Finding All Maximal Subsequences with Hereditary Properties

Drago Bokal[1], Sergio Cabello*[2], and David Eppstein†[3]

1 **Faculty of Natural Sciences and Mathematics, University of Maribor, Slovenia**
2 **Department of Mathematics, FMF, University of Ljubljana, Slovenia**
3 **Computer Science Department, University of California, Irvine, USA**

Abstract

Consider a sequence $s_1, \ldots, s_n$ of points in the plane. We want to find all maximal subsequences with a given hereditary property $\mathcal{P}$: find for all indices i the largest index $j^*(i)$ such that $s_i, \ldots, s_{j^*(i)}$ has property $\mathcal{P}$. We provide a general methodology that leads to the following specific results:

- In $O(n \log^2 n)$ time we can find all maximal subsequences with diameter at most 1.
- In $O(n \log n \log\log n)$ time we can find all maximal subsequences whose convex hull has area at most 1.
- In $O(n)$ time we can find all maximal subsequences that define monotone paths in some (subpath-dependent) direction.

The same methodology works for graph planarity, as follows. Consider a sequence of edges $e_1, \ldots, e_n$ over a vertex set V. In $O(n \log n)$ time we can find, for all indices i, the largest index $j^*(i)$ such that $(V, \{e_i, \ldots, e_{j^*(i)}\})$ is planar.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases convex hull, diameter, monotone path, sequence of points, trajectory

Digital Object Identifier 10.4230/LIPIcs.SOCG.2015.240

1 Introduction

The increasing availability of massive amounts of data regarding the spatial movements of smart phones, vehicles, tagged wild animals, ice sheets, etc., has led to an increasing interest in geometric algorithms for *trajectory analysis* [1, 4–6, 9, 13, 16, 17]. Such problems are a natural fit for the *windowed geometry* framework of Bannister et al. [2]: in this framework, a trajectory can be described by a sequence S of points in the plane (the vertices of a polyline), and we wish to develop data structures that can quickly answer queries about the shapes formed by contiguous subsequences of S. These queries may in turn be used for exploratory data analysis of a data set, or as subroutines for higher-level problems such as trajectory segmentation, clustering, or simplification.

In this paper, we consider queries for which the answer is a Boolean value: given a sequence $S = s_1, \ldots, s_n$, and a query subsequence $[i, j]$, does the queried subsequence of S have property $\mathcal{P}$ or not? We only consider hereditary properties, i.e., whenever a sequence has property $\mathcal{P}$, so do all of its subsequences. For example, the property of having a convex hull of area at most 1 is hereditary in this sense. For such problems, the issues of data

* Part of this research was done while visiting IST Austria. Supported by the Slovenian Research Agency, program P1-0297, projects J1-4106 and L7-5459.
† Supported in part by NSF grant 1228639 and ONR grant N00014-08-1-1015.

31st International Symposium on Computational Geometry (SoCG'15).
Editors: Lars Arge and János Pach; pp. 240–254

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

structure representation and query time become trivial: we need only store, for each index i, the largest index $j^*(i)$ such that the subsequence $s_i, \ldots, s_{j^*(i)}$ has property $\mathcal{P}$. With this information, the query reduces to a simple comparison of the endpoint j of the query interval with the endpoint $j^*(i)$ of the maximal interval starting at i with property $\mathcal{P}$. However, the preprocessing stage of this problem, in which we compute each of these values $j^*(i)$, can be highly nontrivial. That is the focus of our contribution: efficient algorithms for finding all of the maximal contiguous subsequences of S with the prescribed property.

Analogous windowed query data structures can be considered as well for non-geometric data, such as sequences of timestamped graph edges [3]. For such data, we may seek the maximal subsequences that have some monotone graph property such as being disconnected, being acyclic, being planar, etc.

1.1 New results

Let $S = s_1, \ldots, s_n$ be a sequence of points in the plane. We prove the following results:

- In $O(n \log n \log\log n)$ time we can find, for all indices i, the largest index $j^*(i)$ such that the convex hull of $s_i, \ldots, s_{j^*(i)}$ has at most unit area. In the trajectory problems, this models subsequences in which the moving object is either not moving significantly or is traveling close to a straight line.
- In $O(n \log^2 n)$ time we can find, for all indices i, the largest index $j^*(i)$ such that $s_i, \ldots, s_{j^*(i)}$ has at most unit diameter. In the trajectory problem, this models subsequences in which the object is not moving significantly.
- In $O(n)$ time we can find, for all indices i, the largest index $j^*(i)$ such that there exists a direction for which the path defined by $s_i, \ldots, s_{j^*(i)}$ is monotone. In the trajectory problem, this models subsequences in which the object is moving in some particular direction but may possibly be deviating from a straight line to avoid obstacles.

We develop a methodology, explained in Section 2, that should be useful for many other problems. As another application of these techniques beyond geometry, we show the following result about graph planarity. Let V be a vertex set and let $e_1, \ldots e_n$ be a sequence of edges with endpoints in V. We show how to compute in $O(n \log n)$ time, for all indices i, the largest index $j^*(i)$ such that the graph $(V, \{e_i, \ldots, e_{j^*(i)}\})$ is planar.

For the geometric problems, we do not use any heavy machinery and the results are clearly implementable. For graph planarity we use a deep result of Galil, Italiano, and Sarnak [15] that makes our result purely of theoretical interest.

1.2 Comparison with dynamic data structures

For our problems of finding maximal subsequences with property $\mathcal{P}$, there is a natural alternative approach based on dynamic geometric data structures. Suppose we have a data structure D that can maintain a dynamic set of points, subject to insertions and deletions, and answer queries that ask whether the current set has property $\mathcal{P}$. Then we may use D to compute the sequence of values $j^*(i)$, using a simple scan, as follows:

- Augment S by a special flag value s_{n+1} that cannot be part of a set with property $\mathcal{P}$.
- Initialize D to an empty data structure, and set $j = 0$.
- For $i = 1, 2, 3, \ldots$ do the following:
  - While the set in D has property $\mathcal{P}$, increase j by one and insert s_j into D.
  - Set $j^*(i) = j - 1$.
  - Delete s_i from D.

This algorithm performs n insertions and deletions in D and computes all values $j^*(i)$; its time is bounded by $O(n)$ times the time for a single insertion or deletion. However, for the problems we consider, this would be slower than the time bounds we give.

For instance, consider the problem of finding maximal subsequences of points whose convex hull has area at most 1. A natural approach is to use a dynamic data structure that maintains the area of the convex hull under insertions and deletions of points. The data structure by Overmars and van Leeuwen [20] can be easily extended to maintain the area of the convex hull of n points in $O(\log^2 n)$ time per update. We could use this data structure in the scan algorithm above to compute all maximal subsequences in $O(n \log^2 n)$; however, this is slower by a logarithmic factor than our algorithms. Chan [7] has improved the Overmars and van Leeuwen data structure but we are unsure whether this can be adapted to the convex hull area property and it would still be somewhat slower than our algorithm.

Chan [8] shows how to maintain the diameter dynamically in $O(\log^8 n)$ expected amortized time, improving a previous algorithm of Eppstein [11]. This implies that all maximal subsequences of diameter 1 can be computed in $O(n \log^8 n)$ expected time. This is significantly slower than the algorithm we give.

The monotonicity problem depends on the sequence in which the input points are given so it is not possible to express it using data structures based on dynamic point sets. Nevertheless, a similar scan algorithm could be used together with a data structure that detects whether a dynamic set of vectors (the differences of consecutive points in the input sequence) has the property of lying within a halfspace through the origin. This data structural problem can be solved by using a binary search tree (ordered radially around the origin) in logarithmic time per update. However, again, this would be slower than our algorithm.

For graph planarity we would need a dynamic data structure that that maintains a planar graph under insertion and deletion of edges. Moreover, we also need to be able to query whether the addition of an edge violates planarity. The best data structure for this takes $O(\sqrt{|V|}) = O(\sqrt{n})$ amortized time per query or operation [12]. Thus, we can find all maximal planar graphs in $O(n^{3/2})$ time, significantly slower than our algorithm. (There are better semi-dynamic data structures for planarity [10], but deletions are costly.)

There has also been research on faster dynamic data structures with restrictions on the update order, such as offline updates in which the entire sequence of updates is known in advance. Here, we do know the order of insertions and deletions, but we do not know how they interlace. In fact, the main substance of the problem is about figuring out when deletions should take place.

1.3 Additional related work

Łącki and Sankowski [19] considered a related windowed query framework for graph problems with an offline sequence of edge updates. As in our problems, queries specify a window within this sequence; however, the goal of a query is to determine whether some, all, or none of the versions of the graph within the window have a given property $\mathcal{P}$. For the geometric problems that we consider, an analogous type of problem would involve a data set consisting of a sequence of point insertions and deletions, and a query asking whether all, some, or none of the versions of the point set within a window into the query sequence have a given property. However, the graph properties considered by Łącki and Sankowski are different from the geometric and graph properties considered here.

A one-dimensional variant of the windowed diameter problem may be solved in constant time per query, using a range minimum data structure [14] to determine the minimum and maximum value within a query window. Applying this separately to each coordinate would

allow us to determine the L_∞ diameter of a query window. However, this approach does not generalize to Euclidean diameter, and although it can be made to work for the monotone direction problem, it would result in a more complicated solution than the one we give.

2 General strategy

We first review the notation we (ab)use. For any natural numbers a and b, we let $[a, b]$ denote the integer range $\{a, a+1, \ldots, b\}$. Henceforth, n will be used to denote the length of the input sequence. We write $[n]$ instead of $[1, n]$ and use $\mathbb{U} = \{(i, j) \in [n]^2 \mid i \le j\}$.

Consider a sequence $S = s_1, \ldots, s_n$ of points in the plane. For every pair of indices $(i, j) \in \mathbb{U}$ we define the ***subsequence*** $S[i, j] = s_i, \ldots, s_j$. All subsequences considered in this paper are contiguous subsequences. When $j < i$, $S[i, j]$ is the empty sequence. With a slight abuse of notation, we will sometimes treat $S[i, j]$ as a set instead of as a sequence; for example, we will talk about the diameter or the convex hull of $S[i, j]$.

A property $\mathcal{P}$ for subsequences is ***hereditary*** if it is closed under taking subsequences: if $S[i, j]$ has property $\mathcal{P}$, then $S[i', j']$ also has property $\mathcal{P}$ for all $i \le i' \le j' \le j$. All properties considered in this paper are hereditary.

Consider a fixed hereditary property $\mathcal{P}$. We consider a $n \times n$ matrix $A_\mathcal{P} = (A_\mathcal{P}(i, j))_{(i,j)\in\mathbb{U}}$, defined (only for pairs of indices in $\mathbb{U}$) by

$$A_\mathcal{P}(i, j) \;=\; \begin{cases} 1, & \text{if } S[i, j] \in \mathcal{P}, \\ 0, & \text{otherwise.} \end{cases}$$

Values in the bottom triangle $\{(i, j) \mid j < i\}$ are undefined. We want to find for each row i the last index $j^*(i)$ with $A_\mathcal{P}(i, j^*(i)) = 1$. When the property $\mathcal{P}$ is clear from the context, we drop the subscript and simply write A instead of $A_\mathcal{P}$.

A ***rectangle*** (of indices) is a subset of indices

$$[a, a+h] \times [b, b+w] \;=\; \{(i, j) \in [n]^2 \mid a \le i \le a+h,\ b \le j \le b+w\}.$$

For a rectangle $R = [a, a+h] \times [b, b+w]$, its ***height*** is $\text{height}(R) = h + 1$ and its ***width*** is $\text{width}(R) = w + 1$. By ***solving a rectangle*** $R = [a, a+h] \times [b, b+w]$ we mean finding, for each index $i \in [a, a+h]$, the last nonzero of row i of matrix A that lies inside rectangle R. In general, our algorithms will consider rectangles $[a, a+h] \times [b, b+h]$ with $a + h \le b$, that is, contained in $\mathbb{U}$. A rectangle is ***anchored at the diagonal*** if it is of the type $[a-h, a] \times [a, a+w]$, that is, its bottom left corner lies on the diagonal $\{(i, i) \mid i \in [n]\}$.

We will assume that $A(i, i) = 1$, for all $i \in [n]$; that is, any single point of S always satisfies property $\mathcal{P}$. (Otherwise $j^*(i)$ is not defined.) Our algorithms will consider rectangles $R = [a, a+h] \times [b, b+w]$ with the property that, for all $i \in [a, a+h]$, $j^*(i) \in [b, b+w]$. That is, these rectangles contain the last nonzero of A in each of their rows. We call a rectangle with this property a ***frontier rectangle***. Thus, solving a frontier rectangle is equivalent to finding the values $j^*(i)$ for all $i \in [a, a+h]$.

2.1 Decomposing into anchored rectangles

We are going to use a greedy procedure to reduce the problem to a search within disjoint frontier rectangles anchored at the diagonal that together have $O(n)$ height and width.

Take a sequence of indices $\alpha = a_1, a_2, \ldots, n$ such that $a_1 = 1$ and $a_k = j^*(a_{k-1})$, that is, a_k is the largest index with $A(a_{k-1}, a_k) = 1$. (In the special case that $a_k = a_{k-1}$, we redefine a_k to $a_{k-1} + 1$.) This is a greedy decomposition of the sequence into subsequences with

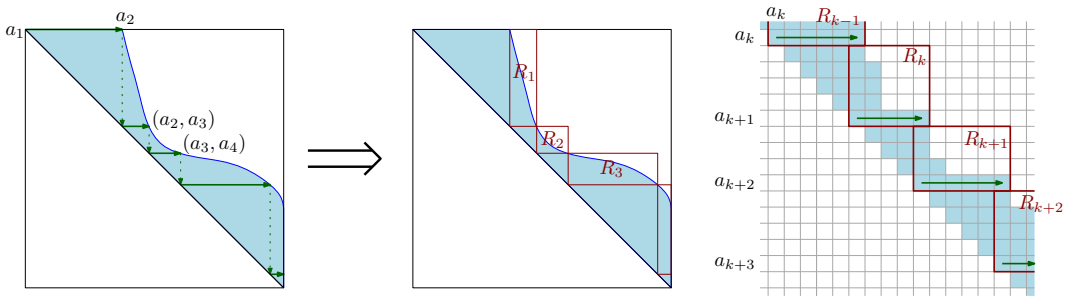

Figure 1 Schema showing the greedy procedure to decompose the problem into (red) frontier rectangles anchored at the diagonal. The blue region denotes entries of matrix A with value 1.

property $\mathcal{P}$. The subsequences are disjoint, except for the starting and ending points a_k, and maximal with respect to this almost-disjoint property. For each index a_k in α, define the rectangle $R_k = [a_k + 1, a_{k+1}] \times [a_{k+1}, a_{k+2}]$. A schematic view is offered in Figure 1.

Each index $i \in [n]$ appears at most once as a first coordinate and at most twice in the second coordinate of rectangles $R_1, R_2, \ldots$. Therefore

$$\sum_k \big(\text{height}(R_k) + \text{width}(R_k)\big) \;=\; O(n).$$

By construction, each R_k is a frontier rectangle anchored at the diagonal. Solving the rectangles $R_1, R_2 \ldots$ we readily obtain all maximal subsequences with property $\mathcal{P}$. We summarize.

▶ **Lemma 2.1.** *Assume that we have the following two subroutines for sequence S and property $\mathcal{P}$:*

(a) *Given an index $a \in [n]$, find the largest index $j^*(a)$ such that $S[a, j^*(a)] \in \mathcal{P}$. This takes $T_{greedy}(j^*(a) - a)$ time for a certain convex function $T_{greedy}(\cdot)$.*

(b) *Given a frontier rectangle R of indices anchored at the diagonal, solve R. This takes $T_{rect}(\text{height}(R) + \text{width}(R))$ time for a certain convex function $T_{rect}(\cdot)$.*

Then we can find all maximal subsequences with property $\mathcal{P}$ in $O(n) + T_{greedy}(O(n)) + T_{rect}(O(n))$ time. ◀

2.2 Solving an anchored rectangle

To solve a frontier rectangle anchored at the diagonal, we are going to use a recursive divide-and-conquer method. The subproblems of this method will be defined by frontier rectangles contained in $\mathbb{U}$, but not necessarily anchored at the diagonal.

Consider a frontier rectangle $[a, a + h] \times [b, b + w]$ contained in $\mathbb{U}$. We use a methodology similar to binary search. See Figure 2 for an schematic view. We select an index m halving the interval $[a, a + h]$. Then we find the largest index c such that $A(m, c) = 1$. With this, we infer the following information:

- In the rectangle $[m, a + h] \times [b, c]$, all the values of A are 1.
- In the rectangle $[a, m] \times [c + 1, b + w]$, all the values of A are 0.

We then recurse in the frontier rectangle $[a, m - 1] \times [b, c]$ and in the frontier rectangle $[m + 1, a + h] \times [c, b + w]$. Since in each step we halve the area where we continue the search, we have a recursion of depth $O(\log(wh))$.

However, the subproblems do not really get smaller: to solve the rectangle $R = [a, a+h] \times [b, b + w]$, we have to consider the subsequence $S[a, b + w]$, which has size $w + b - a$. For late

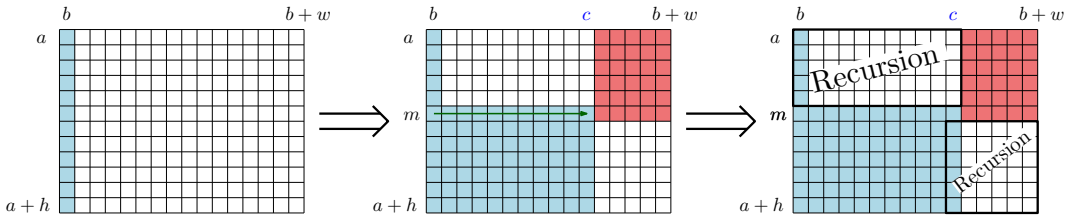

Figure 2 Schema showing the strategy to solve a frontier rectangle. The blue region denotes entries known to have value 1. The red region denotes entries known to have value 0.

subproblems, this may be larger than $w + h$ or wh. However, any of the subsequences $S[i, j]$, where $(i, j) \in R$, can be decomposed into $S[i, a + h]$, $S[a + h, b]$ and $S[b, j]$. The middle sequence $S[a + h, b]$ may be arbitrarily large, but it is a "common factor" to all subsequences $S[i, j]$, $(i, j) \in R$. We replace the subsequence $S[a + h, b]$ by a *sketch* of size $O(w + h)$. The definition of sketch depends on the problem at hand, but the idea is that it should encode the role of $S[a + h, b]$ in the subsequences $S[i, j]$, for all $(i, j) \in R$. In fact, such sketches are also important to find efficiently the index c that controls the division for recursive calls.

To analyze such algorithm, the following technical result will be useful.

▶ **Lemma 2.2.** *The recursion*

$$T(h, w) = \begin{cases} O(h + w) + T(\lfloor h/2 \rfloor, w') + T(\lfloor h/2 \rfloor, w - w') & \text{if } h \geq 2, \\ O(w) & \text{if } h = 0 \text{ or } 1, \end{cases}$$

where $0 \leq w' \leq w$*, implies that* $T(h, w) = O((h + w) \log h)$. ◀

3 Directional monotonicity

In this section, we will regard the subsequence $S[i, j]$ as a *polygonal path*. Consider the unit circle $\mathbb{S}^1$. For a direction $\vec{u} \in \mathbb{S}^1$, the path $S[i, j]$ is ***$\vec{u}$-monotone***, if it is always increasing in the direction $\vec{u}$, that is, the scalar product of $\overrightarrow{s_k s_{k+1}}$ and $\vec{u}$ is positive for each $k \in [i, j - 1]$. The path $S[i, j]$ is ***monotone*** if it is $\vec{u}$-monotone for some direction $\vec{u} \in \mathbb{S}^1$. Let $\Theta(i, j) \subset \mathbb{S}^1$ be the set of directions $\vec{u}$ such that $S[i, j]$ is $\vec{u}$-monotone.

▶ **Lemma 3.1.** *Given an index* $a \in [n]$*, we can find the largest index* $j^*(a) \in [n]$ *such that the path* $S[a, j^*(a)]$ *is monotone in* $O(j^*(a) - a)$ *time.*

Proof. Starting with $j = a + 1$, we increment j until we get that $j = n + 1$ or $S[a, j]$ is not monotone, and then return $j - 1$. At each step, we compute the interval $\Theta(a, j)$ in constant time using that $\Theta(a, j) = \Theta(a, j - 1) \cap \{\vec{u} \in \mathbb{S}^1 \mid \langle \overrightarrow{s_{j-1} s_j}, \vec{u} \rangle > 0\}$. ◀

Lemma 3.1 provides the subroutine needed in Lemma 2.1(a). The next result provides the subroutine needed in Lemma 2.1(b).

▶ **Lemma 3.2.** *Consider a frontier rectangle* R *of indices anchored at the diagonal. We can solve the rectangle* R *in* $O(\text{height}(R) + \text{width}(R))$ *time.*

Proof. Let R be the rectangle $[a-h, a] \times [a, a+w]$. We compute for each index $j \in [a+1, a+w]$ the set of directions $\Theta(a, j)$. This is done incrementally in $O(w)$ time. We compute for each index $i \in [a - h, a - 1]$ the set of directions $\Theta(i, a)$. This is done by tracing the *reverse* of the path $S[a - h, a]$ and using the fact that a path is $\vec{u}$-monotone if and only if its reversal is $(-\vec{u})$-monotone.

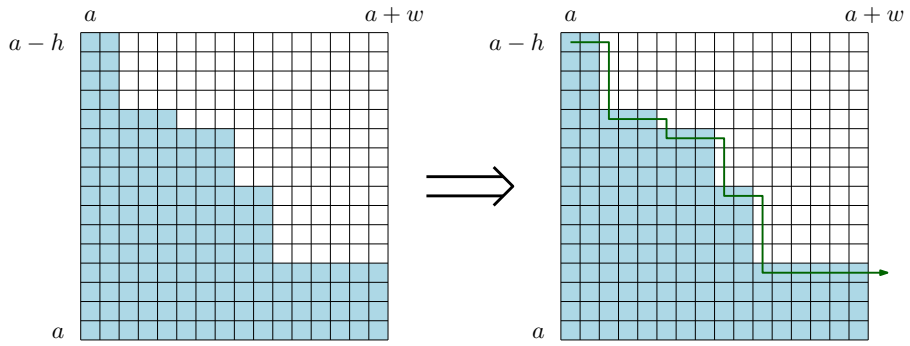

Figure 3 Path followed by (i, j) in the algorithm of Lemma 3.2 to solve a frontier rectangle anchored to the diagonal. The blue region describes query windows that form monotone paths.

Now we just walk along the boundary between monotone and not-monotone in the rectangle R. See Figure 3. Set $i = a - h$ and $j = a + 1$. At each iteration, we compute $\Theta(i, j)$ in constant time using the fact that $\Theta(i, j) = \Theta(i, a) \cap \Theta(a, j)$. If $\Theta(i, j) \neq \emptyset$, we increment j and go to the next iteration. If $\Theta(i, j) = \emptyset$, we deduce that $j^*(i) = j - 1$, increment i, and go to the next iteration. We finish when the pair (i, j) is outside the rectangle R. In this case, we deduce that $j^*(i') = a + w$ for $i' = i, \ldots, a$.

The running time is $O(h + w)$ because at each iteration we spend constant time and increment either i or j. ◀

▶ **Theorem 3.3.** *Let $S = s_1, \ldots, s_n$ be a polygonal path in the plane. In $O(n)$ time we can compute, for all indices $i \in [n]$, the largest index $j^*(i)$ such that the polygonal path $s_i, \ldots, s_{j^*(i)}$ is monotone.*

Proof. Lemmas 3.1 and 3.2 give the subroutines required in the hypothesis of Lemma 2.1, with running times $T_{greedy}(n) = O(n)$ and $T_{rect}(n) = O(n)$. Lemma 2.1 implies that we can find all maximal subsequences in $O(n)$ time. ◀

Note that in this problem, we did not need the recursive approach discussed in Section 2.2. This is because $\Theta(i, j)$, which plays the role of a sketch, has a constant-size description.

4 Diameter of point sets

For a set of points P in the Euclidean plane, its diameter $\mathrm{diam}(P)$ is the maximum distance between any two points: $\mathrm{diam}(P) = \max_{p,p' \in P} \|p - p'\|$. The diameter of n points can be computed in $O(n \log n)$ time; see for example [21, Chapter 4].

4.1 Sketches

Let P and S be sets of points in the Euclidean plane. A subset $Q \subset P$ is a ***diam-sketch of P with respect to S*** if

(i) for each $T \subset S$ we have $\mathrm{diam}(P \cup T) = \mathrm{diam}(Q \cup T)$, and
(ii) $|Q| = O(|S|)$.

Diam-sketches can be constructed using standard tools and have a certain composition property, as the following lemma explains. See Figure 4 for an example of the construction.

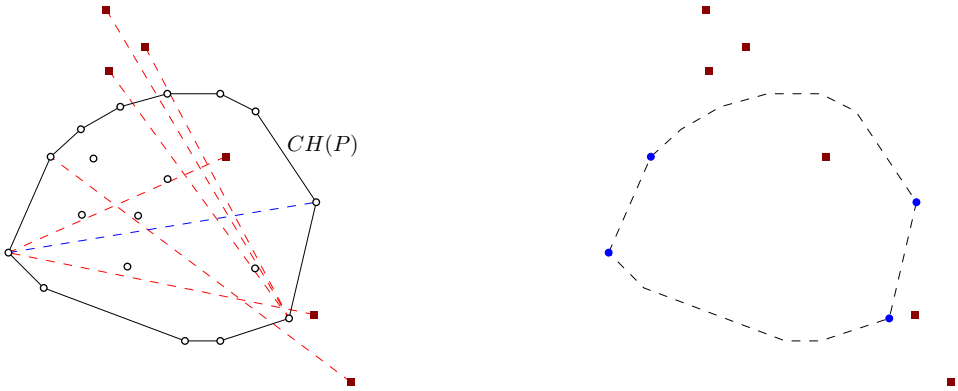

Figure 4 Example of diam-sketch. Left: the dotted black points correspond to P, the squared red points to S. A diametral pair of P and points furthest away from each point of S are shown with dashed segments. Right: the resulting diam-sketch, as constructed in the proof of Lemma 4.1(a).

▶ **Lemma 4.1.** *Diam-sketches have the following properties.*

(a) *Given sets P and S of cardinality at most n, we can compute a diam-sketch of P with respect to S in $O(n \log n)$ time.*

(b) *Let Q be a diam-sketch of P with respect to S, X be a diam-sketch of $Q \cup S_1$ with respect to S_2, and $S_1 \cup S_2 \subset S$. Then X is a diam-sketch of $P \cup S_1$ with respect to S_2.*

Proof Sketch. We show (a) giving an explicit construction. We set $Q = \emptyset$, add to Q a diametral pair of P and, for each point $s \in S$, add a point of P that is furthest from s. Such a set Q can be constructed in $O((|P| + |S|) \log(|P|)) = O(n \log n)$ time using standard tools: we compute the diameter of P, build the furthest-point Voronoi diagram VD of P, and locate each point of S in VD. The details are standard. See Figure 4 for an example of the construction. A small case analysis shows that Q is indeed a diam-sketch.

To show (b), consider any $T \subset S_2$. We have to show that $\operatorname{diam}(P \cup S_1 \cup T) = \operatorname{diam}(X \cup T)$. Since Q is a diam-sketch of P with respect to $S \supset S_1 \cup T$ and X is a diam-sketch of $Q \cup S_1$ with respect to $S_2 \supset T$, we have

$$\begin{aligned} \operatorname{diam}(P \cup S_1 \cup T) &= \operatorname{diam}(P \cup (S_1 \cup T)) = \operatorname{diam}(Q \cup (S_1 \cup T)) \\ &= \operatorname{diam}((Q \cup S_1) \cup T) = \operatorname{diam}(X \cup T). \end{aligned}$$

◀

4.2 Algorithms

▶ **Lemma 4.2.** *Let P be a set of points and $S = s_1, \ldots, s_n$ a sequence of points. Assume that we have a diam-sketch Q of P with respect to S. In $O(n \log n)$ time, we can find the largest index j^* such that $\operatorname{diam}(P \cup S[1, j^*]) \leq 1$.*

Proof. We proceed with a binary search. We initialize the search with $\ell = 0$, $r = n$, and $X = Q$. Through the binary search, we maintain the invariant that $\ell \leq j^* \leq r$ and X is a diam-sketch of $P \cup S[1, \ell]$ with respect to $S[\ell + 1, r]$. In an iteration, we set $m = \lceil (\ell + r)/2 \rceil$ and check whether $\operatorname{diam}(X \cup S[\ell + 1, m]) \leq 1$. If the diameter is at most 1, then we continue the search with $\ell = m$, and set X to be a diam-sketch of $X \cup S[\ell, m]$ with respect to $S[m + 1, r]$. If, on the other hand, the diameter is larger than 1, then we continue the search with $r = m - 1$, and set X to be a diam-sketch of X with respect to $S[\ell + 1, m - 1]$. We finish the search when $\ell = r$ by returning ℓ.

Validity of the invariant $\ell \le j^* \le r$ follows from the standard argument used for binary search. Validity of the property that X is a diam-sketch of $P \cup S[1,\ell]$ with respect to $S[\ell+1,r]$ follows by induction and Lemma 4.1(b). When we continue on the right side (setting $\ell = m$), we apply Lemma 4.1(b) with $S_1 = S[\ell,m]$ and $S_2 = S[m+1,r]$. When we continue on the left side (setting $r = m-1$), we apply Lemma 4.1(b) with $S_1 = \emptyset$ and $S_2 = S[\ell+1,m-1]$.

Correctness of the method follows from the invariant because at the end, when $r = \ell$, we have $\operatorname{diam}(P \cup S[1,\ell]) = \operatorname{diam}(X) \le 1$ and $r = j^* = \ell$.

For the running time, note that at each step, we handle $O(|X| + r - \ell)$ points. Since X is always a diam-sketch with respect to $S[\ell+1,r]$, we have $|X| = O(r-\ell)$. At each iteration, we compute the diameter of $O(r-\ell)$ points and the diam-sketch of $O(r-\ell)$ points with respect to a set of size $O(r-\ell)$ using Lemma 4.1(a). This means that we spend $O((r-\ell)\log(r-\ell))$ at each iteration. Since at each iteration the value $r-\ell$ decreases geometrically, we conclude that the total running time is $O(n \log n)$. ◀

▶ **Lemma 4.3.** *Consider a frontier rectangle R anchored at the diagonal with height h and width w. We can solve R in $O((h+w)\log^2(h+w))$ time.*

Proof. We give a recursive algorithm. A recursive subproblem is described by a frontier rectangle $[a,a+h] \times [b,b+w]$ contained in $\mathbb{U}$, and a diam-sketch Q of $S[a+h,b]$ with respect to $S[a,a+h-1] \cup S[b+1,b+w]$. The original problem is a problem of such type, where $a+h=b$ and $S[a+h,b] = Q = \{s_b\}$.

If $h = 1$, we use Lemma 4.2 twice, once for each row, to find $j^*(a)$ and $j^*(a+1)$. For the row $a+1$ we use Q and the sequence $S[b+1,b+w]$. For the row a we use Q and the sequence $s_a, S[b+1,b+w]$. In this case, we need $O(w \log w)$ time. The case $h = 0$ is similar.

Let us now consider the case when $h \ge 2$ and thus the rectangle has at least three rows. We use the divide-and-conquer approach discussed in Section 2.2. Set $m = a + \lfloor h/2 \rfloor$. We find the last index $c \in [b, b+w]$ such that $\operatorname{diam}(S[m,c]) \le 1$. (Here we are using the property of being a frontier rectangle to infer that $c \ge b$ and thus $(m,c) \in R$.) We have obtained that $j^*(m) = c$. We then recurse on the rectangles $R_1 = [a,m-1] \times [b,c]$ and $R_2 = [m+1,a+h] \times [c,b+w]$. Note that R_1 and R_2 are frontier rectangles; see Figure 2. To recurse in the rectangle R_1, we use a diam-sketch Q_1 of $Q \cup S[m,a+h]$ with respect to $S[a,m-1] \cup S[b,c]$. Note that Q_1 is a diam-sketch of $S[m,b]$ with respect to $S[a,m-1] \cup S[b,c]$ because of Lemma 4.1(b), and thus it is the appropriate diam-sketch for the recursive call. Similarly, for the recursion on the rectangle R_2, we use a diam-sketch Q_2 of $Q \cup S[b,c-1]$ with respect to $S[m+1,a+h] \cup S[c,b+w]$. Again, Q_2 is a diam-sketch of $S[a+h,c-1]$ with respect to $S[m+1,a+h] \cup S[c,b+w]$ because of Lemma 4.1(b), and it provides appropriate ground for recursion. This finishes the description of the algorithm.

To analyze the running time, note that Q has size $O(h+w)$ because it is a diam-sketch with respect to $h+w$ points. If $h \le 1$, we spend $O(w \log w)$ time. Let us now look at the case $h > 1$. The index c can be found in $O((h+w)\log(h+w))$ time using Lemma 4.2 with Q and the sequence $S[m,a+h-1], S[b+1,b+w]$. The sets Q_1 and Q_2 can be computed in $O((h+w)\log(h+w))$ time using Lemma 4.1(a) and noting that Q has size $O(h+w)$. Thus we spend $O((h+w)\log(h+w))$ time plus the time for recursive calls in R_1 and R_2. Let $w' = c-b$. The rectangle R_1 has $m-a \le \lfloor h/2 \rfloor$ rows and $c-b+1 = w'+1$ columns, while the rectangle R_2 has $a+h-m \le \lfloor h/2 \rfloor + 1$ rows and $b+w-c+1 = w-w'+1$ columns. Therefore, denoting by $T(h,w)$ the running time for a rectangle with $h+1$ rows and $w+1$

columns, we have

$$T(h,w) = \begin{cases} O((h+w)\log(h+w)) + T(\lfloor h/2 \rfloor, w') + T(\lfloor h/2 \rfloor, w - w') & \text{if } h \geq 2, \\ O(w \log(h+w)) & \text{if } h \leq 1. \end{cases}$$

Taking the factor $O(\log(h+w))$ out, Lemma 2.2 implies that $T(h,w) = O((h+w)\log^2(h+w))$. ◀

Buchin et al. [6] give the subroutine needed in Lemma 2.1(a) with $T_{greedy}(n) = O(n \log n)$. An exponential search and Lemma 4.2 can also be used to obtain the same result. Lemma 4.3 gives the subroutine needed in Lemma 2.1(b) with $T_{rect}(n) = O(n \log^2 n)$. Then Lemma 2.1 implies the following.

▶ **Theorem 4.4.** *Let $S = s_1, \ldots, s_n$ be a sequence of points in the plane. In $O(n \log^2 n)$ time, we can compute, for all indices $i \in [n]$, the largest index $j^*(i)$ with $\operatorname{diam}(s_i, \ldots, s_{j^*(i)}) \leq 1$.*

5 Area of the convex hull

In this section, we will for simplicity assume general position: no two points have the same x-coordinate and no three points are collinear.

For a point set P, we denote by $CH(P)$ its convex hull. For each point s outside $CH(P)$, let $\tau(s,P)$ be the two points on the boundary of $CH(P)$ that support the tangents to $CH(P)$ through s.

5.1 Sketches

Let P be a set of points and let $S = s_1, \ldots, s_n$ be a sequence of points. Let $p_{\max}$ and $p_{\min}$ be the points of P with largest and smallest x-coordinates, respectively. The ***CH-sketch of P with respect to the sequence S*** is the point set

$$\{p_{\max}, p_{\min}\} \cup \left(\bigcup_{i \in [n]} \tau(s_i, P \cup S[1, i-1]) \cap P \right).$$

An example is given in Figure 5. The intuition is that the CH-sketch should contain the points of P that support some tangent during the iterative construction of $CH(P \cup S[1,i])$, for $i = 1, \ldots, n$. We add $p_{\max}$ and $p_{\min}$ for convenience: we will later maintain the area of the upper hulls of P and the CH-sketch, and it is slightly simpler if their starting and ending points match.

Note that we define CH-sketches with respect to *sequences*, while sketches in the previous section were with respect to sets. We do this to achieve better efficiency.

Let Q denote the CH-sketch of P with respect to S. We have the following straightforward consequences of the definition:

- **(i)** $|Q| = O(|S|)$ because each point of S contributes at most two points to Q.
- **(ii)** Each point of Q is a vertex of $CH(P)$ because it is in P and supports a tangent to $CH(P \cup X)$ for some X.
- **(iii)** For each $i \in [n]$, we have $\tau(s_i, P \cup S[1,i-1]) = \tau(s_i, Q \cup S[1,i-1])$, as a point supporting a tangent to $CH(P \cup S[1,i-1])$ through s_i is either in Q by definition or in $S[1,i-1]$.

We next discuss some properties about composition of CH-sketches. In our algorithm, we are going to keep two CH-sketches, each with respect to a different sequence. Because of this, some statements become cumbersome.

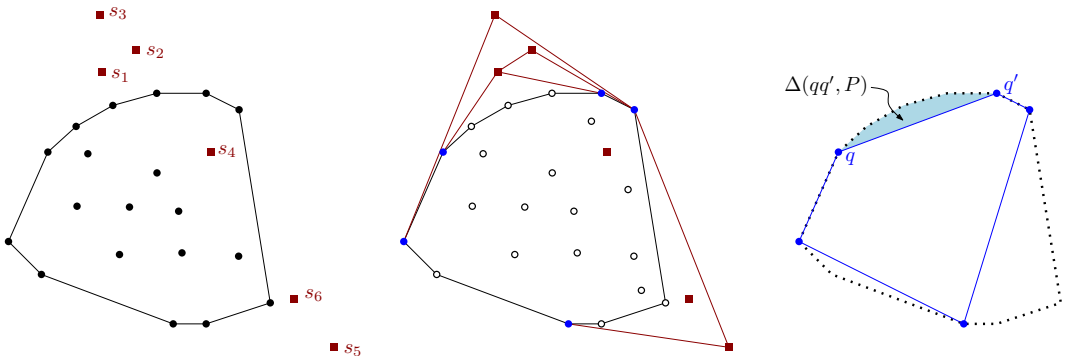

Figure 5 Left: A set P of points (black dots) with its convex hull and the sequence $S = s_1, \ldots, s_6$ (red squares). Center: the sequence of convex hulls $CH(P \cup S[1, i])$ for $i = 1, \ldots, 6$. The points of the CH-sketch Q of P with respect to S are marked in solid blue. Right: $CH(Q)$ for the CH-sketch of P and the clipped region $\Delta(qq', P)$ for an edge qq' of $CH(Q)$.

▶ **Lemma 5.1.** *Let Q_1 be a CH-sketch of P with respect to a sequence $S_1[1, n_1]$. Let Q_2 be a CH-sketch of P with respect to a sequence $S_2[1, n_2]$. Consider indices c_1 and c_2 such that $1 \leq c_1 \leq n_1$ and $1 \leq c_2 \leq n_2$.*

(a) *If Q' is a CH-sketch of $Q_1 \cup S_1[1, c_1]$ with respect to $S_1[c_1 + 1, n_1]$, then Q' is a CH-sketch of $P \cup S_1[1, c_1]$ with respect to $S_1[c_1 + 1, n_1]$.*

(b) *If Q' is a CH-sketch of $Q_2 \cup S_1[1, c_1]$ with respect to $S_2[1, c_2]$, then Q' is a CH-sketch of $P \cup S_1[1, c_1]$ with respect to $S_2[1, c_2]$.*

5.2 Clipped regions

Let P be a set of points and let Q be a subset of the vertices of $CH(P)$. Each edge qq' of $CH(Q)$ separates a portion of $CH(P) \setminus CH(Q)$ from $CH(Q)$. We denote such a region by $\Delta(qq', P)$. See Figure 5, right. We use $\Delta(Q, P)$ for the family of ***clipped regions*** $\{\Delta(qq', P)\}_{qq'}$, where qq' iterates over all edges of $CH(Q)$.

Let Q be a CH-sketch of P with respect to S. Consequence (iii) of the definition of CH-sketch is important to easily obtain the area of $CH(P \cup S[1, i])$ from the area of $CH(Q \cup S[1, i])$ and the area of each of the clipped regions $\Delta(Q, P)$. Indeed, for every index i, a clipped region of $\Delta(Q, P)$ is contained either in $CH(Q \cup S[1, i])$ or in the closure of its complement. Therefore

$$\text{area}(CH(P \cup S[1, i])) \;=\; \text{area}(CH(Q \cup S[1, i])) + \sum_{qq'} \text{area}(\Delta(qq', P)),$$

where the sum is taken over all edges qq' of $CH(Q)$ that are also edges of $CH(Q \cup S[1, i])$. See Figure 5 for an example. We use clipped regions to keep track of area difference between $CH(P)$ and $CH(Q)$ though the addition of points of S.

5.3 Algorithms

We are going to keep point sets sorted by their x-coordinates. The sequence S is also going to be kept sorted by x-coordinates. This *does not* mean that the x-coordinates of $s_1, s_2, \ldots, s_n$ are increasing when these points are given in sequence order. This means that, besides the sequence S, we have another list where the elements appearing in S are sorted by x-coordinates. For a set or sequence S, we will use $\mathcal{L}_x(S)$ to denote the list containing S sorted by x-coordinate.

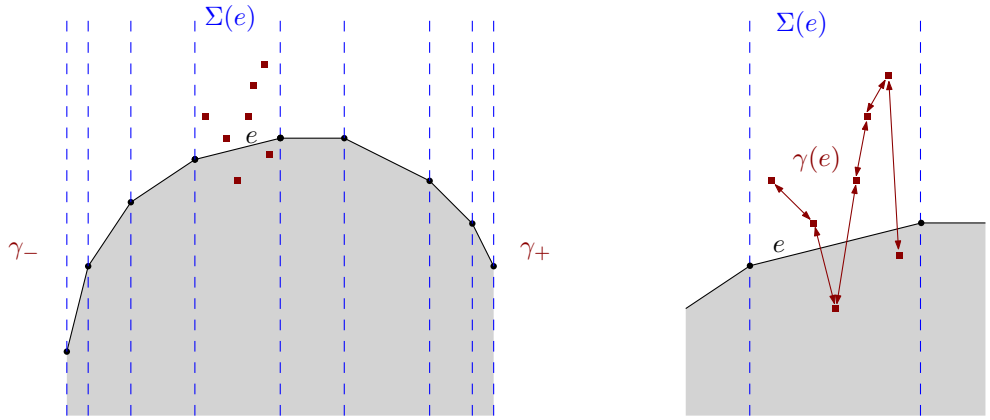

Figure 6 Data maintained during the incremental algorithm in the proof of Lemma 5.2.

▶ **Lemma 5.2.** *Let P be a set and S a sequence of points, both of cardinality at most n. Assume that we have the corresponding lists $\mathcal{L}_x(P)$ and $\mathcal{L}_x(S)$. We can compute a CH-sketch of P with respect to S in $O(n \log\log n)$ time. Moreover, the CH-sketch is obtained sorted by x-coordinate.*

Proof Sketch. We iteratively add the points $s_1, \ldots, s_n$ of S, maintain $CH(P \cup S[1,i])$, and mark the points $\tau(s_i, P \cup S[1,i-1]) \cap P$ to be added to the CH-sketch. We maintain separately the upper and the lower hull of $P \cup S[1,i]$. We discuss only the upper hull.

Through the iterative procedure, we use a list UH that stores the edges of the upper hull of $P \cup S[1,i]$. For each segment e of UH, let $\Sigma(e)$ be the vertical slab contained between the two vertical lines through the endpoints of e. For each edge e in UH, we have a list $\gamma(e)$ of the points of $S[i+1,n]$ contained in $\Sigma(e)$, sorted by x-coordinate. We have two additional lists, γ_- and γ_+, that contain the points of $S[i+1,n]$ to the left and to the right of UH, respectively, also sorted by x-coordinate. See Figure 6. We also maintain a van Emde Boas tree [22] for the vertices of UH. Its purpose is to find, at the time of inserting s_i, the segment e of UH whose slab $\Sigma(e)$ contains s_i. Using the order given by x-coordinates, this is a predecessor query in UH. We can identify each point of $P \cup S$ with its rank in the order given by x-coordinates, and use those ranks as keys for the van Emde Boas tree. Thus, we have an universe of $|P \cup S| = O(n)$ elements and each operation takes $O(\log\log n)$ time.

The data can be initialized in $O(n)$ time computing $CH(P)$ from $\mathcal{L}_x(P)$ and with a simultaneous scanning of UH and $\mathcal{L}_x(S)$. The insertion of a point s_i of S starts locating the edge e such that $\gamma(e)$ contains s_i and the appearance of s_i in $\gamma(e)$. We then have to update the convex hull UH and update the lists making some local operations. The insertion of s_i takes $O(\log\log n + |k_{i-1} - k_i|)$ time, where k_i denotes the number of vertices in the upper hull of $P \cup S[1,i]$. At the end of the insertion, we can obtain the points $\tau(s_i, P \cup S[1,i-1])$ of the upper hull and mark them for addition to the CH-sketch. Since each point of $P \cup S$ can be deleted at most once, the time over all insertions is $O(n \log\log n)$.

When we have inserted all the points of S, we construct the CH-sketch going through $\mathcal{L}_x(P)$ and selecting those marked for addition to the CH-sketch. We also insert the first and last point of $\mathcal{L}_x(P)$, since they have smallest and largest x-coordinate. ◀

An approach similar to the one used in the proof of Lemma 5.2 can be used to incrementally maintain the area of $CH(Q \cup S[1,i])$, for $i = 1, \ldots, n$. If Q is a CH-sketch of P with respect to S, we can then use the area of the clipped regions $\Delta(Q,P)$ to compute the area of

$CH(P \cup S[1,i])$, for $i = 1, \ldots, n$. We just have to notice that, for each $i \in [n]$, a clipped region of $\Delta(Q, P)$ is contained in $CH(P \cup S[1,i])$ or in its complement. This leads to the following.

▶ **Lemma 5.3.** *Let P be a set of points and let $S = s_1, \ldots, s_n$ be a sequence of points. Assume that we have a CH-sketch Q of P with respect to S and, for each edge qq' of $CH(Q)$, the area of $\Delta(qq', P)$. Furthermore, assume that we have the corresponding lists $\mathcal{L}_x(Q)$ and $\mathcal{L}_x(S)$. In $O(n \log\log n)$ time, we can find the largest index $j^* \in [n]$ such that* $\text{area}(CH(P \cup S[1, j^*])) \leq 1$. ◀

▶ **Lemma 5.4.** *Consider a frontier rectangle R anchored at the diagonal with height h and width w. We can solve R in $O((h+w)\log(h+w)\log\log(h+w))$ time.*

Proof. (Sketch) We follow very closely the proof of Lemma 4.3. However, the description of a recursive subproblem is given by:

(i) a frontier rectangle $[a, a+h] \times [b, b+w]$ contained in $\mathbb{U}$;
(ii) a CH-sketch Q_{ver} of $S[a+h, b]$ with respect to the reversal of $S[a, a+h-1]$;
(iii) a CH-sketch Q_{hor} of $S[a+h, b]$ with respect to $S[b+1, b+w]$;
(iv) lists $\mathcal{L}_x(Q_{ver})$, $\mathcal{L}_x(Q_{hor})$, $\mathcal{L}_x(S[a, a+h-1])$, and $\mathcal{L}_x(S[b+1, b+w])$;
(v) the convex hull $CH(Q)$, where $Q = Q_{hor} \cup Q_{ver}$; and
(vi) the area of each of the clipped regions $\Delta(Q, S[a+h, b])$.

Note that the description of such a subproblem has size $O(h+w)$.

We construct the base problem, to start the recursion, in $O((h+w)\log(h+w))$ time as follows. The rectangle $[a, a+h] \times [b, b+w]$ is given as the input, with $a + h = b$. We have $Q_{ver} = Q_{hor} = \{s_b\}$. The lists $\mathcal{L}_x(Q_{ver})$ and $\mathcal{L}_x(Q_{hor})$ have only one element. The lists $\mathcal{L}_x(S[a, a+h-1])$ and $\mathcal{L}_x(S[b+1, b+w])$ can be constructed in $O(h \log h)$ and $O(w \log w)$ time, respectively, by just sorting the points from scratch. The remaining data is trivial.

Let us now discuss how we solve a subproblem appearing in the recursion. The case $h \leq 1$ is easier, takes $O(w \log\log w)$ time, and we omit it.

Consider now the case when $h \geq 2$ and thus the rectangle has at least three rows. We use the divide-and-conquer approach discussed in Section 2.2 and already used in Lemma 4.3. Set $m = a + \lfloor h/2 \rfloor$, find the last index $c \in [b, b+w]$ such that $\text{area}(CH(S[m, c])) \leq 1$, and recurse on the rectangles $R_1 = [a, m-1] \times [b, c]$ and $R_2 = [m+1, a+h] \times [c, b+w]$. Recall Figure 2.

The value c can be found in $O((h+w)\log\log(h+w))$ time using Lemma 5.3, as follows. We compute the CH-sketch Q_m of $Q_{hor} \cup S[m, a+h-1])$ with respect to $S[b+1, b+w]$. Because of Lemma 5.2 and since the input can be obtained sorted by x-coordinate from $\mathcal{L}_x(Q_{hor})$, $\mathcal{L}_x(S[a, a+h-1])$, and $\mathcal{L}_x(S[b+1, b+w])$, this can be done in $O((h+w)\log\log(h+w))$ time. Because of Lemma 5.1(b), where S_1 is the reversal of $S[a, a+h-1]$ and $S_2 = S[b+1, b+w]$, Q_m is a CH-sketch of $S[a+h, b] \cup S[m, a+h-1]) = S[m, b]$ with respect to $S[b+1, b+w]$. We can show that the area of the clipped regions $\Delta(Q_m, S[m, b])$ can be obtained in $O((h+w)\log\log(h+w))$ time. Thus Q_m and $S[b+1, b+w]$ satisfy the hypothesis of Lemma 5.3, as needed to find c.

In $O((h+w)\log\log(h+w))$ time, we can collect the data for the recursive calls. For this, we use Lemma 5.2 to compute CH-sketches of CH-sketches with respect to subsequences. Lemma 5.1 is then used to argue that we are indeed computing the CH-sketches required for the recursive call. We omit the detailed arguments.

Thus, we can construct the recursive subproblems in $O((h+w)\log\log(h+w))$ time. It follows that the time $T(h, w)$ to solve a recursive subproblem with $h+1$ rows and $w+1$ rows

is given by

$$T(h,w) = \begin{cases} O((h+w)\log\log(h+w)) + T(\lfloor h/2\rfloor, w') + T(\lfloor h/2\rfloor, w-w') & \text{if } h \geq 2, \\ O(w\log\log w) & \text{if } h \leq 1. \end{cases}$$

Lemma 2.2 implies that $T(h,w) = O((h+w)\log(h+w)\log\log(h+w))$. Thus, we can solve all recursive subproblems in $O((h+w)\log(h+w)\log\log(h+w))$, and the result follows. ◀

There are incremental algorithms to maintain the convex hull explicitly in amortized time $O(\log n)$ per insertion; see for example [21, Chapter 3]. Such a procedure gives the subroutine needed in Lemma 2.1(a) with $T_{greedy}(n) = O(n\log n)$. An exponential search and Lemma 5.3 can also be used to obtain the same result. Lemma 5.4 gives the subroutine needed in Lemma 2.1(b) with $T_{rect}(n) = O(n\log n)$. Then Lemma 2.1 implies the following.

▶ **Theorem 5.5.** *Let $S = s_1, \ldots, s_n$ be a sequence of planar points. In $O(n\log n\log\log n)$ time, we can compute, for all $i \in [n]$, the largest index $j^*(i)$ such that $CH(\{s_i, \ldots, s_{j^*(i)}\}))$ has area at most 1.*

6 Planar graphs

In this section, we move away from geometry to discuss graph planarity. This problem provides a neat use of the methodology we presented and an important improvement over the use of dynamic data structures. We only provide a very high-level overview.

Let G be a planar graph and let X be a subset of its vertices. A graph H is a ***planar-sketch of G with respect to X*** if it satisfies the following conditions:

- $X \subseteq V(H)$,
- H has size $O(|X|)$, and
- for each edge set F with endpoints in X, $G+F$ is planar if and only if $H+F$ is planar.

Galil, Italiano, and Sarnak [15] have shown that such planar-sketches exist and can be computed in linear time. Note that they defined the sketch property for the addition of a single edge ($|F| = 1$). However, Eppstein et al. [12] noted that the same construction works for multiple edges and referred to the sketches as compressed certificates for planarity.

The fact that planar-sketches can be computed in linear time is parallel to Lemma 4.1(a) in this context. One can prove a statement analogous to Lemma 4.1(b) for planar sketches, as follows. If H is a planar-sketch of G with respect to X, F is a set of edges with endpoints in X, $H+F$ is planar, and H' is a planar-sketch of $H+F$ with respect to $Y \subset X$, then H' is a planar sketch of $G+F$ with respect to Y.

Equipped with linear-time planarity testing [18] and the aforementioned linear-time computation of planar-sketches, we can follow the same methodology as in Section 4.2, shaving off a logarithmic factor. Thus, we obtain the subroutine needed in Lemma 2.1(a) with running time $T_{greedy}(n) = O(n)$, and the subroutine needed in Lemma 2.1(b) with $T_{rect}(n) = O(n\log n)$. Then Lemma 2.1 implies the following.

▶ **Theorem 6.1.** *Let $E = e_1, \ldots, e_n$ be a sequence of edges. In $O(n\log n)$ time, we can compute, for all indices $i \in [n]$, the largest index $j^*(i)$ such that the graph defined by $e_i + \cdots + e_{j^*(i)}$ is planar.* ◀

Acknowledgments. We are grateful to the reviewers for their careful comments.

References

1 Boris Aronov, Anne Driemel, Marc J. van Kreveld, Maarten Löffler, and Frank Staals. Segmentation of trajectories for non-monotone criteria. In *SODA 2013*, pages 1897–1911, 2013.

2 Michael J. Bannister, William E. Devanny, Michael T. Goodrich, and Joe Simons. Windows into geometric events. In *CCCG 2014*, 2014.

3 Michael J. Bannister, Christopher DuBois, David Eppstein, and Padhraic Smyth. Windows into relational events: Data structures for contiguous subsequences of edges. In *SODA 2013*, pages 856–864, 2013.

4 Kevin Buchin, Maike Buchin, Marc van Kreveld, Maarten Löffler, Rodrigo I. Silveira, Carola Wenk, and Lionov Wiratma. Median trajectories. *Algorithmica*, 66(3):595–614, 2013.

5 Kevin Buchin, Maike Buchin, Marc van Kreveld, and Jun Luo. Finding long and similar parts of trajectories. *Comput. Geom.*, 44(9):465–476, 2011.

6 Maike Buchin, Anne Driemel, Marc J. van Kreveld, and Vera Sacristan. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *J. Spatial Information Science*, 3(1):33–63, 2011.

7 Timothy M. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. *J. ACM*, 48(1):1–12, 2001.

8 Timothy M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *J. ACM*, 57(3), 2010.

9 Chen Chen, Hao Su, Qixing Huang, Lin Zhang, and Leonidas Guibas. Pathlet learning for compressing and planning trajectories. In *SIGSPATIAL'13*, pages 392–395, 2013.

10 Giuseppe Di Battista and Roberto Tamassia. On-Line planarity testing. *SIAM J. Comput.*, 25(5):956–997, 1996.

11 David Eppstein. Dynamic Euclidean minimum spanning trees and extrema of binary functions. *Discrete Comput. Geom.*, 13:111–122, 1995.

12 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification. I. Planary testing and minimum spanning trees. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996.

13 David Eppstein, Michael T. Goodrich, and Maarten Löffler. Tracking moving objects with few handovers. In *WADS 2011*, volume 6844 of *LNCS*, pages 362–373. Springer, 2011.

14 Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM J. Comput.*, 40(2):465–492, 2011.

15 Zvi Galil, Giuseppe F. Italiano, and Neil Sarnak. Fully dynamic planarity testing with applications. *J. ACM*, 46(1):28–91, 1999.

16 Joachim Gudmundsson, Jyrki Katajainen, Damian Merrick, Cahya Ong, and Thomas Wolle. Compressing spatio-temporal trajectories. *Comput. Geom.*, 42(9):825–841, 2009.

17 Joachim Gudmundsson, Marc van Kreveld, and Bettina Speckmann. Efficient detection of patterns in 2D trajectories of moving points. *GeoInformatica*, 11(2):195–215, 2007.

18 John Hopcroft and Robert Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

19 Jakub Łącki and Piotr Sankowski. Reachability in graph timelines. In *ITCS 2013*, pages 257–268, 2013.

20 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981.

21 Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

22 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Process. Lett.*, 6(3):80–82, 1977.