Two Approaches to Building Time-Windowed Geometric Data Structures[*]

Timothy M. Chan[†] John Hershberger[‡] Simon Pratt[§]

May 9, 2019

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 both the time-windowed 2D diameter decision problem and the time-windowed 2D convex hull area decision problem in $O(n \log n)$ time, 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 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.

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]

[*]A preliminary version of this work appeared in *Proceedings of the 32nd International Symposium on Computational Geometry*, 2016 [14]. Much of this work was done while the first and third authors were at the Cheriton School of Computer Science, University of Waterloo, Canada. The work was part of the third author's Master's thesis at Waterloo [27]. This research was supported by The Natural Sciences and Engineering Research Council of Canada and an Ontario Graduate Scholarship.

[†]Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA. E-mail: `tmc@illinois.edu`

[‡]Mentor Graphics Corp., Wilsonville, OR, USA. E-mail: `john_hershberger@mentor.com`

[§]E-mail: `me@SimonDavidPratt.com`

presented more results on a variety of other time-windowed problems. In the same year, Chan and Pratt [13] studied the time-windowed closest pair problem.

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 minimal end-time t' such that 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 $t' \le t_2$.

For this reason, Bokal *et al.* refer to time-windowed decision problems as the problem of finding minimal 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 $O(n \log^2 n)$ 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.

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

[2] In fact, Chan and Pratt [13] show that we can reduce space to $O(n)$ bits while maintaining $O(1)$ query time, by using succinct rank/select data structures [25].

Chan and Pratt [13] study 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 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 rely on geometric properties of the closest pair and cannot 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 [29]. 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 horizontal or vertical time-labeled line segments in $\mathbb{R}^2$, we want to determine if there are any intersections between horizontal segments with vertical 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 $O(n \log n)$. 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 $O\big(n \log^5 n\big)$ preprocessing time. See Section 3 for details. Previously, a naïve 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 considers 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 goal is to compute, for each row, the leftmost column at which the matrix entry is 1. First, Bokal *et al.* decompose the upper triangle into disjoint rectangles, each with a corner on the main diagonal $i = j$, which together cover all the entries we want; this decomposition is generated by a simple greedy algorithm. Then, for each such rectangle, the problem is solved by divide-and-conquer: At the median row, we find the leftmost column at which the matrix entry is 1; this splits the rectangle into 4 subrectangles. The entries in the top-right subrectangle all have value 1, and the entries in the bottom-left subrectangle all have value 0. We recursively solve the subproblem for the top-left and bottom-right subrectangles. Unfortunately, the subproblem size (the number of objects relevant to the subproblem) does not decrease, but Bokal *et al.* showed that for certain problems such as 2D diameter and 2D convex area decision, the input to the subproblem can be replaced by a smaller-size *sketch* (similar to "coresets" in the context of approximation algorithms). The efficiency of the resulting divide-and-conquer algorithm relies on such sketches.

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 efficient 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 naïve 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, because of a special property concerning the update sequences that arise in our applications. Specifically, we prove a linear upper bound on the number of 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. Although there has been at least one prior paper on geometric data structures for FIFO updates [31], our combinatorial bound for planar convex hull appears new, and may be of independent interest. From this combinatorial result, we can immediately solve the time-windowed 2D width decision problem in $O(n \operatorname{polylog} n)$ preprocessing time via Eppstein's dynamic width data structure [20], and we can solve time-windowed 2D convex hull area decision problem in $O(n \log n)$ preprocessing time via a known dynamic convex hull data structure [6, 26].

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 $\mathcal{R} \subseteq S \times S$, 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 farther 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 [32] for the latest results.

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

Observation 1. *A query window $[t_1, t_2]$ contains an interacting pair if and only if $[t_1, t_2]$ contains $[t(q), t(q^+)]$ for some $q \in S$.*

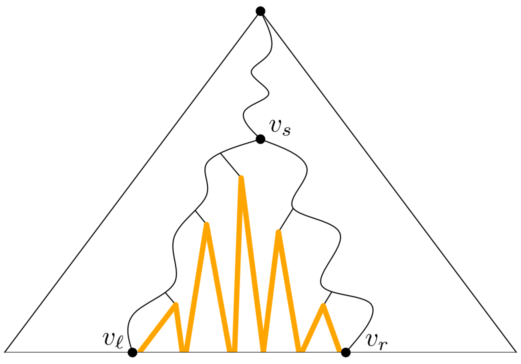

Figure 1: A range tree showing the path followed by a query for an interval $[\ell, r]$ that splits at v_s, and the subtrees in **bold** that fall within the query range between leaves v_ℓ and v_r.

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 $t(q) \le t(p)$ without loss of generality. Then $t(q) \le t(q^+) \le t(p)$ by definition of the successor q^+, and the claim follows. $\square$

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 $(t(p), -t(q^+))$. The time-windowed problem can then be solved by precomputing the *maxima* [29] of the 2D point set $\{(t(p), -t(q^+)) \mid q \in S\}$, 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 q^+ for all $q \in S$. If we can solve the generalized range successor problem in $P(n)$ preprocessing time and $Q(n)$ 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 [29] 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.

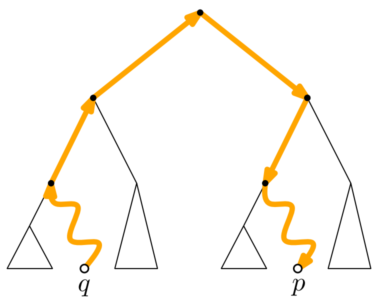

Figure 2: A search path finding the successor p of a point q is shown in **bold**. The search is divided into an "up" phase followed by a "down" phase.

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.
- 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.

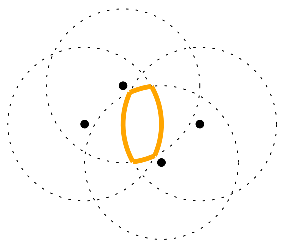

Figure 3: The boundaries of unit disks centered at four points in $\mathbb{R}^2$. The boundary of the unit-disk intersection is shown in **bold**.

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 $Q(n) = O(\log^2 n)$.

We can use *fractional cascading* to speed up the algorithm further [17, 18]. 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 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 2. *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 [19] 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 [29]. We can build the secondary structures at all nodes of the range tree in $O(n \log n)$ time per level, and thus $P(n) = O(n \log^2 n)$ total time. (Unlike in 2D, it is not clear if we could speed up the building time by linear-time merging.)

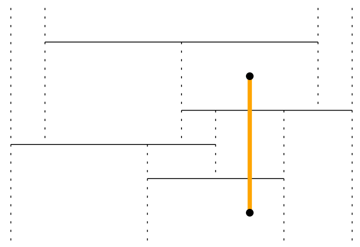

Figure 4: A set of horizontal segments is shown in solid lines, with their vertical decomposition shown in dotted lines. A query vertical segment is shown in **bold**; its endpoints are in different cells.

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 [29]. 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 3. *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 horizontal and vertical line segments in $\mathbb{R}^2$, and for a horizontal segment p and a vertical segment q (or vice versa), p and 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, 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 (see Figure 4), 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 4. *We can preprocess for the time-windowed 2D orthogonal intersection detection problem in $O(n \log n \log\log n)$ time.*

Remark 1. An open problem is to remove the extra $\log\log n$ factor. Perhaps the techniques from [12] 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 smallest 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 that cause $\Omega(n^2)$ structural changes. For example, consider the case of inserting a point that causes $\Omega(n)$ 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 [28]. The deletion-only case was studied by Chazelle [16], and Hershberger and Suri [21]. Random update sequences were studied by Mulmuley [24] and Schwarzkopf [30].

We call an update sequence in which objects are inserted and deleted in the same order a *first-in-first-out (FIFO) update sequence.* Sheng and Tao [31] examined data structures for extreme point queries in the plane under FIFO updates, but did not consider the explicit maintenance of the convex hull. We prove a combinatorial lemma, stating that, for such sequences, the number of structural changes to the convex hull is at most linear.[3]

Lemma 1. *The number of structural changes to the upper hull of a set of points in $\mathbb{R}^2$ over n FIFO updates is $O(n)$.*

Proof. We want to bound the number of edge insertions/deletions on the upper convex hull of a dynamic set P of points subject to FIFO insertions and deletions. We order the $n = |P|$ points of P by their insertion times: $P = \{p_1, p_2, \ldots, p_n\}$. Each point $p \in P$ is inserted at time $i(p)$ and deleted at time $d(p)$, with $i(p) < d(p)$. Let us assume that all insertion and deletion times are distinct, so $j < k$ iff $i(p_j) < i(p_k)$ iff $d(p_j) < d(p_k)$.

We can create a *lifetime plot* for the points of P by plotting the horizontal line segment $[(i(p_i), i), (d(p_i), i)]$ for each point $p_i \in P$. These segments lie in the band between the two monotone paths formed by the sequences $\{(i(p_i), i)\}$ and $\{(d(p_i), i)\}$.

We divide the time interval $[i(p_1), d(p_n)]$ into subintervals in which no point is both inserted and deleted. That is, in each subinterval the active points of P consist of one set subject only to insertions and one set subject only to deletions. We can visualize this graphically by drawing a

[3]This improves the $O(n \log n)$ upper bound from the earlier conference version of this paper [14].

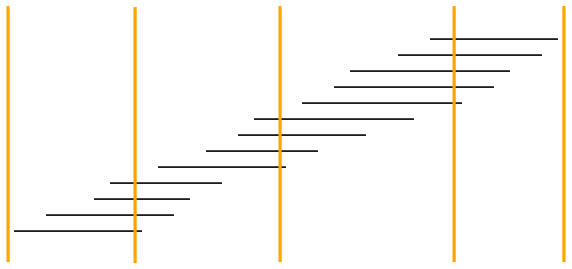

Figure 5: Lifetime plot for a FIFO update sequence, and a division into time intervals such that in each interval, no point is both inserted and deleted.

monotone staircase inside the band: we draw the vertical segment from $(d(p_1), 1)$ up to the top of the band, draw the horizontal segment from there to the right side of the band, and repeat, zigzagging up inside the band. The vertical segments of the staircase divide time into the desired intervals. Phrased differently, we can define a sequence of time values $t_0, t_1, \dots, t_k$ by setting t_0 to a value less than $i(p_1)$, then inductively setting t_i to a value just less than $d(p_X)$, where p_X is the first point inserted after t_{i-1}. At the penultimate time value t_{k-1}, there are no points left to insert, and the final time value t_k is chosen to be greater than $d(p_n)$. (See Figure 5 for an example.) By construction, the lifetime of each point in P overlaps the interiors of exactly two of the time intervals (t_i, t_{i+1}).

We can count the number of combinatorial changes to $H(P)$, the upper convex hull of P, by summing the number of changes over all intervals. In each interval, we have one set of points D subject to deletions only and one set I subject to insertions only. The sum of $|D| + |I|$ over all intervals is $2n$.

The edges that appear on the convex hull during a single interval are of three types: edges joining points of D, edges joining points of I, and *mixed* edges joining a point of D to a point of I.

For any set of points S, an edge (u, v) of $H(S)$ is also an edge of $H(S')$ for any $S' \subseteq S$ that includes both u and v. Thus the I-only edges of $H(D \cup I)$ are a subset of the edges of $H(I)$. The number of convex hull edges of a dynamic insertions-only set I is at most $2|I|$, because each inserted vertex creates at most two edges (incident to it), however many edges it hides (removes from the hull). Likewise, the D-only edges of $H(D \cup I)$ are a subset of the edges of $H(D)$, and their number is at most $2|D|$, as we can see by running time backward and applying the insertions-only bound.

Now we bound the mixed edges, i.e., those with one endpoint from each of D and I. Since no point of I is present at the beginning of the time interval, and no point of D is present at the end of the interval, all mixed edges are created and destroyed during the interval. We bound the number of mixed edges by charging each edge's deletion to a point of D or I.

Consider a mixed edge $e = (u, v)$, with $u \in D$ and $v \in I$. The deletion of e may be due to a deletion from D or an insertion into I. If the deletion of e is due to a point deletion, it can only be due to the deletion of u, by the subset-hull observation above. In this case we charge edge e to vertex $u \in D$. If the deletion of e is due to the insertion of a point $x \in I$, we charge the deletion either to u or to x. When x is inserted, it hides (causes the removal of) one or more mixed edges. Let $e_1, e_2, \dots, e_m$ be the mixed edges removed by x, numbered in left-to-right order. If $e \in \{e_1, e_m\}$, then we charge e to x; at most two edges are charged to x in this way. Otherwise, we charge e to u. The upper convex hull of x and the two I-vertices in e_1 and e_m completely hides the edges $e_2, \dots, e_{m-1}$, ensuring that u is never on the convex hull again. Thus a point u of D is charged for a mixed edge only if u is leaving the convex hull for the last time. Since each point of D is charged at most twice (once from each side) and each point of I is also charged at most twice, the total number of mixed edges is at most $2(|D| + |I|)$.

Combining the bounds for D-only, I-only, and mixed edges, and summing over all time intervals, it follows that the total number of edge insertions and deletions on the convex hull of a point set subject to FIFO insertions and deletions is $O(n)$. □

3.1 2D width decision

We can immediately apply Lemma 1 to solve the time-windowed 2D width decision problem, by using Eppstein's dynamic 2D width data structure [20] 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 $O(n^{\varepsilon})$ time [1], but this was improved by Chan to $O(\log^6 n)$ expected time [9], derandomized by Chan and Tsakalidis [15], further improved by Kaplan *et al.* to $O(\log^5 n)$ amortized time [23] and by Chan to $O(\log^4 n)$ [11]. This proves the following result.

Theorem 5. *We can preprocess for the time-windowed 2D width decision problem in $O(n \log^5 n)$ 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 [6, 7, 26], most of which can be modified to maintain the area. For example, Brodal and Jacob's 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 yields an overall running time of $O(n \log n)$.

Brodal and Jacob's data structure is very complicated. In the proof below, we suggest a simpler alternative solution for FIFO updates by instead adapting Overmars and van Leeuwen's dynamic convex hull data structure, namely, the *hull tree* [26]:

Theorem 6. *We can preprocess for the time-windowed 2D convex hull area decision problem in $O(n \log n)$ 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* of a given point set is a binary tree whose root node stores the upper hull edge (called the *bridge*) that crosses the median vertical line, and whose left/right subtrees are the hull trees for the subset of all points to the left/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 [26]). At each node, we store the area above the upper hull of its corresponding subset of points. Pointer structures can be set up to let us traverse the upper hull at any node of the tree [22, 26].

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

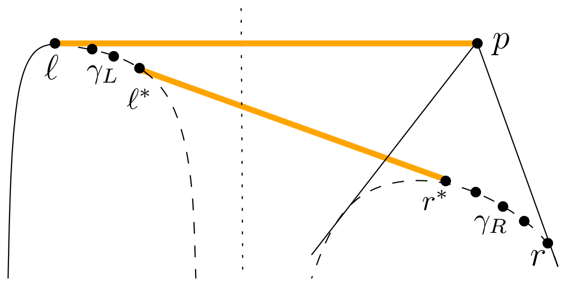

Figure 6: Deletion of p causes the old bridge (ℓ, p) to change to the new bridge (ℓ^*, r^*), both shown in **bold**. Computing the new bridge requires walking from ℓ to ℓ^* and r to r^*. If point p is instead being inserted, computing the new bridge requires walking from ℓ^* to ℓ.

p was an endpoint of the bridge of the node. We need to compute a new bridge. Overmars and van Leeuwen [26] originally proposed a binary search, but we use a linear search instead (inspired by the variants of hull trees by Chazelle [16] and Hershberger and Suri [22] for deletion-only sequences). Specifically, let ℓ be the left endpoint of the old bridge, and let r be the successor of p in the old upper hull. (See Figure 6.) A simple linear scan walking rightward from ℓ and leftward from r can find the new bridge (ℓ^*, r^*) in $O(|\gamma_L| + |\gamma_R|)$ time, where γ_L denotes the subchain from ℓ to ℓ^* in the left upper hull, and γ_R denotes the subchain from r^* to r in the new right upper hull. The change in area at the current node (i.e., the area of the polygon with vertices $\ell\gamma_L\ell^*r^*\gamma_R rp\ell$) can be computed in the same amount of time. The $O(|\gamma_L| + |\gamma_R|)$ cost is bounded by the number of structural changes to the upper hull at the node.

Next 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 mimic the deletion algorithm in reverse. In fact, the details are a little simpler: a linear search from ℓ^* can find ℓ, the left endpoint of the new bridge (see Figure 6), in $O(|\gamma_L|)$ time. The change in the area can again be computed in $O(|\gamma_L| + |\gamma_R|)$ time, which is bounded by the number of structural changes to the upper hull at the node.

Let $T(n)$ denote the total time for performing an entire FIFO update sequence by the above method, over a hull tree of n points. By Lemma 1, the total cost is $O(n)$ plus the cost of maintaining the left and the right hull subtrees. Thus, $T(n) = 2T(n/2) + O(n)$, which solves to $T(n) = O(n \log n)$. □

Acknowledgement. We wish to thank Haim Kaplan and Micha Sharir for their suggestions, which led to an improvement of an earlier version of Lemma 1.

References

[1] Pankaj K. Agarwal and Jiří 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)*, pages 11–19, 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-logarithmic 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. Dynamic geometric data structures via shallow cuttings. In *Proceedings of the 35th International Symposium on Computational Geometry (SoCG)*, 2019.

[12] 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.

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

[14] Timothy M. Chan and Simon Pratt. Two approaches to building time-windowed geometric data structures. In *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG)*, pages 28:1–28:15, 2016.

[15] Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discrete & Computational Geometry*, 56(4):866–881, 2016.

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

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

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

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

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

[21] 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.

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

[23] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2495–2504, 2017.

[24] 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.

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

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

[27] Simon Pratt. Three approaches to building time-windowed geometric data structures. Master's thesis, Cheriton School of Computer Science, University of Waterloo, 2016. `https://uwspace.uwaterloo.ca/handle/10012/10654`.

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

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

[30] 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.

[31] Cheng Sheng and Yufei Tao. FIFO indexes for decomposable problems. In *Proceedings of the 30th ACM Symposium on Principles of Database Systems (PODS)*, pages 25–35, 2011.

[32] Gelin Zhou. Two-dimensional range successor in optimal time and almost linear space. *Information Processing Letters*, 116(2):171–174, 2016.