



Instance-Optimal Imprecise Convex Hull

Sarita de Berg 

IT University of Copenhagen, Denmark

Ivor van der Hoog 

IT University of Copenhagen, Denmark

Eva Rotenberg 

IT University of Copenhagen, Denmark

Daniel Rutschmann 

IT University Copenhagen, Denmark

Sampson Wong 

University of Copenhagen, Denmark

Abstract

Imprecise measurements of a point set $P = (p_1, \dots, p_n)$ can be modelled by a family of regions $F = (R_1, \dots, R_n)$, where each imprecise region $R_i \in F$ contains a unique point $p_i \in P$. A *retrieval* models an accurate measurement by replacing an imprecise region R_i with its corresponding point p_i .

We construct the convex hull of an imprecise point set in the plane, by determining the cyclic ordering of the convex hull vertices of P as efficiently as possible. Efficiency is interpreted in two ways: (i) minimising the number of retrievals, and (ii) the computation time to determine the set of regions that must be retrieved.

Previous works focused on only one of these two aspects: either minimising retrievals or optimising algorithmic runtime. Our contribution is the first to simultaneously achieve both. Let $r(F, P)$ denote the minimal number of retrievals required by any algorithm to determine the convex hull of P for a given instance (F, P) . For a family F of n constant-complexity polygons, our main result is a reconstruction algorithm that performs $\Theta(r(F, P))$ retrievals in $O(r(F, P) \log^3 n)$ time.

Compared to previous approaches that achieve optimal retrieval counts, we improve the runtime per retrieval from polynomial to polylogarithmic. We extend the generality of previous results to simple k -gons, to pairwise disjoint disks with radii in $[1, k]$, and to unit disks where at most k disks overlap in a single point. Our runtime scales linearly with k .

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases convex hull, imprecise geometry preprocessing model, partial information

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.25

Related Version *Full Version*: <https://arxiv.org/abs/2504.02611>

Funding This research is supported by the Carlsberg Foundation Fellowship CF21-0302 “Graph Algorithms with Geometric Applications” and the VILLUM Foundation grants VIL37507 “Efficient Recomputations for Changeful Problems”.

Ivor van der Hoog: This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 899987.

Sampson Wong: This project has received funding from the European Union’s Skłodowska-Curie Actions Postdoctoral Fellowship grant agreement No 101146276.

1 Introduction

Imprecision is inherent in real-world data. It arises from rounding errors in floating-point computations, inaccuracies in measurement, and delayed sampling in GPS devices. In many scenarios, imprecise data can be refined at a cost by computing exact values or by taking additional samples. This is formalised in the model of *imprecise geometry* introduced by Held and Mitchell [13], and studied further in [8–10, 13, 17, 19, 25].



© Sarita de Berg, Ivor van der Hoog, Eva Rotenberg, Daniel Rutschmann, and Sampson Wong; licensed under Creative Commons License CC-BY 4.0

33rd Annual European Symposium on Algorithms (ESA 2025).

Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman; Article No. 25; pp. 25:1–25:15

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Imprecise geometry. An imprecise point set [13] is defined as a family of regions $F = (R_1, \dots, R_n)$, where each region R_i contains a unique but unknown point p_i . A *realisation* $P \sim F$ is a sequence $P = (p_1, \dots, p_n)$ where $p_i \in R_i$. An input instance is a pair (F, P) . A *retrieval* operation reveals the precise location of a point p_i , replacing R_i with p_i .

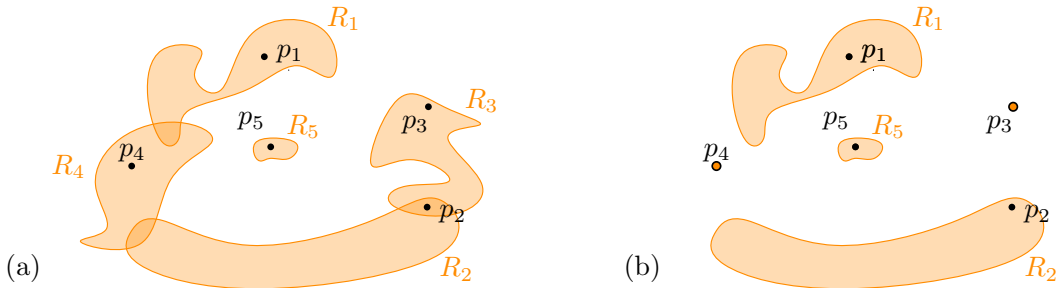
Let $F \odot_P B$ denote the family F after retrieving all $R_i \in B$, where $B \subset F$. The aim of a *reconstruction algorithm* is to identify a subset $B \subset F$ such that for all realisations $P_1, P_2 \sim F \odot_P B$, the algorithm's output (e.g., the cyclic order of the region indices of the points around the convex hull) is identical for P_1 and P_2 . Figure 1 illustrates an example. After retrieving the subset $B = \{R_3, R_4\}$, the cycling ordering of the convex hull vertices is identically (p_1, p_3, p_2, p_4) for all realisations of $F \odot_P B$.

We evaluate reconstruction algorithms by three criteria: the preprocessing time, the total number of retrievals, and the running time per retrieval. Next, we consider instance-optimal algorithms, which minimise the total number of retrievals in the strictest sense.

Worst-case results. Many geometric problems have been studied in imprecise geometry, including Delaunay triangulations [8, 13, 19, 25], convex hulls [9, 10], Gabriel graphs [19], and onion decompositions [17]. For these geometric problems, it is known that any reconstruction algorithm must, in the worst case, use $\Omega(n)$ retrievals. Most of the existing algorithms assume that the regions in F are pairwise disjoint unit disks [8, 9, 13, 17, 19, 25]. Under these assumptions, deterministic worst-case optimal algorithms exist with $O(n \log n)$ preprocessing time, $O(n)$ retrievals, and $O(n)$ reconstruction time. Ezra and Mulzer [10] allow F to consist of arbitrary lines and reconstruct the convex hull using $O(n)$ retrievals and $O(n \cdot \alpha(n))$ expected time where $\alpha(n)$ denotes the inverse Ackermann function. Buchin, Löffler, Morin and Mulzer [3] permit overlapping unit disks of *ply* k (i.e., at most k disks intersect each point in the plane). We summarize these results in the top three rows of Table 1.

Instance-optimality. In many instances, worst case bounds are overly pessimistic. Intuitively, an algorithm is instance-optimal if, for every input (F, P) , no other algorithm performs better on that instance. Constructing instance-optimal algorithms is challenging, as they must match every alternative algorithm, including those tailored for specific instances.

Afshani, Barbay, and Chan [1] proved that, for many geometric problems including constructing the convex hull, instance-optimal reconstruction time is unachievable. Likewise, it is easy to see that instance-optimality in the number of retrievals cannot be guaranteed in general: E.g., let F consist of two overlapping intervals R_1 and R_2 in \mathbb{R} , with $R_1 \setminus R_2 \neq \emptyset$ and $R_2 \setminus R_1 \neq \emptyset$. Define P_1 with $p_1 \in R_1 \setminus R_2$ and $p_2 \in R_1 \cap R_2$, and P_2 with $p_1 \in R_1 \cap R_2$



■ **Figure 1** (a) A family of regions $F = R_1, \dots, R_5$ and a sequence $P = (p_1, \dots, p_5)$ with $P \sim F$. (b) If we retrieve R_3 and R_4 to obtain F' then for all $P' \sim F'$ the convex hull equals (p_1, p_3, p_2, p_4) . Note that if p_3 would lie in $R_2 \cap R_3$ instead, then retrieving only R_3 and R_4 does not suffice.

and $p_2 \in R_2 \setminus R_1$. In (F, P_1) , retrieving R_1 suffices; in (F, P_2) , retrieving R_2 suffices. No algorithm can distinguish between the two instances without making a retrieval, and hence must use two retrievals on one of the instances. Since exact instance-optimality cannot be obtained, we instead aim for *asymptotic* instance-optimality.

Formally, we denote for any input instance by $r(F, P)$ the minimum integer such that there exists a subset $B \subset F$ of size $r(F, P)$ where all realizations $P' \sim (F \odot_P B)$ have the same vertex-ordering along the convex hull of P' . We note that $r(F, P)$ is, equivalently, the optimal number of retrievals. We say that an algorithm is *instance-optimal* if for all inputs (F, P) it uses $\Theta(r(F, P))$ retrievals. We highlight that $r(F, P)$ depends on both the region family F and the specific realisation P . To illustrate, suppose F consists of nested rectangles $R_n \subset R_{n-1} \subset \dots \subset R_1$. If $P \sim F$ is such that the convex hull of (p_1, p_2, p_3, p_4) contains all of R_5 , then retrieving just those four suffices: $r(F, P) = 4$. But if all p_i coincide then $r(F, P) = n$.

Prior instance-optimal work. Only a few instance-optimal reconstruction algorithms are known (see Table 1). Bruce, Hoffmann, Krizanc, and Raman [2] presented the first such algorithm for the convex hull. Their approach is computationally expensive, using an unspecified but superlinear time per retrieval (we discuss this in detail in the full version). Subsequent works show that polylogarithmic time per retrieval is achievable, albeit on geometric structures that are much simpler than the convex hull.

Van der Hoog, Kostitsyna, Löffler, and Speckmann [21] introduced the first near-linear instance-optimal algorithm, solving the sorting problem for one-dimensional intervals. Their algorithm preprocesses the intervals in $O(n \log n)$ time and uses $\Theta(r(F, P))$ retrievals, with at most logarithmic time per retrieval. Later, Van der Hoog, Kostitsyna, Löffler, and Speckmann [22] extended this approach to two-dimensional inputs for Pareto front reconstruction, under the assumption that F consists of pairwise disjoint axis-aligned rectangles. Their method also guarantees at most logarithmic time per retrieval.

Here, we show that polylogarithmic time per retrieval is achievable for reconstructing the convex hull with an instance-optimal number of retrievals. Moreover, our work generalises previous works in that we can handle overlapping regions in two dimensions, whereas previous works could only support one-dimensional inputs or disjoint two dimensional inputs.

Simultaneous work. Simultaneously and independently of this paper, Löffler and Raichel [18] developed an algorithm for reconstructing the convex hull when F is a set of unit disks of ply k . Their number of retrievals (Table 1) is not instance-optimal but rather worst-case optimal for every instance F . Formally, they use $O(w(F))$ retrievals for some region-dependent value $w(F)$ with $r(F, P) \leq w(F) \leq n$. We discuss their result in more detail in the full version.

Instance optimality in other fields. The study of instance-optimal algorithms extends beyond computational geometry. A prime example is sorting under partial information. Given a partial order O over a set X and an unknown linear extension L , the goal is to sort X using a minimum number of comparisons, where a comparison queries the order of a pair $(x, y) \in X$ under L . This is directly analogous to retrievals. Kahn and Saks [16] introduced an exponential-time algorithm that is instance-optimal in the number of comparisons. Since then, significant progress has been made on improving the runtime of such algorithms [5, 15], culminating in near-linear time algorithms that are instance-optimal in both runtime and comparisons [12, 23, 24]. Another example is the bidirectional shortest path problem. Haeupler, Hladík, Rozhoň, Tarjan, and Tětek [11] show an algorithm that finds the shortest path between two nodes s and t using an instance-optimal number of edge-weight comparisons.

■ **Table 1** Previous results that compute a *sorting*, *Pareto front*, *convex hull*, or *Delaunay* triangulation. The bottommost result is simultaneous and independent work from ours.

Shapes	Overlap	Structures	Preprocess	Retrievals	Reconstruction time	Source
Unit disks	No	many	$O(n \log n)$	$O(n)$	$O(n)$	[8, 9, 13, 17, 19, 25]
Disks	k	Delaunay	$O(n \log n)$	$O(n)$	$O(n \log k)$ rand.	[4]
Lines	Yes	hull	$O(n \log n)$	$O(n)$	$(n \cdot \alpha(n))$ rand.	[10]
Intervals	Yes	sorting	$O(n \log n)$	$\Theta(r(F, P))$	$O(r(F, P) \cdot \log n)$	[21]
Axis rect.	No	front	$O(n \log n)$	$\Theta(r(F, P))$	$O(r(F, P) \cdot \log n)$	[22]
Smooth	Yes	front, hull	$O(\text{poly } n)$	$\Theta(r(F, P))$	$O(r(F, P) \cdot \text{poly } n)$	[2]
$O(1)$ -gons	Yes	hull	$O(n \log^3 n)$	$\Theta(r(F, P))$	$O(r(F, P) \cdot \log^3 n)$	Thm 33
k -gons	Yes	hull	$O(kn \log^3 n)$	$\Theta(r(F, P))$	$O(r(F, P) \cdot k \log^3 n)$	Thm 34
$[1, k]$ -disks	No	hull	$O(kn \log^3 n)$	$\Theta(r(F, P))$	$O(r(F, P) \cdot k \log^3 n)$	Full version
Unit disks	k	hull	$O(kn \log^3 n)$	$\Theta(r(F, P))$	$O(r(F, P) \cdot k \log^3 n)$	Full version
Unit disks	k	hull	$O(k^3 n)$	$O(w(F))$	$O(k^3 \cdot w(F))$	[18]

Our contributions. We present the first algorithm for convex hull reconstruction that is instance-optimal while only requiring polylogarithmic time per retrieval. If F is a family of n constant-complexity simple polygons, we preprocess F in $O(n \log^3 n)$ time and reconstruct the convex hull of any $P \sim F$ using $\Theta(r(F, P))$ retrievals and $O(r(F, P) \log^3 n)$ total time. Our approach applies to simple k -gons with a linear factor in k in space and time, to pairwise disjoint disks with radii in $[1, k]$, and to unit disks of ply k (see Table 1). Compared to previous approaches for convex hulls that achieve optimal retrieval counts [2], we exponentially improve the runtime per retrieval from polynomial to polylogarithmic. Compared to near-linear time algorithms for convex hulls (or, Delaunay triangulations) [8–10, 13, 18, 19, 25], we significantly reduce the number of retrievals used and broaden the generality of input families. The latter comes at a cost of a polylogarithmic slowdown if $r(F, P)$ is linear in n .

Organisation. In Section 2, we provide some preliminaries. Then, in Section 3, we present our general algorithm for reconstructing the convex hull and prove its optimality (Theorem 18). In Section 4, we present a data structure for simple k -gons and give an instance-optimal algorithm that uses $O(kn)$ space and $O(r(F, P) \cdot k^2 \log^3(kn))$ time (Theorem 33). In the full version we improve the dependency on k to near-linear (Theorem 34). In the full version, we study pairwise disjoint disks with a radius in $[1, k]$, or, unit disks with ply k .

2 Preliminaries

Recall that an imprecise point set is defined as a family of geometric regions $F = (R_1, \dots, R_n)$, and a realisation $P \sim F$ is defined as a sequence P of n points with $p_i \in R_i$. We explicitly allow for duplicate points in P and do not assume general position. The algorithm has two phases [13]: a *preprocessing phase*, where only F is available, and a *reconstruction phase*, in which we may perform a *retrieval* on any $R_i \in F$, replacing R_i with the point p_i . To maintain generality, we require only that each R_i is a point or a closed, connected, bounded region whose boundary is a simple closed piecewise- C^1 curve. Non-point regions must have connected interiors and coincide with the closure of their interiors; see Figure 2. Next, we define retrievals, convex hulls, vertices, edges, reconstruction strategies, and instance-optimality.

Retrievals. Given a family of regions F , a *retrieval* operation on a non-point region R_i replaces R_i with its associated point p_i (thereby updating F). We write $F \odot_P A$ for the family resulting from retrieving all regions in $A \subseteq F$. We denote by $F - A$ the family obtained by removing all regions in A from F . This removal is used in our analysis when we identify regions whose corresponding points cannot appear on the convex hull.

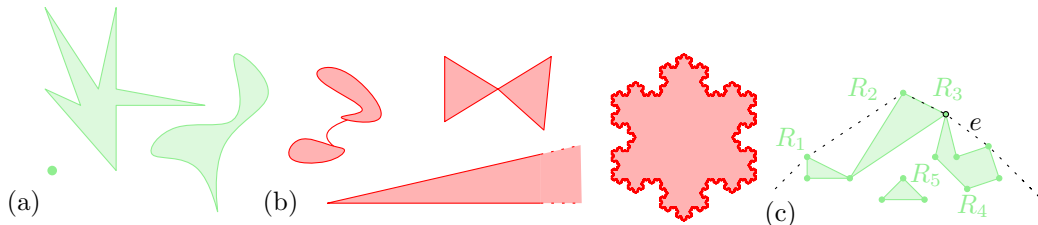
Critical assumptions. Since all regions in F are closed, we claim that we must assume P may contain duplicates and require that the convex hull of a point sequence includes all collinear and duplicate points. To illustrate, let F consist of n identical unit squares, and suppose $P \sim F$ includes four points forming the corners of one square. A lucky algorithm might retrieve these four points first and construct the convex hull [14]. If we assumed general position or excluded duplicates from the convex hull, this algorithm could then conclude that all remaining points cannot lie on the convex hull and terminate early. However, such behaviour cannot be guaranteed against an adversary. Since all regions are indistinguishable, an adversary may simply give these four points last. Alternatively, if F consisted of open regions, we could assume general position and exclude duplicates.

Upper quarter convex hull. We focus on the *upper quarter convex hull* of P . The other three quarters can be constructed analogously, and the full convex hull results from combining them. Let $\text{CH}(P)$ denote the upper quarter convex hull: the boundary of the smallest convex set containing $P \cup \{(-\infty, -\infty), (+\infty, -\infty)\}$. We assume that $(R_{n+1}, R_{n+2}) = (\{(-\infty, -\infty)\}, \{(+\infty, -\infty)\})$, with corresponding points (p_{n+1}, p_{n+2}) . These are always included in F and P . For any family of regions F , we denote by $\text{OCH}(F)$ the *upper quarter outer convex hull*: the boundary of the smallest convex area enclosing all regions in F . Thus, $\text{CH}(P)$ and $\text{OCH}(F)$ represent convex hulls over point sequences and region families, respectively. As convex hulls define boundary curves, we may say that a point $p \in \mathbb{R}^2$ lies *on*, *inside*, or *outside* $\text{CH}(P)$ or $\text{OCH}(F)$.

Vertices and edges. We define convex hull vertices and edges of $\text{OCH}(F)$. Although intuitive in simpler settings, these concepts require care due to the generality of F .

► **Definition 1.** A point $p \in R$ is a vertex of a region R if p lies on the boundary of R and every open line segment through p contains a point not on the boundary of R . For any $R \in F$, $V(R)$ is the (infinite) set of vertices of R .

Intuitively, the edges of $\text{OCH}(F)$ connect successive vertices lying on its boundary. Due to potential overlaps between regions, vertices may coincide. As illustrated in Figure 2(c), a single edge may therefore be considered to connect different pairs of overlapping vertices. To avoid such degeneracy, we define $V(F) = \bigcup_{R \in F} V(R)$ as a set, and define edges robustly:



■ **Figure 2** (a) Regions may have bends and sharp corners, and, be points. (b) Regions may not be unbounded, non-simple, connected by a line, or have infinitely many sharp corners. (c) Vertices between regions may coincide, and point regions (R_3) may coincide with vertices of other regions.

► **Definition 2.** We define an edge (s, t) of $\text{OCH}(F)$ to be a pair of distinct vertices in $V(F)$ where the subcurve of $\text{OCH}(F)$ from s to t contains no vertices in $V(F)$ other than s and t .

Note that a disk has infinitely many vertices but no edges under these definitions. We also define when a region appears on the outer convex hull:

► **Definition 3.** We say a region R appears on $\text{OCH}(F)$ if there is a point p on $\text{OCH}(F)$ with $p \in R$. (Observe that we may always pick p to be a vertex of R .)

Reconstruction algorithms. We will retrieve regions until all realisations $P' \sim F$ yield the same ordering of points along their convex hull. We formalise this via a partial order:

► **Definition 4.** Given $P \sim F$, let $\preceq(\text{CH}(P))$ be the partial order on $[n]$ induced by the left-to-right traversal of $\text{CH}(P)$, i.e. for $p_a, p_b \in P$, we have $a \prec b$ if and only if p_a and p_b both lie on $\text{CH}(P)$ and p_a lies strictly to the left of p_b .

We say a family F is *finished* if all realisations $P_1, P_2 \sim F$ satisfy $\preceq(\text{CH}(P_1)) = \preceq(\text{CH}(P_2))$. A *reconstruction algorithm* retrieves some $B \subseteq F$ such that $F \odot_P B$ is finished. Let $r(F, P)$ denote the minimum number of retrievals needed by any such algorithm.

► **Observation 5.** For any fixed $P \sim F$, let $A \subset F$ be a smallest subset of F such that $F \odot_P A$ is finished. Then $|A|$ is a tight lower bound for $r(F, P)$.

A reconstruction algorithm is *instance-optimal* if for all inputs (F, P) it retrieves $\Theta(r(F, P))$ regions before F is finished. We distinguish two types of reconstruction algorithms:

- A *reconstruction strategy* is any such algorithm, analysed only by the number of retrievals.
- A *reconstruction program* is a reconstruction algorithm executed on a pointer machine or RAM, and is analysed by both retrievals and instructions.

The previous instance-optimal reconstruction strategy. Bruce, Hoffmann, Krizanc, and Raman [2] present an instance-optimal reconstruction strategy for region families F consisting of points and closed piecewise- C^1 regions. They do not present a corresponding reconstruction program. When F consists of disks or polygons, a reconstruction program may be derived, albeit with unspecified polynomial-time costs per retrieval. We further discuss their algorithm in the full version. We rely on a key concept of their paper, i.e. a witness.

► **Definition 6.** Let F be a family of regions and let $A \subseteq F$. Any $A \subset F$ is a *witness* if for all $P' \sim (F - A)$, the family $A \cup P'$ is not finished.

We observe that a reconstruction strategy is instance-optimal if, at each step, it retrieves all regions from a constant-size witness set of F .

2.1 Explicit dynamic planar convex hull

We rely on a data structure to maintain the upper quarter convex hull of a dynamic point set. Overmars and van Leeuwen [20] provide the best-known solution, achieving deterministic update time $O(\log^2 n)$. Their data structure, known as the *Partial Hull Tree* (PHT), is now textbook material. We follow their terminology:

► **Definition 7** (PHT from [20]). Given a two-dimensional point set S , the Partial Hull Tree stores S in a leaf-based balanced binary tree T (sorted by x -coordinates). For each interior node $v \in T$, denote by $\text{CH}(v)$ the (upper quarter) convex hull of all points in the leaves of the subtree rooted at v . For each $v \in T$, with children (x, y) and parent w , the PHT stores:

- The bridge $e(\nu)$ (the unique edge in $CH(\nu)$ that is not in $CH(x)$ and $CH(y)$), and
 - a concatenable queue $\mathbb{E}^*(\nu)$ (a balanced tree of the edges in $CH(\nu) - CH(w)$).
- We denote by $\mathbb{E}(\nu)$ a balanced tree over $CH(\nu)$ and by $V[\nu]$ the vertices in the subtree of ν .

Note that for the root ρ of T , $\mathbb{E}(\rho) = \mathbb{E}^*(\rho)$.

3 Witnesses and an instance-optimal reconstruction strategy

We introduce a new geometric classification over the edges of $OCH(F)$. This leads to an alternative instance-optimal reconstruction strategy, detailed in Algorithm 1. Our results hold in full generality for any family F of points and closed regions whose boundaries are simple, closed, piecewise- C^1 curves. In particular, each region is connected and bounded, and any non-point region has a connected interior equal to its closure; see Figure 2. We define a classification over vertex pairs (s, t) , which we apply to edges of $OCH(F)$. However, we formulate the classification for arbitrary vertex pairs, as this generality will be useful for our data structures. We say that two regions $R_a, R_b \in F$ are *strictly vertically separated* if there exists a vertical line ℓ where R_a and R_b lie in opposite open half-planes defined by ℓ .



■ **Figure 3** Three examples of a band (blue) between regions (green, brown).

► **Definition 8** (Band. Fig. 3). For any pair of regions (R_a, R_b) (where $a = b$ is allowed) we define $\text{band}(R_a, R_b)$ as the area enclosed by the (full) outer convex hull of $\{R_a, R_b\}$.

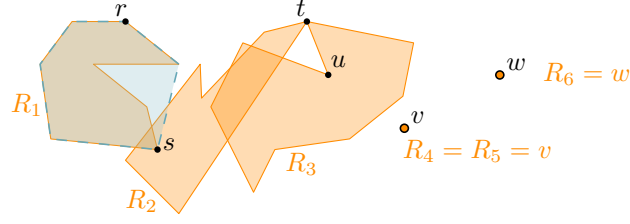
► **Definition 9** (Fig. 4). We say $(s, t) \in V(R_a) \times V(R_b)$ with $s \neq t$ and $R_a, R_b \in F$, is:

- canonical in F if the following holds for all $x \in \{s, t\}$. Either:
 - x is exclusively a vertex of point regions, or
 - x is a vertex of a unique region that is not a point region.
- dividing in F if (s, t) is canonical and either:
 - $a = b$, or
 - R_a, R_b are strictly vertically separated.
- occupied in F if (s, t) is dividing and:
 - $\text{band}(R_a, R_b)$ contains a vertex in $V(F - R_a - R_b) - \{s\} - \{t\}$.

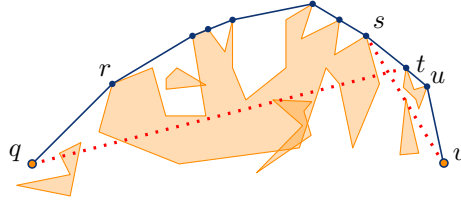
In our reconstruction strategy, we retrieve regions until no non-canonical, non-dividing, or occupied edges remain. This alone does not guarantee that F is finished, so we introduce a final characterisation based on convex chains.

► **Definition 10** (Spanning chain. Fig. 5). A convex chain $C = (q, r, \dots, s, t)$ is spanning in F if the following three conditions hold:

- all edges on C are dividing in F ,
- $q \in V(R_a)$, $r, s \in V(R_b)$, $t \in V(R_c)$ where R_b intersects the inside of $OCH(\{R_a, R_c\})$,
- all vertices of C between and including r and s are in $V(R_b)$.



- **Figure 4** We illustrate all (sub) cases of vertex pairs corresponding to Definition 9:
- All pairs that include t are non-canonical, as t is a vertex of more than one non-point region.
 - All pairs in $\{r, s, u, v, w\}^2$ are all canonical, as v is exclusively a vertex of two point regions.
 - The pair (u, v) is non-dividing, as R_3 and R_4 are not vertically separated.
 - The pairs (s, u) , (r, u) , (r, v) , (s, v) , and any pair in $\{r, s, u, v\} \times \{w\}$ are dividing because their regions are vertically separated. (r, s) is dividing because they are vertices of the same region.
 - The pair (r, s) is not occupied since $\text{band}(R_1, R_1)$ only contains vertices of R_1 . The pair (v, w) is not occupied since $\text{band}(R_4, R_6)$ only contains vertices v and w .
 - The pair (u, w) is occupied as $V(F - R_3 - R_6)$ contains the vertex $t \in R_2$. The pairs (s, u) , (r, u) , (r, v) , (s, v) , (r, w) , and (s, w) are also occupied.



■ **Figure 5** The convex chain from q to t is spanning. The chain from r to v is not spanning.

We use $\text{OCH}(\{R_a, R_c\})$ rather than $\text{band}(R_a, R_c)$ in Definition 10 to ensure meaningful behaviour even when R_a and R_c are point regions (since $\text{band}(R_a, R_c)$ would then be empty). Algorithm 1 processes edges of $\text{OCH}(F)$ by a priority based on Definitions 9 and 10:

non-canonical \gg canonical but non-dividing \gg occupied \gg in spanning chain.

In the following two subsections, we first show that each case in Algorithm 1 retrieves a witness. Then, we prove that if Algorithm 1 is instance-optimal.

3.1 Proving that Algorithm 1 retrieves only witnesses

The first case of Algorithm 1 considers a non-canonical edge, which implies that there is a point on $\text{OCH}(F)$ that lies in two regions that are not both point regions.

► **Lemma 11** (Proof in the full version). *Let $s \in \text{OCH}(F)$ be a point that lies in two regions R_a, R_b that are not both point regions. Then $\{R_a, R_b\}$ is a witness.*

The second case of Algorithm 1 considers an edge (s, t) of $\text{OCH}(F)$ that is canonical but not dividing in F . Let $s \in V(R_a)$ and $t \in V(R_b)$. Because (s, t) is canonical but not dividing it holds that $a \neq b$ and R_a and R_b are distinct regions that are not strictly vertically separated. The following lemma implies that $\{R_a, R_b\}$ is indeed a witness.

► **Lemma 12** (Proof omitted). *Let $R_a, R_b \in F$ with $R_a \neq R_b$ appear on $\text{OCH}(F)$. If R_a and R_b are not strictly vertically separated, then $\{R_a, R_b\}$ is a witness.*

■ **Algorithm 1** A data structure friendly reconstruction strategy: $\text{Reconstruct}(F)$.

```

if  $\exists$  an edge  $(s, t)$  of  $\text{OCH}(F)$  that is non-canonical in  $F$  then
  Let  $s \in V(R_a)$  and  $s \in V(R_b)$  (or  $t$ ) where  $R_a$  is not a point region
  Reconstruct( $F \odot_P \{R_a, R_b\}$ )
else if  $\exists$  an edge  $(s, t)$  of  $\text{OCH}(F)$  that is canonical but not dividing in  $F$  then
  Let  $s \in V(R_a)$  and  $t \in V(R_b)$  for  $a \neq b$ 
  Reconstruct( $F \odot_P \{R_a, R_b\}$ )
else if  $\exists$  an edge  $(s, t)$  of  $\text{OCH}(F)$  that is occupied in  $F$  then
  Let  $s \in V(R_a)$  and  $t \in V(R_b)$  for  $a \neq b$ 
  Let  $R_i$  with  $R_i \neq R_a$  and  $R_i \neq R_b$  be a region with  $V(R_i) \cap \text{band}(R_a, R_b) \neq \emptyset$ 
  Reconstruct( $F \odot_P \{R_a, R_b, R_i\}$ )
else if  $\exists$  a contiguous subchain  $C = (q, r, \dots, s, t)$  of  $\text{OCH}(F)$  spanning in  $F$  then
  Let  $R_a, R_b, R_c$  be as in the definition of spanning chains
  Reconstruct( $F \odot_P \{R_a, R_b, R_c\}$ )
else
  Return  $\text{OCH}(F)$ 
end if

```

The third case of Algorithm 1 considers an occupied edge. The next lemma shows that any choice of region that has a vertex in the occupied band gives rise to a witness.

► **Lemma 13** (Proof omitted). *Let (s, t) be an edge of $\text{OCH}(F)$ that is occupied in F . Let $s \in V(R_a)$ and $t \in V(R_b)$ for $a \neq b$. Let R_i be a region not equal to R_a or R_b with a vertex $q \in V(R_i) \cap \text{band}(R_a, R_b)$. Then $\{R_a, R_b, R_i\}$ is a witness.*

The final case of Algorithm 1 considers any remaining spanning subchains of $\text{OCH}(F)$.

► **Lemma 14** (Proof omitted). *Let (q, r, \dots, s, t) be a contiguous subchain of $\text{OCH}(F)$ that is spanning in F . Let R_a, R_b, R_c be as in Definition 10. Then $\{R_a, R_b, R_c\}$ is a witness.*

3.2 Proving that Algorithm 1 is instance-optimal

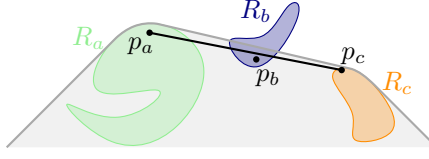
We now prove that Algorithm 1 is instance-optimal. That is, the algorithm retrieves the minimal number of regions (up to constant factors) necessary to determine the ordering of points on the convex hull, for any realisation $P \sim F$. We first observe that Algorithm 1 only terminates when F is *terminal*, meaning: every edge of $\text{OCH}(F)$ is dividing, no edge of $\text{OCH}(F)$ is occupied, and no contiguous subchain of $\text{OCH}(F)$ is spanning.

For the remainder of this section, let A denote the multiset of regions that appear on $\text{OCH}(F)$. If F is terminal, the regions in A satisfy the following structure:

► **Lemma 15.** *If F is terminal, then all regions in A are strictly vertically separated from each other, with the exception of pairs of regions that are duplicates of the same point.*

Proof. Walk along $\text{OCH}(F)$ from left to right, considering edges (s, t) with $s \in R_a$, $t \in R_b$, $a \neq b$. As F is terminal, this edge is dividing, hence R_a lies strictly to the left of R_b . All previously encountered regions lie strictly left of R_a , hence also strictly to the left of R_b . Moreover, this edge is not occupied, so R_b is disjoint from all regions not equal to R_b . ◀

With Lemma 15, we define for each region R that appears on $\text{OCH}(F)$ its left (or right) neighbour as the first region on $\text{OCH}(F)$ that lies *strictly* left (or right) of R . Note that two duplicate point regions on $\text{OCH}(F)$ have the same neighbours. Next, we show that the points corresponding to regions in A always appear on the convex hull (Lemma 16), and that no points corresponding to regions in $F - A$ appear on the convex hull (Lemma 17).



■ **Figure 6** p_b must be above $p_a p_c$, as otherwise there is a spanning contiguous subchain of $\text{OCH}(F)$.

► **Lemma 16.** *If F is terminal, then for any $P' \sim A$ each point in P' lies on $\text{CH}(P')$.*

Proof. Let $P' \sim A$ and $R_b \in A - \{(-\infty, -\infty), (\infty, -\infty)\}$ be arbitrary. Let R_a and R_c be the left and right neighbour of R_b on $\text{OCH}(F)$. Observe that these neighbours are well-defined as $(-\infty, -\infty)$ and $(\infty, -\infty)$ lie on $\text{OCH}(F)$. Suppose by contradiction that p_b lies below the segment $p_a p_c$. Then R_b intersects the inside of $\text{OCH}(\{R_a, R_c\})$, see Figure 6, so there is a contiguous subchain of $\text{OCH}(F)$ that is spanning in F . This contradicts F being terminal. Since R_b was arbitrary, this shows that any three consecutive points in P' form a convex-down angle. Hence, P' spans a convex polygon, so all points in P' appear on $\text{CH}(P')$. ◀

► **Lemma 17.** *If F is terminal, then for any $P \sim F$ only points of regions in A lie on $\text{CH}(P)$.*

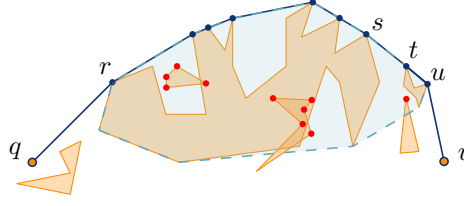
Proof. Let $P' \sim A$ be arbitrary. Suppose by contradiction that there is a region $R_i \in F - A$ with a point $p_i \in R_i$ on or outside of $\text{CH}(P')$. Since the inside of $\text{CH}(P')$ is convex, we may require p_i to be an extreme point of R_i , so $p_i \in V(R_i)$. By Lemma 15 and Lemma 16, there exist consecutive regions $R_a, R_b \in A$ for which p_i lies on or above $p_a p_b$. As $p_a p_b$ lies fully inside $\text{band}(R_a, R_b)$, the point p_i lies in or directly above $\text{band}(R_a, R_b)$. We make a case distinction, where p_i lies directly above R_b (or R_a), or, on the upper tangent of R_a, R_b .

Suppose first that p_i lies directly above a point $q \in R_b$. Let R_a, R_c be the left and right neighbour of R_b on $\text{OCH}(F)$, respectively. Let r be the point on $\text{OCH}(F)$ directly above p_i . Then $r \in \text{band}(R_b, R_c)$ (or $r \in \text{band}(R_a, R_b)$), hence also $p_i \in \text{band}(R_b, R_c)$ as bands are convex and p_i lies on qr . Hence, F is not terminal. Similarly, p_i being directly above a point in R_a implies that F is not terminal. Furthermore, p_i cannot lie above the upper tangent of R_a, R_b since $R_i \notin A$. This shows that p_i does not lie above $\text{band}(R_a, R_b)$, hence $p_i \in \text{band}(R_a, R_b)$, so F is not terminal. ◀

► **Theorem 18.** *Algorithm 1 is an instance-optimal reconstruction strategy.*

Proof. First, we show that when Algorithm 1 terminates, the set F is finished. Let F be terminal. Let $A \subseteq F$ be the multiset of regions that appear on $\text{OCH}(F)$. Let $P' \sim A$ and $Q' \sim F - A$ be arbitrary. By Lemma 17, no points in Q' are on $\text{CH}(P' \cup Q')$. By Lemma 16, the points in P' all appear on $\text{CH}(P' \cup Q')$. Moreover, together with Lemma 15, this implies that for all $P_1 \sim A$ and $P_2 \sim A$, $\preceq(\text{CH}(P_1)) = \preceq(\text{CH}(P_2))$. So, F is finished. Hence, Algorithm 1 is a correct reconstruction strategy.

We now show instance-optimality. By Lemmas 11, 12, 13 and 14, the algorithm retrieves a constant-size witness in each iteration. Let X_1, \dots, X_k be the sets of non-point regions of these witnesses. Since the witnesses have constant size, Algorithm 1 does $O(k)$ retrievals. A retrieval turns a region into a point region, hence the X_i are pairwise disjoint. On the other hand, if $(F \odot_P A)$ is finished for some $A \subseteq F$, then A needs to contain at least one element from each X_i (by the definition of witness). As the X_i are disjoint, this forces $|A| \geq k$. Hence, Algorithm 1 is instance optimal by Observation 5. ◀



■ **Figure 7** The convex chain from q to v is hit in F since (s, t) is occupied due to the red vertices.

4 Regions that are simple k -gons

We present a reconstruction program that executes our strategy from Algorithm 1 in polylogarithmic time per retrieval. To this end, we restrict F to a family of n (possibly overlapping) simple polygons, each with at most k vertices. Recall that $V(F) = (v_1, \dots, v_m)$ denotes the set of all $m \in O(kn)$ distinct vertices in F . After each retrieval, we update F by replacing R_i with p_i , and update V by deleting $O(k)$ vertices from V and adding p_i to V .

► **Observation 19.** For any family of regions F , $\text{OCH}(F)$ has the same edges as $\text{CH}(V(F))$.

Consider a PHT T of $V(F)$ (Definition 7). After one retrieval, let T' be the updated PHT. We define the *recourse set* as the set symmetric difference of all bridges in T and all bridges in T' . *Recourse* is defined as the maximum size, across all possible retrievals, of any recourse set. A PHT has $O(\log m)$ recourse per update, and hence $O(k \log m)$ recourse per retrieval.

Augmenting the Partial Hull Tree. Within each leaf of T , corresponding to a vertex $v \in V(F)$, we maintain two doubly linked lists. The *point region list* stores all point regions that are v . The *non-point region list* stores the other regions $R \in F$ where $v \in V(R)$. Recall that Algorithm 1 retrieves regions corresponding to the endpoint of edges that are:

non-canonical \gg canonical but non-dividing \gg occupied \gg in spanning chain.

For every node $\nu \in T$ with bridge $e(\nu) = (s, t)$, we maintain pointers to the leaves containing s and t , along with Boolean flags indicating whether $e(\nu)$ is canonical or dividing. An update to F may make $O(n)$ edges occupied. So instead, we maintain a different property (Fig. 7):

► **Definition 20.** A convex chain $C = (q, r, \dots, s, t, \dots, u, v)$ of vertices is hit in F if:

- all edges of C are dividing,
- the edge (s, t) is occupied in F ,
- $s \in V(R_a), t \in V(R_b)$ and $a \neq b$, and
- $q \notin V(R_a), r, \dots, s \in V(R_a), t, \dots, u \in V(R_b)$ and $v \notin V(R_b)$.

Recall Definition 7, where for $\nu \in T$, $\mathbb{E}(\nu)$ denotes a balanced binary tree on $\text{CH}(\nu)$. The PHT stores in ν a *concatenable queue* $\mathbb{E}^*(\nu)$ which is $\mathbb{E}(\nu)$ minus all edges that are in $\mathbb{E}(w)$ where w is the parent of ν . We further augment the data structure by maintaining four balanced trees associated with $\mathbb{E}^*(\nu)$, where edges are ordered by their appearance in $\mathbb{E}^*(\nu)$:

1. a tree $\Lambda^*(\nu)$ storing all edges $(s, t) \in \mathbb{E}^*(\nu)$ that are non-canonical in F ,
2. a tree storing all edges $(s, t) \in \mathbb{E}^*(\nu)$ that are canonical and non-dividing in F ,
3. a tree storing all contiguous subchains of $\mathbb{E}^*(\nu)$ that are spanning in F ,
4. a tree storing all contiguous subchains of $\mathbb{E}^*(\nu)$ that are hit in F .

We only give the first tree a name since all others are maintained in similar fashion. We denote by Λ_ν a balanced tree on all non-canonical edges in $\mathbb{E}(\nu)$.

► **Observation 21** (by Definition 9). *For any canonical bridge (s, t) in T , there exists a unique area $R(s)$ that contains s . $R(s)$ is either a point, or a unique region in F and can be found in $O(1)$ time by checking the regions lists stored at one of the leaves that (s, t) points to.*

We store for every region in F their maximum and minimum x -coordinate, this way we can test if a pair of regions is vertically separated in constant time.

Candidate chains. Our data structure maintains for each $\nu \in T$, a collection of special edges and special chains. We observe that these chains have a special structure:

► **Definition 22.** *Let $\nu \in T$ and consider $\mathbb{E}(\nu)$ (not $\mathbb{E}^*(\nu)$). Any contiguous subchain $C = (q, s, \dots, r)$ of $\mathbb{E}(\nu)$ is a candidate chain if all edges of C are dividing in F and:*

- $C = (q, s, r)$ with $q \in V(R_a)$, $s \in V(R_b)$, $r \in V(R_c)$, and $q, r \notin V(R_b)$, or,
- all interior vertices of C are in $V(R_b)$ and $q, r \notin V(R_b)$,
(note that this implies that all interior vertices are not in $V(R_x)$ for $x \neq b$).

► **Observation 23.** *Any subchain of $\mathbb{E}(\nu)$ that is spanning in F is also a candidate chain.*

► **Observation 24.** *Any subchain $(q, r, \dots, s, t, \dots, u, v)$ of $\mathbb{E}(\nu)$ is hit in ν only if (q, r, \dots, s, t) and (s, t, \dots, u, v) are candidate chains.*

► **Lemma 25.** *Let $\nu \in T$. There are at most two candidate chains that contain an edge (s, t) of $\mathbb{E}(\nu)$ and, given $\mathbb{E}(\nu)$ and our data structure, we can find these in $O(k)$ time.*

Proof. Suppose two candidate chains C and C' share (s, t) . By the definition of candidate chains, (s, t) must be the final edge of C and the first edge of C' . First, identify the up to two length-3 chains in $\mathbb{E}(\nu)$ containing (s, t) . Check each such chain in $O(1)$ time for the dividing property by consulting the Boolean flags. If all edges are dividing, we apply Observation 21 to test whether the chain satisfies the conditions of a candidate chain.

Next, consider candidate chains of length ≥ 4 in which (s, t) is an interior edge. Check, in $O(1)$ time via Observation 21, whether both s and t lie in a unique region R_b . If so, perform an in-order traversal of $\mathbb{E}(\nu)$ – moving up to $O(k)$ steps left from s and up to $O(k)$ steps right from t – to find the maximal contiguous chain with all edges dividing and interior vertices contained in $V(R_b)$. The chain ends when we encounter an edge whose endpoints do not lie entirely in $V(R_b)$. This identifies the unique candidate chain C of length ≥ 4 containing (s, t) as an interior edge. To find chains where (s, t) is the first or last edge, apply the above procedure to the edges immediately before and after (s, t) . ◀

Update time. After each retrieval, we update bridges for all nodes along $O(k)$ root-to-leaf paths in T . For each bridge (s, t) , we update its leaf pointers with constant overhead. Corollaries 27, 28, 30, and 32 show how to determine whether a bridge is canonical, dividing, part of a spanning chain, or part of a hit chain in F , respectively, in $O(k \log^2 m)$ time. Since the recourse per retrieval is $O(k \log m)$, this yields an overall update time of $O(k^2 \log^3 m)$.

► **Observation 26** (by Definition 9). *For any node $\nu \in T$, its bridge $e(\nu) = (s, t)$ is canonical in F if and only if both leaves containing s and t meet one of the following conditions:*

- The non-point region list is empty, or,
- The point region list is empty and the region list contains exactly one region.

► **Corollary 27.** *We can dynamically maintain, in $O(k \log^2 m)$ time, for each node $\nu \in T$ a balanced binary tree $\Lambda^*(\nu)$ of all edges $(s, t) \in \mathbb{E}^*(\nu)$ that are non-canonical in F .*

Proof. Test, using Observation 26, whether any bridge is canonical in constant time. Since the recourse is $O(k \log m)$, the set of all bridges can be updated without incurring asymptotic overhead.

To maintain $\Lambda^*(\nu)$ for each node $\nu \in T$, we follow the technique from [20] for concatenable queues. For any node ν with child x , compute $\Lambda(x)$ from $(\Lambda(\nu), e(\nu), \Lambda^*(x))$ in $O(\log m)$ time by splitting $\Lambda(\nu)$ at the bridge $e(\nu)$, and joining the result with $\Lambda^*(x)$. Traverse all $O(k)$ updated root-to-leaf paths, using the split operation, in $O(k \log^2 m)$ total time. Then, traverse the paths bottom-up. At each node ν with children x and y , we have access to the newly updated trees $\Lambda(x)$ and $\Lambda(y)$. Compute $(\Lambda(\nu), \Lambda^*(x), \Lambda^*(y))$ by splitting $\Lambda(x)$ and $\Lambda(y)$ at the endpoints of the new bridge $e(\nu)$ and joining the result. ◀

► **Corollary 28.** *We can dynamically maintain in $O(k \log^2 m)$ time for each node $\nu \in T$ a balanced binary tree on all edges $(s, t) \in \mathbb{E}^*(\nu)$ that are canonical but non-dividing in F .*

Proof. The set of bridges has $O(k \log m)$ recourse. Once a bridge is known to be canonical, we apply Observation 21 to determine the unique areas $R(s)$ and $R(t)$ containing s and t . We then test in constant time whether $R(s)$ and $R(t)$ are vertically separated, using stored x -coordinates. The tree is maintained using the same procedure as in Corollary 27. ◀

► **Lemma 29.** *For any $\nu \in T$, given $e(\nu) = (s, t)$ and $\mathbb{E}(\nu)$, we may find the at most two contiguous subchains C of $\mathbb{E}(\nu)$ with $s, t \in C$ that are spanning in F in $O(k \log m)$ time.*

Proof. Apply Lemma 25 to obtain all $O(1)$ candidate chains in $\mathbb{E}(\nu)$ containing (s, t) . Let $C = (q, \dots, r')$ be such a chain. To test whether C is spanning in F , use Observation 21 to find the unique regions $R(q)$, R_b , and $R(r')$. Construct $\text{OCH}(R(q) \cup R(r'))$ in $O(k \log k)$ time and perform a convex hull intersection test with R_b in $O(\log m)$ time using the algorithm of Chazelle [6]. The chain is spanning if and only if this test returns true. ◀

► **Corollary 30.** *We can dynamically maintain in $O(k^2 \log^2 m)$ time for each node $\nu \in T$ a balanced binary tree of all subchains of $\mathbb{E}^*(\nu)$ that are spanning in F .*

Proof. Whenever the set of leaves of a node $\nu \in T$ changes (which occurs for $O(k \log m)$ nodes), invoke Lemma 29 on $e(\nu)$ to maintain all chains that are spanning in F (and contiguous subchains of $\text{CH}(\nu)$ but not of $\text{CH}(x)$ or $\text{CH}(y)$). The balanced binary tree associated to $\mathbb{E}^*(\nu)$ can be maintained in an identical manner as previous corollaries. ◀

► **Lemma 31.** *For any node $\nu \in T$ and any edge $(x, y) \in \mathbb{E}(\nu)$, we can find all contiguous subchains of $\mathbb{E}(\nu)$ that contain (x, y) and are hit in F in $O(k \log^2 m)$ time.*

Proof. Apply Lemma 25 to find all candidate chains containing (x, y) . Then, for each candidate chain, consider the adjacent edge and apply the lemma again to construct all possible hit chains. For each resulting subchain, test whether the middle edge $i(s, t)$ is occupied as follows: Construct $\text{band}(R(s), R(t))$ in $O(k \log k)$ time [7]. Temporarily remove $R(s)$ and $R(t)$ from F , and delete s and t from $V(F)$, in $O(k \log^2 m)$ time. Update the corresponding PHT T' without updating auxiliary data. The hull $\text{CH}(V(F) \setminus \{s, t\})$ is stored at the root of T' . By intersection testing between $\text{band}(R(s), R(t))$ and $\text{CH}(T')$ in $O(\log m)$ time [6], we can test if any hit chain is occupied. ◀

► **Corollary 32.** *We can dynamically maintain in $O(k^2 \log^3 m)$ time for each node $\nu \in T$ a balanced binary tree of all subchains of $\mathbb{E}^*(\nu)$ that are hit in F .*

Proof. As with Corollary 30, we invoke Lemma 31 on $O(k \log m)$ nodes where the leaf set changes. For each, we update the associated trees using split and join operations. ◀

► **Theorem 33.** *Let F be a family of n simple polygons, where each region in F has at most k vertices. We can preprocess F using $O(kn)$ space and $O(k^2 n \log^3(kn))$ time, such that given $P \sim F$ we reconstruct $\text{CH}(P)$ using $O(kn)$ space and $O(r(F, P)k^2 \log^3(kn))$ time.*

Proof. By Corollaries 27, 28, 30, and 32, we may maintain our augmented Partial Hull Tree in $O(k^2 \log^3(kn))$ time per retrieval. This data structure maintains at its root four balanced trees storing all: non-canonical edges, canonical but non-dividing edges; subchains of $\text{OCH}(F)$ that are spanning in F , or hit in F . We observe that Algorithm 1 only considers occupied edges if all edges on $\text{OCH}(F)$ are dividing. Then, any occupied edge corresponds to a hit subchain. Thus, we may immediately use these trees to execute Algorithm 1. This procedure performs $O(r(F, P))$ retrievals and so the theorem follows. ◀

In the full version, we note that our approach has two bottlenecks. First, the algorithm from Lemma 25 may “skip” over $O(k)$ edges to find the largest candidate chain. By applying a technique similar to skip-lists, we improve its running time by a factor k . Secondly, our approach frequently uses a subroutine where for any two regions $R(s)$ and $R(t)$ we construct $\text{band}(R(s), R(t))$ in $O(k \log k)$ time. We show that by storing for each region R_i its convex hull, this construction time becomes polylogarithmic and we obtain the following:

► **Theorem 34.** *Let F be a family of n simple polygons where each region in F has at most k vertices. We can preprocess F using $O(kn \log n)$ space and $O(kn \log^3(kn))$ time, such that given $P \sim F$ we reconstruct $\text{CH}(P)$ using $O(r(F, P) \cdot k \log^3(kn))$ time.*

References

- 1 Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. Instance-optimal Geometric Algorithms. *Journal of the ACM (JACM)*, 2017.
- 2 Richard Bruce, Michael Hoffmann, Danny Krizanc, and Rajeev Raman. Efficient Update Strategies for Geometric Computing with Uncertainty. *Theory of Computing Systems (TOCS)*, 2005. doi:10.1007/s00224-004-1180-4.
- 3 Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Preprocessing Imprecise Points for Delaunay Triangulation: Simplified and Extended. *Algorithmica*, 2011. doi:10.1007/s00453-010-9430-0.
- 4 Kevin Buchin and Wolfgang Mulzer. Delaunay Triangulations in $O(\text{sort}(n))$ Time and More. *Journal of the ACM (JACM)*, 2011. doi:10.1145/1944345.1944347.
- 5 Jean Cardinal, Samuel Fiorini, Gwenaél Joret, Raphaël Jungers, and J. Ian Munro. Sorting under partial information (without the ellipsoid algorithm). *Combinatorica*, 33(6):655–697, December 2013. doi:10.1007/s00493-013-2821-5.
- 6 Bernard Chazelle and David P Dobkin. Detection is easier than computation. In *ACM Symposium on Theory Of Computing (STOC)*, 1980.
- 7 Mark de Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications, third edition*. Springer, 2008. doi:10.1007/978-3-540-77974-2.
- 8 Olivier Devillers. Delaunay Triangulation of Imprecise Points: Preprocess and Actually get a Fast Query Time. *Journal of Computational Geometry (JoCG)*, 2011. doi:10.20382/jocg.v2i1a3ff.ffinria-00595823.
- 9 William Evans and Jeff Sember. The Possible Hull of Imprecise Points. *Canadian Conference on Computational Geometry (CCCG)*, 2011. URL: <https://www.cs.ubc.ca/~will/papers/possiblehull.pdf>.
- 10 Esther Ezra and Wolfgang Mulzer. Convex Hull of Points Lying on Lines in $o(n \log n)$ Time after Preprocessing. *Computational Geometry*, 2013. doi:10.1016/j.comgeo.2012.03.004.

- 11 Bernhard Haeupler, Richard Hladík, Václav Rozhoň, Robert E Tarjan, and Jakub Tětek. Bidirectional dijkstra's algorithm is instance-optimal. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 202–215. SIAM, 2025. doi:10.1137/1.9781611978315.16.
- 12 Bernhard Haeupler, Richard Hladík, John Iacono, Václav Rozhoň, Robert E. Tarjan, and Jakub Tětek. *Fast and Simple Sorting Using Partial Information*, pages 3953–3973. SIAM, 2025. doi:10.1137/1.9781611978322.134.
- 13 Martin Held and Joseph Mitchell. Triangulating Input-constrained Planar Point Sets. *Information Processing Letters (IPL)*, 2008. doi:10.1016/j.ipl.2008.09.016.
- 14 Simon Kahan. A model for data in motion. In *ACM Symposium on Theory of Computing (STOC)*, pages 265–277, New York, NY, USA, 1991. Association for Computing Machinery. doi:10.1145/103418.103449.
- 15 Jeff Kahn and Jeong Han Kim. Entropy and sorting. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of Computing, STOC '92*, pages 178–187, New York, NY, USA, July 1992. Association for Computing Machinery. doi:10.1145/129712.129731.
- 16 Jeff Kahn and Michael Saks. Balancing poset extensions. *Order*, 1(2):113–126, June 1984. doi:10.1007/BF00565647.
- 17 Maarten Löffler and Wolfgang Mulzer. Unions of Onions: Preprocessing Imprecise Points for Fast Onion Decomposition. *Journal of Computational Geometry (JoCG)*, 2014. doi:10.1007/978-3-642-40104-6_42.
- 18 Maarten Löffler and Benjamin Raichel. Preprocessing disks for convex hulls, revisited. *arXiv preprint arXiv:2502.03633*, 2025. doi:10.48550/arXiv.2502.03633.
- 19 Maarten Löffler and Jack Snoeyink. Delaunay Triangulation of Imprecise Points in Linear Time after Preprocessing. *Computational Geometry*, 2010. doi:10.1016/j.comgeo.2008.12.007.
- 20 Mark H. Overmars and Jan van Leeuwen. Dynamically maintaining configurations in the plane (detailed abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 135–145. ACM, 1980. doi:10.1145/800141.804661.
- 21 Ivor van der Hoog, Irina Kostitsyna, Maarten Löffler, and Bettina Speckmann. Preprocessing Ambiguous Imprecise Points. *International Symposium on Computational Geometry (SoCG)*, 2019. doi:10.4230/LIPIcs.SoCG.2019.42.
- 22 Ivor van der Hoog, Irina Kostitsyna, Maarten Löffler, and Bettina Speckmann. Preprocessing imprecise points for the pareto front. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2022.
- 23 Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. Simpler optimal sorting from a directed acyclic graph. In *Symposium on Simplicity in Algorithms (SOSA)*. SIAM, 2025. doi:10.1137/1.9781611978315.26.
- 24 Ivor Van Der Hoog and Daniel Rutschmann. Tight bounds for sorting under partial information. In *Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2024. doi:10.1109/FOCS61266.2024.00131.
- 25 Marc van Kreveld, Maarten Löffler, and Joseph Mitchell. Preprocessing Imprecise Points and Splitting Triangulations. *SIAM Journal on Computing (SICOMP)*, 2010. doi:10.1137/090753620.