Two Approaches to Building Time-Windowed Geometric Data Structures*

Timothy M. Chan[1] and Simon Pratt[1]

1 **Cheriton School of Computer Science, University of Waterloo**
{tmchan,s2pratt}@uwaterloo.ca

Abstract

Given a set of geometric objects each associated with a time value, we wish to determine whether a given property is true for a subset of those objects whose time values fall within a query time window. We call such problems *time-windowed decision problems*, and they have been the subject of much recent attention, for instance studied by Bokal, Cabello, and Eppstein [SoCG 2015]. In this paper, we present new approaches to this class of problems that are conceptually simpler than Bokal *et al.*'s, and also lead to faster algorithms. For instance, we present algorithms for preprocessing for the time-windowed 2D diameter decision problem in $O(n \log n)$ time and the time-windowed 2D convex hull area decision problem in $O(n\alpha(n) \log n)$ time (where α is the inverse Ackermann function), improving Bokal *et al.*'s $O(n \log^2 n)$ and $O(n \log n \log\log n)$ solutions respectively.

Our first approach is to reduce time-windowed decision problems to a generalized range successor problem, which we solve using a novel way to search range trees. Our other approach is to use dynamic data structures directly, taking advantage of a new observation that the total number of combinatorial changes to a planar convex hull is near linear for any *FIFO* update sequence, in which deletions occur in the same order as insertions. We also apply these approaches to obtain the first $O(n \operatorname{polylog} n)$ algorithms for the time-windowed 3D diameter decision and 2D orthogonal segment intersection detection problems.

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

Keywords and phrases time window, geometric data structures, range searching, dynamic convex hull

Digital Object Identifier 10.4230/LIPIcs.SoCG.2016.28

1 Introduction

Time-windowed geometric problems have been the subject of many recent papers and are motivated by timestamped social network data and *Geographic Information System* (GIS) data, the latter of which may consist not only of longitude, latitude, and altitude coordinates but also time. A 2014 paper by Bannister *et al.* [4] examined time-windowed versions of convex hull, approximate spherical range searching, and approximate nearest neighbor queries. At SoCG 2015, Bokal *et al.* [5] presented more results on a variety of other time-windowed problems. In the same year, Chan and Pratt [12] studied the time-windowed closest pair problem.

* Research supported by The Natural Sciences and Engineering Research Council of Canada and the Ontario Graduate Scholarship.

32nd International Symposium on Computational Geometry (SoCG 2016).
Editors: Sándor Fekete and Anna Lubiw; Article No. 28; pp. 28:1–28:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Let S be a set of n objects, where each object $s \in S$ is associated with a time value $t(s)$. In this paper, we consider problems for which the answer is a Boolean value: given a query interval of time $[t_1, t_2]$ called a *time window*, does the subset of S whose time values are within the query window have property $\mathcal{P}$ or not? We call these *time-windowed decision problems*. For brevity, we say objects whose time values are within the query window are themselves within the query window.

Without loss of generality, we assume that time values are given as integers from 1 to n, for otherwise we can replace time values with their rank during preprocessing. The query time only increases by $O(1)$ predecessor searches on the query time values.

In this and the previous paper [5], we focus only on hereditary properties, meaning if a set S has $\mathcal{P}$ then any superset $S' \supseteq S$ also has it.[1] Examples of hereditary properties include: the set of points has greater than unit diameter, or the convex hull of a set of points has greater than unit area.

As observed by Bokal *et al.*, it suffices to find for each start time t the maximal end-time t' such that all objects within the window $[t, t']$ have $\mathcal{P}$. Afterwards, we can easily obtain a data structure with $O(n)$ words of space and $O(1)$ query time.[2] We do so by storing the resulting t' in a table indexed by start time t (recall that time values have been initially reduced to integers from 1 to n). A query for a time window $[t_1, t_2]$ is answered by looking up the t' in the table for start time t_1, and checking if $t_2 \leq t'$.

For this reason, Bokal *et al.* refer to time-windowed decision problems as the problem of finding maximal contiguous subsequences with hereditary properties.

Since answering a query after preprocessing is trivial, for the rest of the paper we focus only on bounding the preprocessing time.

1.1 Previous results

Recently, Bokal *et al.* [5] presented an approach to time-windowed decision problems. They achieve the following geometric results:

1. *2D diameter decision*: Given a set of n time-labeled points in $\mathbb{R}^2$, determine if there exist two points greater than unit distance apart, whose time values are within a query time window. Their approach obtains $O\big(n \log^2 n\big)$ preprocessing time.
2. *2D convex hull area decision*: Given a set of n time-labeled points in $\mathbb{R}^2$, determine whether the convex hull of points within a query time window has greater than unit area. Their approach obtains $O(n \log n \log \log n)$ preprocessing time.
3. *2D monotone paths*: Given a set of n points in $\mathbb{R}^2$, determine if the points within a query time window form a monotone path in some (subpath-dependent) direction. Their approach obtains $O(n)$ preprocessing time.

They also show that their approach works for graph planarity. Given a graph whose edge set contains n time-labeled edges, determine if the subgraph on the edges within a query time window is planar. Their approach obtains $O(n \log n)$ preprocessing time.

Chan and Pratt [12] studied the time-windowed closest pair decision problem, in which we are given a set of n time-labeled points in $\mathbb{R}^d$ and we wish to determine whether there exist two points at most unit distance apart. They solve the problem in $O(n)$ time using

[1] The definition in [5] considers subsets instead of supersets, but is equivalent after complementation.

[2] In fact, Chan and Pratt [12] show that we can reduce space to $O(n)$ bits while maintaining $O(1)$ query time, by using succinct rank/select data structures [23].

grids. They also consider the exact version of the problem, which is to find the closest pair of points within the time window. They solve this problem in $O(n \log n \log\log n)$ preprocessing time and $O(\log\log n)$ query time with $O(n \log n)$ words of space. Their techniques relied on geometric properties of the closest pair such that they could not be trivially modified to solve the time-windowed diameter problem.

1.2 New results

We achieve the following results:

1. *2D and 3D diameter decision*: We improve Bokal *et al.*'s preprocessing time bound in 2D from $O\big(n \log^2 n\big)$ to $O(n \log n)$. Thus, we obtain the first optimal algorithm for the problem in the algebraic decision-tree model [26]. Furthermore, we obtain the first nontrivial result in 3D with $O\big(n \log^2 n\big)$ preprocessing time. See Section 2 for details.
2. *2D orthogonal segment intersection detection*: Given a set of n orthogonal (horizontal or vertical) time-labeled line segments in $\mathbb{R}^2$, we want to determine if there are any intersections between segments whose time values are within a query time window. We give the first nontrivial result for this problem, obtaining $O(n \log n \log\log n)$ preprocessing time. See Section 2 for details.
3. *2D convex hull area decision*: We improve Bokal *et al.*'s preprocessing time bound from $O(n \log n \log\log n)$ to $O(n\alpha(n) \log n)$ (where α is the inverse Ackermann function). See Section 3 for details.
4. *2D width decision*: Given a set of n time-labeled points in $\mathbb{R}^2$, we want to determine whether the points within a query time window have greater than unit width. We give the first nontrivial result for this problem, obtaining $O\big(n \log^8 n\big)$ preprocessing time. See Section 3 for details. Previously, a naive approach using Chan's dynamic data structure [8] would give a worse $O\big(n^{3/2} \operatorname{polylog} n\big)$ time bound.

1.3 Techniques

Bokal *et al.*'s main approach computes the upper triangle of a binary $n \times n$ matrix where entry i, j has value 1 if and only if the set of objects within the window $[i, j]$ has $\mathcal{P}$. The first step is to greedily decompose the upper triangle into disjoint rectangles along the diagonal. The second step is to compute the values within each rectangle. First they compute the maximal end-time for the row which divides height of the rectangle in half. This splits the rectangle into 4 sub-rectangles, above and below the median row and left and right of its maximal end-time. The entries in the top-right have value 0, the entries in the bottom-left have value 1, and they recurse on top-left and bottom-right. Efficiency comes from the using a bounded-size *sketch* of uncomputed regions during recursion. A sketch is similar to a coreset in that it approximates a subset of objects, and its exact nature depends on the problem. Their results for both the 2D diameter and the 2D convex area decision problem are obtained via this approach.

Our first approach, which we discuss in Section 2, is much more direct: we simply reduce the problem to a *range successor search* problem. This (static) data structure problem can be solved by standard range searching techniques, and if we are not too concerned with extra logarithmic factors, we immediately obtain efficient solutions to the diameter decision problem in 2D *and* 3D, as well as orthogonal line segment intersection detection. To further improve the logarithmic factors, we come up with a more clever way to traverse a range tree.

Our second approach, which we discuss in Section 3, simply adds the objects in sequence to a dynamic data structure until $\mathcal{P}$ is true for the objects in the structure, then removes

from the beginning until $\mathcal{P}$ is not true, then repeats. Bokal *et al.* [5] have already suggested this naive use of dynamic data structures as a natural solution to the problem, but dismissed it as inefficient. We show that it is actually efficient in the case of the 2D convex hull area and width decision problems! We do so by a new bound on the combinatorial complexity of the structural changes to the convex hull under a certain sequence of updates in which the order of insertions is the same as the order of deletions. We call these *FIFO* update sequences. With this combinatorial bound (which may be of independent interest) it is straightforward to solve the time-windowed 2D width decision problem in $O(n \operatorname{polylog} n)$ time using Eppstein's dynamic width data structure [18], and with a more careful analysis, we can solve time-windowed 2D convex hull area decision problem in $O(n\alpha(n) \log n)$ time using hull trees [24].

2 Generalized range successor approach

In this section, we focus on time-windowed decision problems for properties that deal with pairs. More precisely, given a symmetric relation $\mathcal{R} = S \times S$, we consider the property $\mathcal{P}$ that there exist $p, q \in S$ such that $(p, q) \in \mathcal{R}$. We call such properties *pairwise interaction properties*. If $(p, q) \in \mathcal{R}$, we say that p *interacts with* q; we also say that p is *in* q*'s range*.

Examples of such problems include: diameter decision, for which two points interact if they are further apart than unit distance; segment intersection detection, in which two segments interact with each other if they intersect; and closest pair decision, in which two points interact if they are nearer than unit distance. Note that such properties are *hereditary*.

Our approach is to reduce the time-windowed problem to the following data structure problem:

▶ **Definition 1.** Let each object $s \in S$ have weight $w(s)$. In the *generalized range successor problem*, we want to preprocess S to find the *successor* of a query object q among the objects in q's range, that is, the object $p \in S$ that interacts with q with the smallest $w(p) > w(q)$.

The above is a generalization of the original 1D range successor problem where the objects are points in 1D and the objects' ranges are intervals; for example, see [30] for the latest results.

To see how the above data structure problem can be used to solve the time-windowed pairwise interaction problem, we simply find the successor p_q of every $q \in S$ in the generalized range successor problem with weights equal to time values.

▶ **Observation 2.** *A query window* $[t_1, t_2]$ *contains an interacting pair if and only if* $[t_1, t_2]$ *contains* $[t(q), t(p_q)]$ *for some* $q \in S$.

Proof. The "if" direction is trivial. For the "only if" direction, let p, q be an interacting pair in $[t_1, t_2]$ and assume that $t(q) \leq t(p)$ without loss of generality. Then $t(q) \leq t(p_q) \leq t(p)$ by definition of the successor p_q, and the claim follows. ◀

Thus, the answer to a query window $[t_1, t_2]$ is yes if and only if the point $(t_1, -t_2)$ is dominated by some point $(t(p), -t(p_q))$. The time-windowed problem can then be solved by precomputing the *maxima* [26] of the 2D point set $\{(t(p), -t(p_q)) \mid q \in S\}$, which takes linear time by a standard plane sweep after pre-sorting (recall that time values have been initially reduced to integers in $\{1, \ldots, n\}$ and can be trivially sorted in linear time). The running time is then dominated by the cost of computing the successors p_q for all $q \in S$. If we can solve the generalized range successor problem in $P(n)$ preprocessing time and $Q(n)$

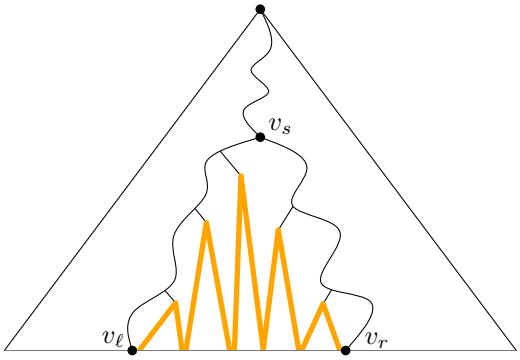

Figure 1 A range tree showing the path followed by a query for an interval $[\ell, r]$ which splits at v_s, and the subtrees in **bold** which fall within the query range between leaves v_ℓ and v_r.

query time, we can solve the corresponding time-windowed problem in $O(P(n) + nQ(n))$ preprocessing time.

In the rest of this section, we can thus focus on solving the generalized range successor problem.

One approach is to first consider the decision version of the problem: deciding whether there exists an object p that lies in q's range and has weight $w(p)$ in the interval $[\ell, r]$ for $\ell = w(q)$ and a given value r. This problem can be solved using standard multi-level data structuring techniques: The primary structure is a 1D range tree [26] on the weights (i.e., time values). See Figure 1. This naturally decomposes any interval $[\ell, r]$ of weights into $O(\log n)$ canonical subtrees by performing a binary search for both ℓ and r until we reach their lowest common ancestor node v_s at which the search splits. The search continues leftward and rightward to leaves v_ℓ and v_r with values ℓ and r respectively. Every right subtree on the path from v_s to v_ℓ, and every left subtree on the path from v_s to v_r are within the interval $[\ell, r]$. At each node v, we store the subset S_v of all objects within its interval in a *secondary structure* for the original range searching problem—deciding whether there exists an object in S_v that lies in a query object q's range. A query for the decision problem can then be answered by making $O(\log n)$ queries to the secondary structures. This increases the query time by a logarithmic factor.

Finally, we can reduce the generalized range successor problem to its decision problem by a binary search over all time values r. This increases the query time by a second logarithmic factor.

2.1 Avoiding binary search

In this subsection, we describe a still better algorithm that solves the generalized range successor problem without going through the decision problem, thereby removing one of the extra logarithmic factors caused by the binary search.

We first find the leaf node v storing q in $O(\log n)$ time. To answer a successor query for q, we proceed in two phases. (See Figure 2.)

- In the first (i.e., "up") phase, we walk upward from v towards the root. Each time our search follows a parent pointer from a left child, we query the secondary structure at the right child to see if there exists an object stored at the right child that is in q's range. If no, we continue upward. If yes, the answer is in the subtree at the right child and we proceed to the second phase starting at this node.

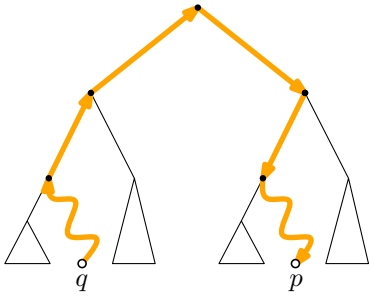

Figure 2 A search path finding the successor p of a point q is shown in **bold**. The search is divided into an "up" phase followed by a "down" phase.

- In the second (i.e., "down") phase, we walk downward from the current node to a leaf. Each time our search descends from a node, we query the secondary structure at the left child to see if there exists an object stored at the left child that is in q's range. If no, the answer is in the right subtree and we descend right. Otherwise, we descend left.

This algorithm makes $O(\log n)$ queries in the secondary structures. We next apply this algorithm to specific time-windowed pairwise interaction problems.

2.2 2D diameter decision

For the application to 2D diameter decision, our set of objects S is composed of points in $\mathbb{R}^2$, and p, q interact if and only if $d(p, q) > 1$. In other words, q's range is the complement of a unit disk.

The secondary structure at a node v needs to handle the following type of query: decide whether a query point q has greater than unit distance from some point in S_v, that is, decide whether q lies outside the intersection D_v of all unit disks centered at the points of S_v (see Figure 3). We store the unit-disk intersection D_v, along with the sorted list of the x-coordinates of the vertices of D_v.

Since we can merge two unit-disk intersections in linear time (similar to how we can merge two planar convex hulls in linear time), we can build the secondary structures at all nodes of the range tree bottom-up in $P(n) = O(n \log n)$ time.

Given point q, a query in the secondary structure at node v reduces to binary search for the x-coordinate of q in the list X_v and takes $O(\log n)$ time. Since the algorithm in Section 2.1 requires $O(\log n)$ queries in the secondary structures, the overall query time is $Q(n) = O\big(\log^2 n\big)$.

We can use *fractional cascading* to speed up the algorithm further [15, 16]. Since the technique is well known, we give just a quick sketch. Recall that the algorithm in Section 2.1 is divided into two phases.

- For the "up" phase, we first move the list X_v of each right child v to its parent. We pass a fraction of the elements of the list at each node to the lists at both children during preprocessing. This way, we can determine where the x-coordinate of q is in the list of the parent from where it is in the list of the child in $O(1)$ time. We can then answer all $O(\log n)$ queries in the secondary structures during the "up" phase in $O(\log n)$ overall time, after an initial binary search at the leaf in $O(\log n)$ time.
- For the "down" phase, we pass a fraction of the elements of the list at each node to the list at its parent during preprocessing. This way, we can determine where the x-coordinate

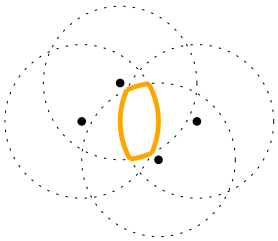

Figure 3 The boundaries of unit disks centered at four points in $\mathbb{R}^2$. The boundary of the unit-disk intersection is shown in **bold**.

of q is in the list of a child from where it is in the list of its parent in $O(1)$ time. We can then answer all $O(\log n)$ queries in the secondary structures during the "down" phase in $O(\log n)$ overall time, after an initial binary search in $O(\log n)$ time.

We conclude that a generalized range successor query in this setting can be answered in $Q(n) = O(\log n)$ time. This gives us the following result.

▶ **Theorem 3.** *We can preprocess for the time-windowed 2D diameter decision problem in $O(n \log n)$ time.*

2.3 3D diameter decision

In 3D, a query in the secondary structure at node v becomes deciding whether the query point q lies outside the intersection D_v of unit balls centered at the points of S_v. In 3D, the unit-ball intersection D_v still has linear combinatorial complexity and can be constructed in $O(|S_v| \log |S_v|)$ time by Clarkson and Shor's randomized algorithm [17] or Amato *et al.*'s deterministic algorithm [3]. We store the xy-projection of the upper and lower boundary of D_v in a planar point location structure [26]. We can build the secondary structures at all nodes of the range tree in $O(n \log n)$ time per level, and thus $P(n) = O\big(n \log^2 n\big)$ total time. (Unlike in 2D, it is not clear if we could speed up the building time by linear-time merging.)

Given point q, a query in a secondary structure reduces to planar point location for the xy-projection of q and takes $O(\log n)$ time [26]. Since the algorithm in Section 2.1 requires $O(\log n)$ queries in the secondary structures, the overall query time is $Q(n) = O(\log^2 n)$. (Unlike in 2D, we cannot apply fractional cascading to speed up the algorithm.) This gives us the following result.

▶ **Theorem 4.** *We can preprocess for the time-windowed 3D diameter decision problem in $O\big(n \log^2 n\big)$ time.*

2.4 Orthogonal segment intersection detection

For the application to 2D orthogonal segment intersection detection, our set of objects S is composed of vertical and horizontal line segments in $\mathbb{R}^2$, and p, q interact if and only if they intersect.

Without loss of generality, we assume that all x- and y-coordinates are given as integers from 1 to $O(n)$, for otherwise we can replace coordinate values with their rank during preprocessing. The query time only increases by $O(1)$ predecessor searches on the coordinate values, costing no more than $O(\log n)$ time.

The secondary structure at a node v now needs to handle the following type of query: decide whether a query segment q intersects some segment in S_v. Without loss of generality,

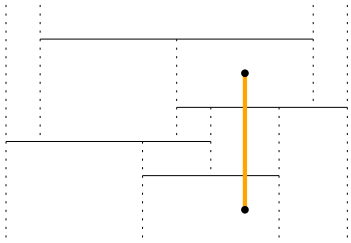

Figure 4 A set of horizontal segments is shown in solid lines, with their vertical decomposition shown in dotted lines. A query vertical segment is shown in **bold**, whose endpoints are in different cells.

assume that q is vertical and the segments in S_v are horizontal. We store the vertical decomposition VD_v (also called the trapezoidal decomposition) in a planar point location structure.

Since we can compute the vertical decomposition VD_v in $O(|S_v| \log\log |S_v|)$ time by a standard plane sweep with van Emde Boas trees, we can build the secondary structures at all nodes of the range tree in $P(n) = O(n \log n \log\log n)$ time.

Given vertical segment q, a query in the secondary structure at node v requires testing whether both endpoints of q lie in the same cell in VD_v, which reduces to two planar point location queries. Since the subdivision is orthogonal, we can apply Chan's orthogonal point location structure [10], which achieves $O(\log\log U)$ query time when coordinates are integers from $\{1, \ldots, U\}$—recall that coordinate values have been initially reduced to integers bounded by $U = O(n)$. Since the algorithm in Section 2.1 requires $O(\log n)$ queries in the secondary structures, the overall query time is $Q(n) = O(\log n \log\log n)$. This gives us the following result.

▶ **Theorem 5.** *We can preprocess for the time-windowed 2D orthogonal intersection detection problem in $O(n \log n \log\log n)$ time.*

▶ Remark. An open problem is to remove the extra $\log\log n$ factor. Perhaps the techniques from [11] for the 4D offline dominance searching problem may be relevant.

3 FIFO update sequence approach

In the previous section, we have presented an approach to building data structures to solve time-windowed pairwise interaction problems, but the 2D width and the 2D convex hull area decision problems, for instance, cannot be expressed in terms of a pairwise interaction property.

As mentioned in the introduction, both problems are on hereditary properties. The most obvious approach to solve a problem on a hereditary property is to use a dynamic data structure directly, inserting each object in order until $\mathcal{P}$ is satisfied, then deleting each object in the same order until $\mathcal{P}$ is no longer satisfied, and repeating. By storing for each $i \in \{1, \ldots n\}$ the largest j for which $\{s_i, \ldots, s_j\}$ satisfies $\mathcal{P}$, we can answer queries for the time-windowed problem. However, this approach does not seem to yield efficient solutions in some settings. For example, for the 2D width decision problem, we would need a fully dynamic data structure for 2D width decision, but the best result to date has near $\sqrt{n}$ update time [8]. Agarwal and Sharir [2] gave a dynamic data structure for 2D width decision with polylogarithmic update time but only for *offline* update sequences; their data structure

does not seem to work in our application when we do not know a priori in what order the deletions are intermixed with the insertions.

Both the 2D width and 2D convex hull area problem are about the convex hull. There exist sequences of n updates to the convex hull in the plane which cause $O(n^2)$ many structural changes. For example, consider the case of inserting a point which causes $O(n)$ points to be no longer on the convex hull, then deleting and re-inserting the same point n times. However, in our application points are deleted in the same order as they are inserted, so this particular example cannot occur.

Restricted update sequences on the dynamic convex hull have been studied before. The insertion-only case was studied by Preparata [25]. The deletion-only case was studied by Chazelle [14], and Hershberger and Suri [20]. Random update sequences were studied by Mulmuley [22] and Schwarzkopf [27].

We call an update sequence in which objects are inserted and deleted in the same order a *first-in-first-out (FIFO) update sequence*, which to the best of our knowledge has not been studied before. We prove a combinatorial lemma, stating that for such sequences the number of structural changes to the convex hull is always near linear.

▶ **Lemma 6.** *The number of structural changes to the upper hull of a set of points in $\mathbb{R}^2$ over n FIFO updates is at most $O(n \log n)$.*

The proof of this lemma uses an old observation by Tamir [29] for arbitrary update sequences: although the number of edge creations or destructions in the convex hull could be quadratic, it turns out that the number of distinct edges created or destroyed is near linear.

▶ **Observation 7.** *There are $O(n \log n)$ distinct edges created or destroyed in the upper hull of a set of points in $\mathbb{R}^2$ over an arbitrary sequence of n insertions or deletions.*

Proof. Consider the vertical line at the median x-coordinate. At any time, there is just one hull edge that crosses the vertical line (called the *bridge*), and thus the number of possible bridges over time is $O(n)$. Thus, the number of distinct edges that appear on the upper hull over time satisfies the recurrence

$$E(n) \;=\; 2E(n/2) + O(n)\,,$$

implying that $E(n) = O(n \log n)$. ◀

Lemma 6 now follows immediately by combining the above observation with another observation about FIFO update sequences:

▶ **Observation 8.** *For a FIFO update sequence, once an edge uv has been removed from the upper hull by the insertion of a point w, uv can never again be an edge of the upper hull.*

Proof. Since w is above the line through uv, we know that uv cannot be an edge of the upper hull while w is alive. But since w was inserted after u and v, by the FIFO property w must be deleted after u and v, and therefore uv can never again be an upper hull edge. ◀

3.1 2D width decision

We can immediately apply Lemma 6 to solve the time-windowed 2D width decision problem, by using Eppstein's dynamic 2D width data structure [18] as a black box. Eppstein's algorithm maintains the width in time $O(k \cdot f(n) \cdot \log n)$ where k is the number of structural changes to the convex hull, and $f(n)$ is the time to solve the dynamic 3D convex hull problem

(more precisely, answer gift-wrapping queries and perform updates for a 3D point set). Note that Eppstein used Agarwal and Matoušek as a black box to solve the dynamic 3D convex hull problem in $O(n^\epsilon)$ time [1], but this has since been improved by Chan to $O\big(\log^6 n\big)$ expected time [9], and derandomized by Chan and Tsakalidis [13]. This proves the following result.

▶ **Theorem 9.** *We can preprocess for the time-windowed 2D width decision problem in $O\big(n \log^8 n\big)$ time.*

3.2 2D convex hull area decision

For the time-windowed 2D convex hull area decision problem, we can now directly apply known fully dynamic convex hull data structures [24, 7, 6], most of which can be modified to maintain the area. For example, Brodal and Jacob's (extremely complicated) data structure [6] can maintain the convex hull and its area in $O(k \cdot \log n)$ amortized time, where k is the number of structural changes to the convex hull. This would imply an $O\big(n \log^2 n\big)$-time algorithm, which is worse than Bokal *et al.*'s result [5].

We show how to reduce this to $O(n\alpha(n) \log n)$ by directly adapting a simpler known dynamic convex hull data structure, namely Overmars and van Leeuwen's *hull tree* [24], and carefully analyzing it for FIFO sequences.

▶ **Lemma 10.** *We can maintain the convex hull and its area for a 2D set of n points under FIFO updates in $O(n\alpha(n) \log n)$ total time.*

Proof. It suffices to maintain the upper hull and the area above it inside a sufficiently large bounding box, since we can similarly maintain the lower hull and the area below it, and subtract the areas from the bounding box.

A *hull tree* is a binary tree whose root node stores the upper hull edge (called the *bridge*) that crosses the median vertical line, and whose left and right subtrees are the hull trees on all points left and right of the median, respectively. Here, we assume that the x-coordinates of all n points are known in advance, which is true in our application (the assumption can be removed by extra steps to balance the hull tree, for example, via tree rotations [24]). At each node, we store the area above the upper hull. Pointer structures can be set up to let us traverse the upper hull at any node of the tree [24, 21].

The following definitions will be helpful: Consider the upper hull at a tree node. When we insert a point u which causes a polygonal chain $v_1 v_2 \cdots v_k$ to disappear from the upper hull, we say that u *kills* v_i, and that (u, v_i) forms a *killing pair*, for each $i = 1, 2, \ldots, k$. (For technical reasons, we allow $i = 1$ and $i = k$ in the definition, counterintuitively.) Symmetrically, if we delete a point u which causes a polygonal chain $v_1 v_2 \cdots v_k$ to appear in the upper hull, we say that u *revives* v_i, and that (u, v_i) forms a *revival pair*, for each $i = 1, 2, \ldots, k$.

Deletion. Consider the deletion of a point p at a node of the hull tree. Without loss of generality, suppose that p is to the right of the median. We first recursively delete p in the right subtree. Suppose that p was an endpoint of the bridge of the node. We need to compute a new bridge. Overmars and van Leeuwen [24] originally proposed a binary search, but we will use a linear search instead (inspired by the variants of hull trees by Chazelle [14] and Hershberger and Suri [21] for deletion-only sequences). Specifically, let ℓ_0 be the left endpoint of the old bridge, and let r_0 and r_1 be the predecessor and successor of p in the old right upper hull respectively. (See Figure 5.) A simple rightward linear search from ℓ_0 and r_0

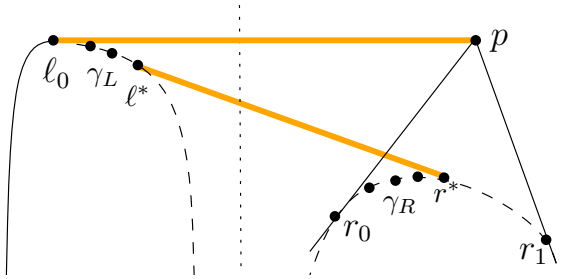

Figure 5 Deletion of p causes the old bridge (ℓ_0, p) to change to the new bridge (ℓ^*, r^*), both shown in **bold**. Computing the new bridge requires walking from ℓ_0 to ℓ^* and r_0 to r^*. If point p is instead being inserted, computing the new bridge requires walking from ℓ^* to ℓ_0. Any pair (p, ℓ) with $\ell \in \gamma_L$ or pair (p, r) with $r \in \gamma_R$ is a revival or killing pair at some node of the hull tree.

can find the new bridge (ℓ^*, r^*) in $O(|\gamma_L| + |\gamma_R|)$ time, where γ_L denotes the subchain from ℓ_0 to ℓ^* in the left upper hull, and γ_R denotes the subchain from r_0 to r^* in the right upper hull. (Note that ℓ^* must be right of ℓ_0, and r^* must be right of r_0.) The change in area at the current node can be computed in $O(|\gamma_L| + |\gamma_R|)$ time (it is the area of the polygon with vertices $\ell_0\gamma_L\ell^* r^* p$, plus the change in area at the right child, minus the area of the polygon with vertices $r_0\gamma_R r^* p$).

To account for the $O(|\gamma_L|)$ cost, observe that for each $\ell \in \gamma_L$, (p, ℓ) is a revival pair for the upper hull at the current node. We charge one unit to each such pair (p, ℓ). Note that each pair is charged at most once during the entire algorithm.

To account for the $O(|\gamma_R|)$ cost, observe that for each $r \in \gamma_R$, (p, r) is a revival pair for the upper hull at the right child. We charge one unit to each such pair (p, r). If (p, r) is charged, then r lies strictly below the upper hull at the current node and cannot be charged again at an ancestor. Thus, each pair is charged at most once this way.

Insertion. Consider the insertion of a point p at a node of the hull tree. Without loss of generality, suppose that p is to the right of the median. We first recursively insert p in the right subtree. We need to compute the new bridge (if it changes). We can just mimick the deletion algorithm in reverse. In fact, the details are a little simpler: a linear search from ℓ^* can find ℓ_0, the left endpoint of the new bridge (see Figure 5) in $O(|\gamma_L|)$ time. The change in the area can again be computed in $O(|\gamma_L| + |\gamma_R|)$ time. We can account for the cost again by charging, this time, to killing instead of revival pairs.

Total time. The total cost over all updates is proportional to the number of charges, which is bounded by $K(n)$, the worst-case number of distinct pairs (u, v) such that (u, v) is a killing/revival pair for the upper hull of at least one node of the hull tree, over all sets of n points. In the next subsection, we prove that $K(n) = O(n\alpha(n)\log n)$ (Lemma 13), which would then imply an $O(n\alpha(n)\log n)$ time bound. ◀

▶ **Theorem 11.** *We can preprocess for the time-windowed 2D convex hull area decision problem in $O(n\alpha(n)\log n)$ time.*

3.3 Bounding the number of killing/revival pairs

One ingredient is still missing: a proof that $K(n) = O(n\alpha(n)\log n)$. Naively we could bound the number of killing/revival pairs by the number of structural changes to the upper hull at

each node, and applying Lemma 6 would give us the recurrence $K(n) = 2K(n/2)+O(n \log n)$, implying a weaker bound $K(n) = O\big(n \log^2 n\big)$.

We propose a different combinatorial argument to bound $K(n)$. Our approach contains a nice application of *Davenport-Schinzel (DS) sequences* [28]. Recall that a DS sequence *of order 3* is a sequence Σ of characters $s_1, s_2, \ldots$ from an alphabet A such that no two consecutive characters are the same and for any two characters $a, b \in A$, the alternating sequence a, b, a, b, a of length 5 does not appear as a subsequence anywhere in Σ, whether contiguous or not. Hart and Sharir [19, 28] proved that an order-3 DS sequence over an alphabet of size n has length at most $O(n\alpha(n))$.

DS sequences occur often in computational geometry, such as in bounding the combinatorial complexity of the lower envelope of line segments. Surprisingly in our case, we do not relate our problem to lower envelopes or other substructures in arrangements. Rather, we relate directly to DS sequences.

It suffices to bound the number of killing pairs, since revival pairs are symmetric, by reversing time. To this end, we first concentrate on a special kind of killing pairs: for the upper hull at a fixed node of the hull tree, a *bridge killing pair* is a killing pair (u, v) where u and v lie on opposite sides of the median vertical line.

▶ **Lemma 12.** *For the upper hull at a fixed node, the number of bridge killing pairs is at most $O(n\alpha(n))$ for any FIFO update sequence.*

Proof. By symmetry, it suffices to count killing pairs (u, v) where u is to the right and v is to the left of the median vertical line. Take the sequence of all such killing pairs $(u_1, v_1), \ldots, (u_n, v_m)$ ordered by time, where in case of ties, simultaneous killings are ordered in decreasing x-order of v_i. Define the sequence Σ to be $v_1, \ldots, v_m$. We claim that Σ cannot have any alternating subsequence of length 5.

Assume that an alternating subsequence of killed points $\ldots, a, \ldots, b, \ldots, a, \ldots, b, \ldots, a, \ldots$ or $\ldots, b, \ldots, a, \ldots, b, \ldots, a, \ldots, b, \ldots$ of length 5 occurs in Σ. Without loss of generality, assume that a is to the left of b. In either case, the subsequence $\ldots, b, \ldots, a, \ldots, b, \ldots, a, \ldots$ of length 4 occurs in Σ.

Consider the time in this length-4 subsequence when a is killed ($b\underline{a}ba$) and let the vertex which kills it be u (see Figure 6). At this time, b must exist, because it will be killed later; furthermore, b is on or below the current upper hull, and so lies below the line segment $\overline{au}$. Now, fast forward to the time in the subsequence when b is next killed ($ba\underline{b}a$). Note that this must be at a different time, because if u kills a and b at the same time, b would be placed before a in Σ by our tie-breaking rule. At this new time, a and u must both exist, because a will be killed later again, and u was inserted after b and will be deleted after b by definition of FIFO sequences. But b is below $\overline{au}$ and cannot appear on the upper hull and cannot be killed at this time: a contradiction. This completes the proof of the claim.

The sequence Σ may still have identical consecutive characters, but a repeated pair can occur only "between" two different insertion events and there are at most n insertion events. After removal of $O(n)$ repeated characters, Σ thus becomes a DS sequence of order 3 and by the known upper bound has length at most $O(n\alpha(n))$ [28]. The lemma follows. ◀

▶ **Lemma 13.** *The number of distinct killing/revival pairs over all nodes in the hull tree satisfies $K(n) = O(n\alpha(n) \log n)$ for any FIFO update sequence.*

Proof. By Lemma 12, there are $O(n\alpha(n))$ bridge killing pairs for the upper hull at the root node. The remaining killing pairs are killing pairs at nodes of the left subtree and nodes of

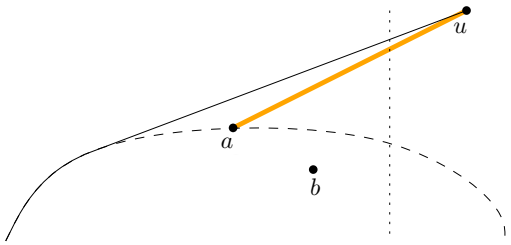

Figure 6 u kills a. The dotted line is the median vertical line and the dashed line was part of the convex hull before u was inserted. The point b must be below the **bold** line segment $\overline{au}$.

the right subtree. We thus obtain the recurrence

$$K(n) \;=\; 2K(n/2) + O(n\alpha(n))\,,$$

implying that $K(n) = O(n\alpha(n)\log n)$. ◀

▶ Remark. We leave open the possibility of removing the tiny $\alpha(n)$ factor in the combinatorial bound on $K(n)$. Alternatively, there is the possibility of reducing $\alpha(n)$ to $\log(\alpha(n))$ in the running time of Lemma 10 and Theorem 11, by replacing linear search with fingered binary search.

Another interesting question is whether Lemma 6's $O(n \log n)$ bound on the number of structural changes to the 2D convex hull for FIFO update sequences can be improved. We are not aware of any superlinear lower bound. (Concerning Lemma 7 for arbitrary update sequences, there is an $\Omega(n\alpha(n))$ lower bound [29].)

Acknowledgements We wish to thank Haim Kaplan and Micha Sharir for suggesting the use of Tamir's observation [29], which leads to an $\alpha(n)$ factor improvement to our earlier version of Lemma 6.

References

1 Pankaj K. Agarwal and Jirı Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, 1995.
2 Pankaj K. Agarwal and Micha Sharir. Off-line dynamic maintenance of the width of a planar point set. *Computational Geometry: Theory and Applications*, 1:65–78, 1991.
3 Nancy M. Amato, Michael T. Goodrich, and Edgar A. Ramos. Parallel algorithms for higher-dimensional convex hulls. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 683–694, 1994.
4 Michael J. Bannister, William E. Devanny, Michael T. Goodrich, Joseph A. Simons, and Lowell Trott. Windows into geometric events: Data structures for time-windowed querying of temporal point sets. In *Proceedings of the 26th Canadian Conference on Computational Geometry (CCCG)*, 2014.
5 Drago Bokal, Sergio Cabello, and David Eppstein. Finding all maximal subsequences with hereditary properties. In *Proceedings of the 31st International Symposium on Computational Geometry (SoCG)*, pages 240–254, 2015.
6 Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 617–626, 2002.
7 Timothy M. Chan. Dynamic planar convex hull operations in near-logarithmaic amortized time. *Journal of the ACM*, 48(1):1–12, 2001.

8 Timothy M. Chan. A fully dynamic algorithm for planar width. In *Proceedings of the 17th Annual Symposium on Computational Geometry (SoCG)*, pages 172–176, 2001.

9 Timothy M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. *J. ACM*, 57(3):16:1–16:15, 2010.

10 Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word ram. *ACM Transactions on Algorithms*, 9(3):22, 2013.

11 Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of the 27th Annual Symposium on Computational Geometry (SoCG)*, pages 1–10, 2011.

12 Timothy M. Chan and Simon Pratt. Time-windowed closest pair. In *Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG)*, 2015.

13 Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. In *Proceedings of the 31st International Symposium on Computational Geometry (SoCG)*, pages 719–732, 2015.

14 Bernard Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31(4):509–517, 1985.

15 Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(1-4):133–162, 1986.

16 Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1(1-4):163–191, 1986.

17 Kenneth L. Clarkson and Peter W. Shor. Application of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989.

18 David Eppstein. Incremental and decremental maintenance of planar width. *Journal of Algorithms*, 37(2):570–577, 2000.

19 Sergiu Hart and Micha Sharir. Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes. *Combinatorica*, 6(2):151–178, 1986.

20 John Hershberger and Subhash Suri. Offline maintenance of planar configurations. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 32–41, 1991.

21 John Hershberger and Subhash Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32(2):249–267, 1992.

22 Ketan Mulmuley. Randomized multidimensional search trees: Lazy balancing and dynamic shuffling. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 180–196, 1991.

23 J. Ian Munro. Tables. In *Proceedings of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 37–42. Springer, 1996.

24 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23(2):166–204, 1981.

25 Franco P. Preparata. An optimal real-time algorithm for planar convex hulls. *Communications of the ACM*, 22(7):402–405, 1979.

26 Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, USA, 1985.

27 Otfried Schwarzkopf. Dynamic maintenance of geometric structures made easy. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 197–206, 1991.

28 Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, 1995.

29 Arie Tamir. Improved complexity bounds for center location problems on networks by using dynamic data structures. *SIAM Journal on Discrete Mathematics*, 1(3):377–396, 1988.

30 Gelin Zhou. Two-dimensional range successor in optimal time and almost linear space. *Information Processing Letters*, 2015.