{P} = \prod_{j=1}^{d} \mathcal{P}_j^*$ is a piercing set for $\mathcal{S}$. Let c^* be the size of a minimum piercing set for $\mathcal{S}$. Since $|\mathcal{P}_j^*| \leq c^*$, for $j = 1, \ldots, d$, we conclude that $|\mathcal{P}| \leq (c^*)^d$. Therefore, by dynamically maintaining a piercing set for each of the sets $\mathcal{S}_j$ independently, we can maintain a piercing set for $\mathcal{S}$ of size at most $(c^*)^d$. The cost of an update is $O(dc^* \log n)$. Alternatively, we can maintain in $\bar{O}(d\frac{\log n}{\epsilon})$ amortized time per update a piercing set for $\mathcal{S}$ of size at most $(1+\varepsilon)^d (c^*)^d$.

A divide-and-conquer scheme proposed in [11] consists of finding the median x of the endpoints of the intervals in $\mathcal{S}_1$, and partitioning the set $\mathcal{S}$ into three subsets: (1) $\mathcal{S}' = \{\mathcal{S} \cap (\mathbf{x_1} = x)\}$, i.e., the set of $(d-1)$-dimensional boxes obtained from the boxes in $\mathcal{S}$ that are cut by the hyperplane $\mathbf{x_1} = x$, (2) $\mathcal{S}_l$, the boxes in $\mathcal{S}$ that are fully contained in the half-space $\mathbf{x_1} < x$, and (3) $\mathcal{S}_r$, the boxes in $\mathcal{S}$ that are fully contained in the half space $\mathbf{x_1} > x$. Now, a piercing set $\mathcal{P}$ for $\mathcal{S}$ is obtained by recursively computing piercing sets $\mathcal{P}_l, \mathcal{P}_r, \mathcal{P}'$ for the sets $\mathcal{S}_l, \mathcal{S}_r, \mathcal{S}'$, respectively, and by setting $\mathcal{P} = \mathcal{P}_l \cup \mathcal{P}_r \cup \mathcal{P}'$ (after converting the points in $\mathcal{P}'$ to d-dimensional points by adding a coordinate at the front whose value is x). (The 1-dimensional case is solved exactly in $\Theta(n \log n)$-time.) It can be shown that $|\mathcal{P}| \leq (1 + \log_2 n)^{d-1} c^*$ for arbitrary boxes, and $|\mathcal{P}| \leq 2^{d-1} c^*$ for congruent boxes; see [11] for proof and other bounds/tradeoffs.

The above scheme can be dynamized in a straightforward way, using our exact and approximate solutions for intervals. Indeed, whenever a box is inserted or deleted from $\mathcal{S}$, the median might change, but only locally (i.e., the new median is either the current median, or the endpoint immediately to the left/right of the current median), and thus the sets $\mathcal{S}_l, \mathcal{S}_r$, and $\mathcal{S}'$ can be easily maintained by adding and removing a constant number of boxes. Now, if $\mathcal{S}'$ consists of 1-dimensional boxes (i.e., intervals), we update the appropriate (exact or approximate) dynamic data structure, and proceed to handle the sets $\mathcal{S}_l$ and $\mathcal{S}_r$ recursively. Otherwise, we handle all three sets recursively.

We can view the whole process as a d-dimensional binary tree. At the root of the main tree we store the d-dimensional set $\mathcal{S}$, and the median x of the endpoints of the intervals in $\mathcal{S}_1$. The left child of the root stores the set $\mathcal{S}_l$, and the right child stores $\mathcal{S}_r$. In addition, the root points to the $(d-1)$-dimensional binary tree whose root stores the set $\mathcal{S}'$. For each of the 1-dimensional sets, we construct our exact (alternatively, approximate) dynamic piercing data structure for intervals. We observe that the depth of the main tree is $O(\log n)$, that the sets stored in the nodes of its k'th level are pairwise separable, and, therefore, the sum of the sizes of all minimum (alternatively, nearly minimum) piercing sets computed by their descendant piercing structures is at most c^* (alternatively, $(1+\varepsilon)c^*$).

Insertions and deletions are handled similarly. We start at the root of the main tree, with the original update instruction (i.e., insert/delete some d-dimensional box B). At the current node v storing a k-dimensional set: If $k > 1$, we first update the set and median that are stored with v, according to the update instruction. Next, for each of the children of v and for the root of v's $(k-1)$-dimensional substructure, we issue an appropriate update instruction and treat each of them recursively. If $k = 1$, we update the piercing data structure associated with v, according to the update instruction.

Using the exact interval piercing data structure, we obtain:

Theorem 10 *We can maintain a piercing set of size c for a set $\mathcal{S}$, $|\mathcal{S}| \leq n$, of arbitrary (axis-parallel) d-dimensional boxes in $O(c \log n)$-time per update, where $c \leq (1 + \log_2 n)^{d-1} c^*$. For cubes we have $c \leq 2^{d-1} c^*$.*

As a corollary, we can maintain an approximation of the size $b(\mathcal{S})$ of a maximum independent subset of $\mathcal{S}$ (that is a maximum subset of $\mathcal{S}$ consisting of pairwise disjoint boxes), since $\frac{c}{(1+\log n)^{d-1}} \leq b(\mathcal{S}) \leq c$.

4 Conclusion

We developed a data structure for maintaining a minimum (or nearly minimum) piercing set for a set of intervals on a line. The efficiency of our maintenance schemes was demonstrated in the improved (approximate) solutions that were obtained to the shooter location problem. We also applied these maintenance schemes to obtain (conceptually) simpler algorithms for computing a minimum (or nearly minimum) piercing set for a set of arcs on a circle, and for maintaining a small piercing set for a set of d-dimensional boxes. A natural question is whether or not it is possible to maintain a minimum piercing set for intervals in $O(\log n)$ time (rather than $O(c \log n)$ time) per update. It would also be nice to be able to maintain in $O(c \log n)$ time per update a minimum (rather than minimum plus possibly 1) piercing set for arcs on a circle.

References

[1] J. Chaudhri and S.C. Nandy "Generalized shooter location problem", in *Lecture Notes in Computer Science* 1627, pp. 389–401, 1999.

[2] B. Chazelle "Cutting hyperplanes for divide-and-conquer", *Discrete Comput. Geom.* 9 (1993), pp. 145–158.

[3] K. Daniels and V. Milenkovic "Limited Gaps", in *Proc. 6th Canad. Conf. Comput. Geom.*, pp. 225–231, 1994.

[4] R. J. Fowler and M. S. Paterson and S. L. Tanimoto "Optimal packing and covering in the plane are NP-complete", *Information Processing Letters* 12(3) (1981), pp. 133–137.

[5] M. C. Golumbic and P. L. Hammer "Stability in circular arc graphs", *J. of Algorithms* 9 (1988), pp. 314–320.

[6] U. Gupta, D. T. Lee and Y.-T. Leung "Efficient algorithms for interval graphs and circular-arc graphs", *Networks* 12 (1982), pp. 459–467.

[7] W.-L. Hsu and K.-H. Tsai "Linear time algorithms on circular-arc graphs", *Information Processing Letters* 40 (1991), pp. 123–129.

[8] D. T. Lee, M. Sarrafzadeh and Y. F. Wu "Minimum cuts for circular-arc graphs", *SIAM J. Computing* 19(6) (1990), pp. 1041–1050.

[9] S. Masuda and K. Nakajima "An optimal algorithm for finding a maximum independent set of a circular-arc graph", *SIAM Journal on Computing* 17(1) (1988), pp. 41–52.

[10] S. C. Nandy and K. Mukhopadhyaya and B. B. Bhattacharya "Shooter location problem", in *Proc. 8th Canad. Conf. Comput. Geom.*, pp. 93–98, 1996.

[11] F. Nielsen "Fast stabbing of boxes in high dimensions", in *Proc. 8th Canad. Conf. Comput. Geom.*, pp. 87–92, 1996. To appear in *Theo. Comp. Sci.*.

[12] A. S. Rao and C. P. Rangan "Optimal parallel algorithms on circular-arc graphs", *Information Processing Letters* 33 (1989), pp. 147-156.

[13] R. E. Tarjan "Data Structures and Network Algorithms", *Regional Conference Series in Applied Mathematics* 44, SIAM, 1983.

[14] K. H. Tsai and D. T. Lee "k-best cuts for circular-arc graphs", *Algorithmica* 18(2) (1997), pp. 198–216.

[15] C. A. Wang and B. Zhu "Shooter location problems revisited", in *Proc. 9th Canad. Conf. Comput. Geom.*, pp. 223–228, 1997.