Instance Optimal and Universally Optimal Bounds for Imprecise Pareto Fronts

Sarita de Berg
Department of Computer Science, IT University of Copenhagen, Denmark

Nynne Maria Foldager Bække
Technical University of Denmark, Lyngby, Denmark

Frida Astrup Eriksen
Technical University of Denmark, Lyngby, Denmark

Ivor van der Hoog
IT University of Copenhagen, Denmark

Eva Rotenberg
IT University of Copenhagen, Denmark

Daniel Rutschmann
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Abstract

In the imprecise geometry model, the input is a family of regions $F = (R_1, R_2, \ldots, R_n)$, each containing a point $p_i \in R_i$. The task is then to compute some function of the points $p_1, p_2, \ldots p_n$, in our case an implicit representation of their Pareto front. To this end, one may query a region R_i to *retrieve* its contained point $p_i \in R_i$. In this model, efficiency is interpreted in two ways: minimizing (i) the number of retrievals, and (ii) the computation time both for preprocessing, and the execution of the query stage, i.e. for computing which points to query and constructing the output.

We present an algorithm to construct (an implicit representation of) the Pareto front for possibly overlapping rectangles, that is *instance-optimal* with respect to the number of retrievals. This means that for every fixed input (F, P), there is no algorithm that retrieves asymptotically fewer regions to compute the output. This is a strong algorithmic quality, as it means that our algorithm is competitive even to clairvoyant algorithms which only have to verify the correctness of a correct guess. In terms of algorithmic running time, instance-optimality is provably unobtainable. We instead present an algorithm which is within a $\log n$-factor of instance optimality. This generalizes earlier results which assumed the regions to not overlap, at only a minor cost in running time.

For unit squares, we present an algorithm that is not only instance optimal in the number of retrievals, but also *universally optimal* in terms of running time. This means that for any fixed set of regions F, no algorithm has a better worst-case running time for all possible point sets P. Thus, this work presents the first universally optimal algorithm for overlapping planar input. Compared to previous work, our result improves the degree to which the input regions may overlap, the preprocessing time, the number of retrievals, and the running time.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Pareto front, imprecise geometry, instance optimality, universal optimality, preprocessing model, partial information

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.106

Related Version *Full Version*: https://arxiv.org/abs/2605.07523 [11]

Funding This work was supported by the the VILLUM Foundation grant (VIL37507) "Efficient Recomputations for Changeful Problems".

1 Introduction

Computational geometry has a long tradition of studying data structures on planar point sets whose constructions reduce to sorting. Prominent examples include quadtrees, Euclidean minimu m spanning trees, Delaunay triangulations, convex hulls, and Pareto fronts. Since these problems admit an $\Omega(n \log n)$ lower bound on a Real RAM, classical algorithms construct these structures in $\Theta(n \log n)$ time. A long-standing line of research asks when this $\Omega(n \log n)$ barrier can be circumvented. Under additional assumptions on the input, many algorithms are known that run in $o(n \log n)$ time [2, 6, 8, 12, 14, 17, 18, 21, 26, 28, 32, 33, 34, 36]. These assumptions take several forms: One may assume that the input is already sorted [6, 18, 37] or partially sorted [14, 28]. Alternatively, one may assume integer coordinates, allowing faster-than-comparison-based sorting [17, 21]. A different approach is to exploit structural properties of the input (e.g., by bounding the output complexity) [2, 26, 32].

We consider the case where we are given additional *geometric* information about the input to speed up computation. This assumption is formalized by the now-standard *imprecise geometry* model, introduced by Held and Mitchell [22]. This is a well-studied framework for obtaining faster algorithms for geometric problems [1, 4, 12, 15, 16, 22, 24, 25, 33, 34, 35, 36], where the input is an imprecise point set. An imprecise point set is specified as a family of regions $F = (R_1, \ldots, R_n)$, where for each region R_i one may retrieve the (unknown) true point $p_i \in R_i$, see Figure 1(a). By preprocessing F, we can construct the output on P faster. In this paper, we consider the Pareto front which is the ordered sequence of maximal points, see Figure 1(b). When F is a set of (overlapping) axis-aligned rectangles, we construct the Pareto front of P with an optimal number of retrievals, generalizing the result of van der Hoog, Kostitsyna, Löffler, and Speckmann [25]. Moreover, when the regions are restricted to unit squares, we show that our algorithm achieves universally optimal running time, yielding the first universally optimal algorithm for overlapping regions in $\mathbb{R}^2$.

Imprecise geometry. An imprecise point set [22] is defined as a family of regions $F = (R_1, \ldots, R_n)$, where each region R_i contains a unique but unknown point p_i. A *realization* $P \sim F$ is a sequence $P = (p_1, \ldots, p_n)$ 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 realizations $P_1, P_2 \sim F \odot_P B$, the algorithm's output is the same (in our case, the ordered points on the Pareto front, see Figure 1). We evaluate algorithms by three criteria: the preprocessing time, the total number of retrievals, and the running time of the reconstruction algorithm. We aim for a complexity analysis that goes beyond the (trivial) worst-case scenarios.

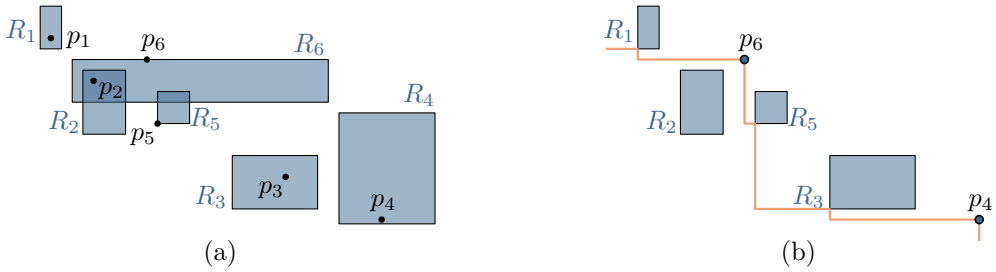

Figure 1 (a) A family of regions $F = (R_1, \ldots, R_6)$ and a sequence $P = (p_1, \ldots, p_6)$ with $P \sim F$. (b) After retrieving R_4 and R_6 the Pareto front equals $(p_1, p_6, p_5, p_3, p_4)$ for *any* $P' \sim F$.

1.1 Worst-case analysis and beyond

Consider an imprecise point set (F, P). For any algorithm $\mathcal{A}$ and input (F, P), we define the *cost* of its execution $\texttt{cost}(\mathcal{A}, F, P)$ to be either its running time $\texttt{runtime}(\mathcal{A}, F, P)$ or the number of retrievals $\texttt{retrievals}(\mathcal{A}, F, P)$ it performs before termination.

In *worst-case* analysis, the worst case cost of an algorithm $\mathcal{A}$ is defined as

$$\texttt{Worst-Case}(\mathcal{A}) := \max_{\text{regions } F} \max_{P \sim F} \texttt{cost}(\mathcal{A}, F, P).$$

For a fixed cost measure, an algorithm $\mathcal{A}$ is *worst-case optimal* if there exists a constant c such that for every algorithm $\mathcal{A}'$, $\texttt{Worst-Case}(\mathcal{A}) \leq c \cdot \texttt{Worst-Case}(\mathcal{A}')$. If the regions in F mutually overlap in some open area A, then F conveys essentially no information about P: any point configuration can be scaled and translated to lie entirely inside A and thus be realized within F. Consequently, for overlapping regions, the worst-case running time of a reconstruction algorithm (for all aforementioned geometric problems) is $\Omega(n \log n)$, and the worst-case number of retrievals is $\Omega(n)$. These bounds are matched by the naive algorithm that ignores F, retrieves all points, and then runs a standard construction algorithm. To obtain meaningful worst-case bounds, much of the literature therefore restricts F to consist of disjoint regions. Under this assumption, researchers have studied the reconstruction of Delaunay triangulations [12, 22, 33, 36], convex hulls [15, 16], Gabriel graphs [36], and onion decompositions [34]. For these problems, any reconstruction algorithm must, in the worst case, perform $\Omega(n)$ retrievals. When the regions in F are pairwise disjoint unit disks [12, 15, 22, 33, 34, 36], the best known worst-case guarantees achieve $O(n \log n)$ preprocessing time, $O(n)$ retrievals and $O(n)$ reconstruction time, which is worst-case optimal. To go beyond such worst-case guarantees, we turn to stricter notions of algorithmic analysis.

Universal optimality. Universal optimality is based on the observation that a running-time analysis can often be strengthened by turning the outer maximum into a universal quantifier. For any fixed set of regions F, we define the *universal cost* of an algorithm $\mathcal{A}$ as

$$\texttt{Universal}(\mathcal{A}, F) := \max_{P \sim F} \texttt{cost}(\mathcal{A}, F, P).$$

For a fixed cost measure, an algorithm $\mathcal{A}$ is *universally optimal* if there exists a constant c such that, for *all sets of regions* F and all algorithms $\mathcal{A}'$, $\texttt{Universal}(\mathcal{A}, F) \leq c \cdot \texttt{Universal}(\mathcal{A}', F)$. For a fixed F, the value $\min_{\mathcal{A}} \texttt{Universal}(\mathcal{A}, F)$ may range from $O(1)$ to $O(n \log n)$. E.g., there exist inputs F where all realizations $P \sim F$ yield the same output and no retrieval nor reconstruction running time is needed. Universal optimality is considerably stronger than worst-case optimality: $\mathcal{A}$ must be efficient for all F, yet for each fixed F it competes with algorithms tailored to F. Consequently, universally optimality is difficult to obtain.

Universal optimality has gained considerable recent attention [2, 7, 13, 19, 20, 23, 24, 25, 26, 27, 28, 29, 30, 31]. For imprecise geometry, only a few results are known. Bruce, Hoffmann, Krizanc, and Raman [5] give a polynomial-time algorithm with a universally optimal number of retrievals for very general F. Löffler and Raichel [35] obtain universal optimal number of retrievals and running time for convex hulls when F consists of unit disks of constant overlap (ply). Their results apply to the Pareto front also. De Berg et al. [4] achieve universally optimal retrievals for convex hulls where F contains overlapping $O(1)$-gons, with reconstruction running time within an $O(\log^3 n)$ factor of universal optimality. Van der Hoog, Kostitsyna, Löffler, and Speckmann [24] give a universally optimal algorithm if F contains one-dimensional intervals and the goal is to sort P. Follow up work [25] extends this to the two-dimensional Pareto front where F consists of n disjoint axis-aligned rectangles.

Instance-optimality. Instance-optimality is an even stronger notion that replaces all maxima by universal quantifiers. For a fixed cost measure, instance optimality of an algorithm $\mathcal{A}$ requires a constant c such that, for all imprecise point sets (F, P) and all algorithms $\mathcal{A}'$, $\texttt{Cost}(\mathcal{A}, F, P) \leq c \cdot \texttt{Cost}(\mathcal{A}', F, P)$. This is an extremely strong requirement, as $\mathcal{A}$ must compete even with algorithms $\mathcal{A}'$ that have (deterministically) guessed the correct output and only verify their guess. For the aforementioned geometric problems, instance-optimal running times are impossible. A representative example is the Pareto front with overlapping unit squares F: Consider the family of algorithms $\mathcal{A}_\pi$ indexed by permutations π, and the family of inputs P_π whose Pareto front equals P in the order π. Given (F, P), the algorithm $\mathcal{A}_\pi$ retrieves all points and verifies in linear time whether its guess π is correct, that is, whether P forms a staircase in the order π; otherwise it constructs the Pareto front in $O(n^2)$ time. For each input P_π, there exists an algorithm $\mathcal{A}_i$ which it runs in linear time, yet no single algorithm can run in linear time for all P_π. Thus, although every $\mathcal{A}_\pi$ is individually inefficient in the worst case, they together make instance-optimal running times unattainable.

Perhaps surprisingly, instance optimality *is* achievable when the cost measure is the number of retrievals. We therefore call an algorithm *instance-optimal* if, for some constant c, for all inputs, it uses at most a factor c more retrievals than any other algorithm. The retrievals performed by Bruce, Hoffmann, Krizanc, and Raman [5] are not only universally optimal, they are instance-optimal. Similarly, the sorting algorithm of van der Hoog, Kostitsyna, Löffler, and Speckmann [24], as well as their algorithm for computing the Pareto front of a point set when F consists of disjoint axis-aligned rectangles [25], are also instance-optimal. Finally, de Berg et al. [4] give an instance-optimal algorithm for constructing the convex hull when F consists of overlapping $O(1)$-gons. A summary is given in Table 1.

Table 1 Results that compute a *sorting*, Pareto *front*, or convex *hull*. For reference, our cost measures have $O(\texttt{instance}) \subseteq O(\texttt{universal}) \subseteq O(n \log n)$. Green arrows indicate that we make an improvement over the state-of-the-art.

Shapes	Overlap	Structures	Preprocess	Retrievals	Reconstruction time	Source
Unit disks	No	many	$O(n \log n)$	$O(n)$	$O(n)$	[12, 15, 22, 34, 36, 33]
Smooth	Yes	front, hull	$O(\text{poly}\, n)$	$O(\texttt{instance})$	$O(\text{poly}\, n)$	[5]
Intervals	Yes	sorting	$O(n \log n)$	$O(\texttt{instance})$	$O(\texttt{universal})$	[24]
Unit disks	k	hull	$O(k^3 n)$	$O(\texttt{universal})$	$O(\texttt{universal} \cdot k^3)$	[35]
Unit disks	k	hull	$O(kn \log^3 n)$	$O(\texttt{instance})$	$O(\texttt{instance} \cdot k \log^3 n)$	[4]
k-gons	Yes	hull	$O(kn \log^3 n)$	$O(\texttt{instance})$	$O(\texttt{instance} \cdot k \log^3 n)$	[4]
Axis rect.	No	front	$O(n \log n)$	$O(\texttt{instance})$	$O(\texttt{universal})$	[25]
Axis rect.	Yes	front	$O(n \log n)$	$O(\texttt{instance})$	$O(\texttt{instance} \cdot \log n)$	Thm. 21
Unit disks	k	front	$O(k^3 n \log n)$	$O(\texttt{universal})$	$O(\texttt{universal} \cdot k^3)$	[35]
Unit squa.	Yes	front	$O(n \log n)$	$O(\texttt{instance})$	$O(\texttt{universal})$	Thm. 22

1.2 Our contributions

We present an instance-optimal Pareto front algorithm for overlapping rectangles. Its running time is within a $\log n$ factor of instance (and thus universal) optimality. This generalizes the result of [24] to overlapping regions, at only a minor running time cost. For unit squares, we achieve the first universally optimal algorithm for overlapping planar input. Compared to the result of [35] for unit disks of ply k, which can also compute the Pareto front, we improve the degree of overlap, the preprocessing time, the number of retrievals, and the reconstruction running time. Table 1 summarizes our results.

To obtain these results, we present a general instance-optimal reconstruction strategy (Section 3), and then present an algorithm to execute the strategy efficiently for rectangles (Section 5). The algorithm for unit squares can be found in the full version [11].

Advantages of our strategy. Our objective is to achieve, after preprocessing, a reconstruction running time that is within a $\log n$ factor of instance optimality. This goal is nontrivial, since this quantity depends on the unknown point set P and therefore cannot be determined in preprocessing. Existing instance-optimal reconstruction strategies are typically adaptive to the *current* set of regions F [5, 25]. These approaches iteratively attempt to identify, based on the information currently available about P, those regions in F that must be retrieved by any correct algorithm. Performing this identification on-the-fly is costly: prior techniques require either polynomial time [5], time polynomial in the overlap [35] (for convex hulls), or restrict F to disjoint regions [4, 25] (for convex hulls or the Pareto front).

In contrast, our strategy relies almost entirely on the *original* regions. During reconstruction, the algorithm only checks whether a region had some dependency property among the original regions and whether it currently lies on the Pareto front. This design shifts much of the algorithmic intricacy to the preprocessing phase. As a result, the reconstruction phase is substantially simpler and uses logarithmic time per retrieval even for overlapping regions.

2 Preliminaries

The algorithmic input is an *imprecise point set*, defined as a family of geometric regions $F = (R_1, \ldots, R_n)$ and a realization $P \sim F$. A realization $P \sim F$ is defined as a sequence P of n points such that $p_i \in R_i$. One is allowed to preprocess F using polynomial time and space. Afterwards, a *reconstruction algorithm* has access to the retrieve operation which replaces a region R_i by its corresponding point p_i. We write $F \odot_P A$ for the family that is obtained by retrieving all regions in $A \subseteq F$. We denote by F^0 the original family of regions before any retrievals. We denote for a family of regions F by $F - A$ the family obtained by removing all regions in A from F.

Pareto front and outer Pareto front. We say that a point p *dominates* q if p lies in the open top-right quadrant of q. The Pareto front $\mathsf{PF}(P)$ are all points of P that are not dominated by some $q \in P$, ordered along the staircase bounding the area of all points $p' \in \mathbb{R}^2$ that are not dominated by some point $q \in P$. Note that, if P does not lie in general position, then this ordering is a partial order. We define for a family of regions F the *outer Pareto front* $\mathrm{OPF}(F)$ as all regions $R_a \in F$ such that there is a point $p \in R_a$ that is not dominated by any point $q \in R_b$ with $a \neq b$ and $R_b \in F$, see Figure 2. If F consists of rectangles, then the outer Pareto front corresponds to the Pareto front of the top-right corners of the regions in F. We define the *inner Pareto front* $\mathrm{IPF}(F)$ as the area of points $p' \in \mathbb{R}^2$ such that for all $P \sim F$, p' is dominated. Note that this is an open area and that if F consists of rectangles, the inner Pareto front corresponds to the area below the staircase of the Pareto front of the bottom-left corners of the regions in F. If some $R_i \in F^0$ is contained within $\mathrm{IPF}(F^0)$ then p_i will never appear on $\mathsf{PF}(P)$. Since we can detect such regions R_i in $O(n \log n)$ time by constructing a Pareto front and doing intersection queries [3], and we use at least $O(n \log n)$ preprocessing time, we henceforth assume that the input F^0 contains no such regions.

Restricting F and P. We consider two scenarios. In our most general scenario, F is a family of (possibly overlapping) closed axis-aligned rectangles. In the other scenario, F is a family of (possibly overlapping) unit squares. We note that, to allow for instance-optimality, we *cannot*

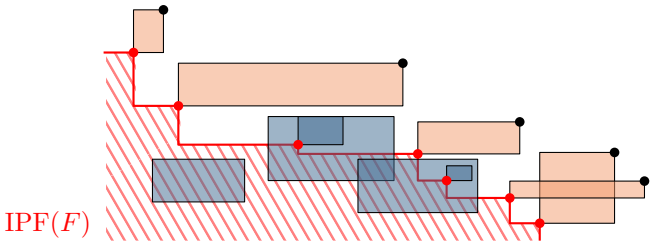

Figure 2 The outer Pareto front OPF(F) consists of the orange regions and corresponds to the Pareto front of the top-right corners (black points). The shaded red area is the inner Pareto front IPF(F): the area below the staircase of the Pareto front of the bottom-left corners (red points).

assume that P lies in general position, i.e. that all points have distinct x- and y-coordinates. Let $F = (R_1, \ldots, R_n)$ be n identical squares. Consider algorithms $\mathcal{A}_i$ where $\mathcal{A}_i$ retrieves p_i first, and inputs $P_i \sim F$ where p_i lies on the top-right corner of R_i. For (F, P_i), $\mathcal{A}_i$ terminates in constant time as per general position, p_i dominates all other points. Yet there exists for any algorithm $\mathcal{A}$ an input (F, P_i) on which $\mathcal{A}$ does $\Omega(n)$ retrievals before finding p_i.

Dominating regions. Recall that by F^0 we denote the original input regions. If we retrieve a region R_i then we replace R_i by p_i, updating the set of regions. We denote at all times by F the "current" set of regions. We subsequently define:

▶ **Definition 1.** *Let F be the current set of regions, and let $R_a \in F$. Then R_a is:*
- always *if for all $P \sim F$ we have that $p_a \in \mathsf{PF}(P)$,*
- dominated *if for all $P \sim F$ we have that $p_a \notin \mathsf{PF}(P)$.*

A point p *dominates* a region R_a if R_a is contained in the bottom-left open quadrant of p.

2.1 Reconstruction algorithms and instance optimality

After preprocessing F^0, a *reconstruction algorithm* can retrieve points to construct the Pareto front of P. We are interested in the combinatorial structure, not in the points of the front itself. For example, if the input is a family F where all $P \sim F$ have the same Pareto front, then we do not wish to retrieve any points. We formalize this, by defining a Pareto front as a partial order and stopping the algorithm when the partial order is the same for all $P \sim F$.

▶ **Definition 2.** *Given $P \sim F$, let $\prec(\mathsf{PF}(P))$ be the partial order on $[n]$ induced by traversing $\mathsf{PF}(P)$ along the corresponding staircase. 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 $\mathsf{PF}(P)$ and either p_a lies strictly to the left or strictly above of p_b.*

A family of regions F is *finished* if for any $P_1, P_2 \sim F$ we have that $\prec(\mathsf{PF}(P_1)) = \prec(\mathsf{PF}(P_2))$. A reconstruction algorithm retrieves some $A \subseteq F$ such that $F \odot_P A$ is finished. We denote by $r(F, P)$ the minimum number of retrievals needed by any algorithm such that the resulting family of regions is finished. We distinguish between two types of reconstruction algorithms: A *reconstruction strategy* is analyzed only by the number of retrievals made. A *reconstruction program* is executed on a RAM, and produces a pointer structure representing $\mathsf{PF}(P)$. Reconstruction programs are analyzed by both the number of retrievals and instructions.

3 A simple instance-optimal reconstruction strategy

Prior instance-optimal reconstruction strategies [4, 5, 24, 25] all use the same technique, introduced in [5]. Let F be the current family of regions and $A \subseteq F$. We say A is a *witness* if for all $P' \sim (F - A)$ the family $A \cup P'$ is not finished. Any strategy that iteratively retrieves

all regions in a constant size witness is instance-optimal [5]. For the Pareto front (and convex hulls) there always exists a witness of size 3 and prior reconstruction strategies focus on finding such a witness set for the current family of regions. In this paper, we propose a simpler strategy with a different proof for instance optimality. To this end, we define a dependency relation between regions (see Figure 3):

▶ **Definition 3.** *A region $R_a \in F_0$ has an* outgoing dependency *in F_0 to a region $R_b \in F_0 - R_a$ if there is a point $q \in R_b$, such that all of the following conditions hold:*

- **I** *the top-right corner of R_b does not dominate R_a, and*
- **II** *q is above the inner Pareto front of F_0, and*
- **III** *either $q \in R_a$, or, the top-right corner of R_a dominates q.*

▶ **Definition 4.** *A region $R_a \in F_0$ is* independent *if it has no outgoing dependency in F^0. We denote by $I^0 \subseteq F^0$ the regions that are independent in F^0. We initially set $I \leftarrow I^0$ and, whenever we retrieve $p \sim R \in I^0$, we replace the region R in I (but not I^0) by p.*

The following lemma implies that we can essentially handle the independent regions separately after dealing with the dependent regions.

▶ **Lemma 5.** *Let F be the current family of regions. Retrieving a region $R_a \in I$ will not cause any non-point region in F to appear on the outer Pareto front.*

Proof. Suppose for contradiction that a region $R_b \in F$ appears on the outer Pareto front after retrieving R_a. As R_b was not on the outer Pareto front before retrieving R_a, there must be a region in F whose top-right corner strictly dominates the top-right corner r of R_b. In particular, this region must be equal to R_a, as otherwise R_b would still not appear on the outer Pareto front after retrieving R_a. Furthermore, r must be above the inner Pareto front of F^0, as it would otherwise never appear on the outer Pareto front. However, the point r then implies that R_a has an outgoing dependency, contradicting $R_a \in I$. ◀

A simple strategy. Our reconstruction strategy (Algorithm 1) is a simple three-stage process. In preprocessing, we compute I^0 and set $I = I^0$ and $F = F^0 - I^0$. Recall that F is updated during the reconstruction phase, while F^0 is fixed. In the first stage, while there exists a region $R_a \in \mathrm{OPF}(F)$ that has an outgoing dependency *in the original input* F^0, we retrieve R_a. In the second and third stage, we consider all regions $R_b \in I$ and retrieve them if it is not yet clear whether they will appear on the Pareto front (Stage 2), or if it is not yet clear *where* they will appear on the Pareto front (Stage 3). We first prove correctness:

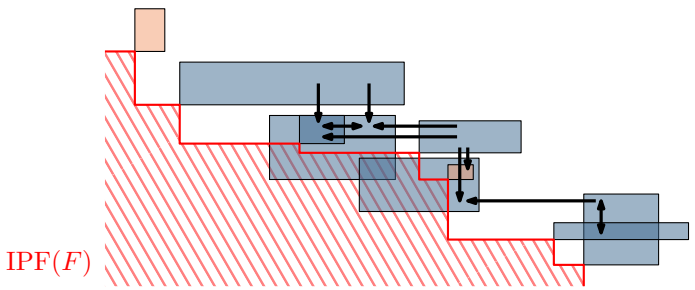

Figure 3 The dependencies between the regions in F_0: an outgoing arrow represents an outgoing dependency. The two orange regions have no outgoing dependencies are are thus *independent* regions.

Algorithm 1 Our simple instance-optimal reconstruction strategy.

```
Preprocessing: F = F^0 − I^0 and I = I^0
while there exists a non-point region R_a ∈ OPF(F) do                ◁Stage 1
    Retrieve R_a.
for all R_a ∈ I do                                                   ◁Stage 2
    if the interior (or the top or right facet) of R_a intersects the staircase OPF(F) then
        Retrieve R_a.
for all vertices p_i ∈ OPF(F) do                                     ◁Stage 3
    Retrieve the R_a ∈ I hit by the upward ray from p_i, unless it hits the bottom-right.
    Retrieve the R_b ∈ I hit by the rightward ray from p_i, unless it hits the top-left.
```

▶ **Lemma 6.** *Let F be the family of regions after the while loop of Algorithm 1, then all regions in F are either retrieved or dominated in F.*

Proof. Suppose for contradiction that there is a region $R_a \in F$ (then $R_a \notin I$) that has not been retrieved and is not dominated. Then R_a is not on $\mathrm{OPF}(F)$, as otherwise it would have been retrieved. Thus there is a region $R_b \in \mathrm{OPF}(F)$ such that R_a is dominated by the top-right corner of R_b. If R_b is a point region, i.e. $R_b = p_b$, then R_a is dominated, a contradiction. Otherwise, either there is an outgoing dependency from R_b to R_a, contradicting that the while loop of Algorithm 1 has terminated, or any point $q \in R_a$ dominated by the top-right corner of R_b must be below the inner Pareto front of F^0. As the top-right corner of R_b dominates R_a, it follows that R_a is in the inner Pareto front and thus dominated. ◀

▶ **Lemma 7.** *The family of regions F^* at the termination of Algorithm 1 is finished.*

Proof. Let $F = F^* - I$. By Lemma 6, $\mathrm{OPF}(F)$ consists of only point regions. Let PF^* denote the Pareto front of these points. We first show that every region in F^* is either always or dominated, and then prove that the order along the Pareto front is fixed.

First, consider a (retrieved) region $R_a = p_a$ on $\mathrm{OPF}(F)$. Assume for contradiction that p_a is neither always nor dominated. Then there exists a $R_b \in I$ such that the top-right corner of R_b dominates p_a and the bottom-left corner of R_b does not dominate p_a. This implies that the upward or rightward ray from p_a hits R_b, and thus R_b is retrieved by Algorithm 1. There can be at most one such region in each direction, as otherwise at least one of these regions would have an outgoing dependency. Thus, all regions in F are either always or dominated.

Next, consider a region $R_c \in I$. When the algorithm terminates, any region in I that intersects the staircase of PF^* in its interior, or top or right facet, has been retrieved. Thus, R_c is either above or strictly below the staircase of PF^*. If R_c is strictly below PF^*, then it is dominated. If R_c is above PF^*, then the point p_c could be dominated only by a point in a region in I. Lemma 5 thus implies that R_c is always (we assumed that none of the regions in F^0 are dominated in F^0). We conclude that all regions in F^* are always or dominated.

What remains is to prove that the order of the always regions in F^* along the Pareto front is fixed, i.e. for any $P_1, P_2 \sim F^*$, we have that $\prec(\mathsf{PF}(P_1)) = \prec(\mathsf{PF}(P_2))$. Suppose there are $P_1, P_2 \sim F^*$, such that $\prec(\mathsf{PF}(P_1)) \neq \prec(\mathsf{PF}(P_2))$. Then there are two regions R_a and R_b such that $a \prec b$ for P_1 but $a \not\prec b$ for P_2. Note that at least one of R_a and R_b must be in I, as otherwise Lemma 6 implies that both R_a and R_b have been retrieved in F^*. Without loss of generality, assume that $R_b \in I$. If $R_a \in I$ also then the fact that there are no outgoing dependency between the two regions and both of them begin always, implies that for any $s \in R_a$ and $t \in R_b$ we have $s \prec t$, as $a \prec b$ for P_1. Contradicting that $a \not\prec b$ for P_2. If $R_a \notin I$,

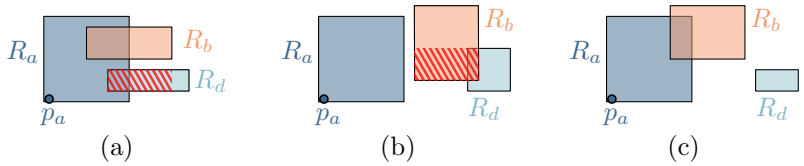

Figure 4 If the shaded red area lies in the inner Pareto front, then there is no outgoing dependency from R_d to R_a. In (c) retrieving R_a will make both R_b and R_d semi-independent.

then R_a is retrieved. In this case, $a \prec b$ for P_1 but $a \not\prec b$ for P_2 implies that $p_a \in R_b$, or the top-right corner of R_b dominates p_a and the bottom-left corner of R_b does not dominate p_a. As before, we conclude that R_b is retrieved, a contradiction. ◀

3.1 Instance-optimality

In this section we prove that Algorithm 1 is an instance-optimal retrieval strategy, i.e. the strategy retrieves $\Theta(r(F^0, P))$ regions. To this end, we call a region $R \in F$ *semi-independent* if R has not been retrieved and R had no outgoing dependencies in $F \cup I$ (see Definition 3).

▶ **Lemma 8.** *Retrieving a region R_a causes at most 4 regions to become semi-independent.*

Proof. Consider a region $R_b \in F - R_a$ that becomes a semi-independent region in $F \odot_P \{R_a\}$. Then R_b not being semi-independent before implies that there is a region R_c, $c \neq b$, that R_b had an outgoing dependency to, and after retrieving R_a one of the conditions I, II, or III fails. We consider two cases: either the retrieved point p_a is in the inner Pareto front of F or not.

Case 1: p_a in IPF(F). As p_a is in IPF(F), retrieving R_a does not change IPF(F). Considering conditions I, II, or III, it must be that $c = a$ and specifically that there is an outgoing dependency from R_b to R_a in F and this is the only outgoing dependency for R_b. We will show that there are at most four regions that become semi-independent by the retrieval of R_a because the outgoing dependency to R_a is removed.

First, note that any region with an outgoing dependency to R_a is by definition not dominated by the top-right corner of R_a, and thus either intersects the top or right boundary of R_a, or is fully above or right of R_a. Next, we show that there is at most one region that intersects the right boundary of R_a and at most one region fully right of R_a that become semi-independent by retrieving R_a. Figure 4(c) gives an example where both of these regions exist. By symmetry, there is also at most one region that intersects the top boundary of R_a and at most one region fully above R_a that become semi-independent by retrieving R_a.

Suppose there are two such regions R_b and R_d that intersect the right boundary of R_a, see Figure 4(a). Without loss of generality, assume that the top of R_b is at least as high as the top of R_d. Then R_b being semi-independent after retrieving R_a implies that there is no outgoing dependency from R_b to R_d. Thus, all points in R_d that are dominated by the top-right corner of R_b, or intersect R_b, (the shaded area in Figure 4(a)) must be inside the (original) inner Pareto front IPF(F^0). However, that means that all points in R_a that are dominated by the top-right corner of R_d must also lie inside the original inner Pareto front, contradicting that R_d has an outgoing dependency to R_a.

Next, suppose there are two such regions R_b and R_d that are fully right of R_a, see Figure 4(b). Again, assume without loss of generality that the top of R_b is at least as high as the top of R_d. By definition of the inner Pareto front, no point dominated by the bottom-left corner of R_b is above the inner Pareto front. Thus, if R_d is fully below R_b, then there is no

point in R_a that is above the inner Pareto front that is dominated by the top-right corner of R_d, contradiction that R_d has an outgoing dependency to R_a. If R_d is not fully below R_b, and the top-right corner of R_b dominates R_d, then there being no outgoing dependency from R_b to R_d implies that R_b is inside the original inner Pareto front. This contradicts our assumption that F^0 contains no dominated regions. If R_d is not fully below R_b, and the top-right corner of R_b does not dominate R_d, then all points in R_b dominated by the top-right corner of R_d must be inside the original inner Pareto front, see Figure 4(b). However, then all points in R_a that are dominated the top-right corner of R_d must also be inside the inner Pareto front, contradicting that R_d has an outgoing dependency to R_a.

Case 2: p_a above IPF(F). Because p_a is above IPF(F), a region R_b can only lose its outgoing dependency to another region R_c with $c \neq a$. In particular, it must be that condition II fails for R_c after retrieving R_a. This means that any point $q \in R_c$ that is dominated by the top-right corner of R_b and is above the inner Pareto front of F, must be dominated by p_a. The proof of case 1 actually implies that there are at most two regions that have their top-right corner right or above of p_a, and have outgoing dependencies only to points that are dominated by p_a. As in case 1, it thus follows there are at most four regions that become semi-independent by the retrieval of R_a. ◀

▶ **Lemma 9.** *For any $R_a \in I$ that is retrieved by Algorithm 1, any algorithm that does* not *retrieve R_a is not finished.*

Proof. We first consider Algorithm 1. Let F be the family of regions (excluding I) when R_a is retrieved and let $P' \sim (F \cup I) - R_a$ be arbitrary. Note that no point in R_a is dominated by any point in another region in I, as that would imply that R_a was dominated in F^0. We consider why R_a was retrieved.

First, assume R_a is retrieved because the interior or the top or bottom facet of R_a intersects OPF(F). Let $p \in R_a$ be a point on the staircase of OPF(F) and let $q \in R_a$ be a point below the staircase of OPF(F). Then setting $p_a = p$ results in p_a being on $\mathsf{PF}(P' \cup \{p_a\})$, whereas setting $p_a = q$ results in p_a not being on $\mathsf{PF}(P' \cup \{p_a\})$.

Second, assume that R_a is retrieved because the upwards ray from a vertex $p_i \in \text{OPF}(F)$ hits R_a not in its bottom-right corner. The case for being hit by a rightward ray is symmetric. Let p be the the point where this ray hits R_a, let q be the top-right corner of R_a. Setting $p_a = p$ will result in both p_a and p_i being on $\mathsf{PF}(P' \cup \{p_a\})$. Setting $p_a = q$ instead, will result in $p_i \notin \mathsf{PF}(P' \cup \{p_a\})$.

In both cases, we constructed two point sets $P' \cup \{q\}$ and $P' \cup \{p\}$ that only differ in the point p_a, but have different Pareto fronts. Thus, the algorithm $\mathcal{A}'$ that retrieves all $R \in (F^0 - R_a)$ is not finished. Any algorithm that does not retrieve R_a retrieves a subset of the regions retrieved by $\mathcal{A}'$ and is therefore also not finished. ◀

▶ **Theorem 10.** *Algorithm 1 is an instance-optimal retrieval strategy.*

Proof. Let (F^0, P) be the input. Fix an algorithm OPT that retrieves $r(F^0, P)$ regions. Let S denote the family of regions retrieved by OPT, and let F^{OPT} be the family of regions after retrieving all of S, i.e. $F^{OPT} := F^0 \odot_P S$. Furthermore, let T be the family of regions that become semi-independent during the execution of OPT. Lemma 8 implies that $|T| \leq 4|S|$. Any region R_a retrieved by our algorithm is either in $S \cup T$, or in at least one the of following cases: (1) R_a is dominated in F^{OPT}, (2) $R_a \in I$, or (3) R_a has an outgoing dependency in F^{OPT}. Next, we show that each of these cases leads to a contradiction. Thus, any region retrieved by our strategy is in $S \cup T$, so at most $5|S| = 5\,r(F^0, P)$ regions are retrieved.

Case 1. Let R_b be the region on the Pareto front in F^{OPT} that dominates R_a. In particular, the top-right corner of (the original region) R_b must dominate R_a. Furthermore, R_a not being dominated in F^0 implies that the top-right corner q of R_a is above $\mathrm{IPF}(F^0)$. The point q implies an outgoing dependency from R_b to R_a in F^0, which means R_b is retrieved by our strategy before R_a is considered. Thus, R_a is not retrieved by Algorithm 1.

Case 2. We apply Lemma 9 to observe that *any* algorithm must retrieve R_a to obtain a finished point set. So, F^{OPT} is not finished, a contradiction.

Case 3. By F^{OPT} being finished and R_a not being dominated, it must be that R_a is always and thus on $\mathrm{OPF}(F^{OPT})$. As R_a has an outgoing dependency in F^{OPT}, there must be a region R_b with a point $q \in R_b$ such that q is above $\mathrm{IPF}(F^{OPT})$, and $q \in R_a$ or the top-right corner of R_a dominates q. In both cases, setting $p_b = q$ and p_a as the top-right or bottom-left corner of R_a results in two distinct Pareto fronts, a contradiction. ◀

4 Overview of the reconstruction programs

In the remainder, we present two reconstruction programs that implement our reconstruction strategy (Algorithm 1). We emphasize the techniques used and the challenges that we face.

Preprocessing. The reconstruction strategy requires preprocessing the original input F^0 by identifying the independent regions. The full set of dependencies has quadratic size. Fortunately, it suffices to determine whether each region has *any* outgoing dependency. A region R has an outgoing dependency to a region R' if certain geometric conditions are satisfied (Definition 3) *and* the top-right corner of R' does not dominate that of R. The main difficulty lies in identifying these geometric conditions efficiently for *only* those regions R' whose top-right corner does not dominate. Such filtering could be achieved using standard techniques at the cost of additional logarithmic factors. To avoid any compromise in preprocessing time, we construct for each region two representative points. We show that a region R has *an* outgoing dependency if and only if its top-right corner dominates at least one of these points, or is in a slightly degenerate case where one of its facets intersects a facet of another region. This characterization reduces the preprocessing task to performing $O(n)$ orthogonal range counting queries, resulting in $O(n \log n)$ preprocessing time.

Reconstructing from rectangles. When F is a family of (overlapping) rectangles, our algorithm achieves a running time of $O(r(F^0, P) \log n)$. The main challenge is to repeatedly identify a non-retrieved region R on $\mathrm{OPF}(F)$ in logarithmic time. We preprocess $F = F^0 - I^0$ into a *layer decomposition* by iteratively removing $\mathrm{OPF}(F)$ from F until no regions remain. During reconstruction, we retrieve regions that are not dominated in the current family of regions F, layer-by-layer. The key observation is that a point retrieved from layer i never dominates any region in the same layer. Consequently, all regions that lie on $\mathrm{OPF}(F)$ at the start of processing layer i remain on $\mathrm{OPF}(F)$ throughout that layer. After completing layer i, we must remove all regions dominated by the newly retrieved points from the remaining layers. By carefully propagating information between layers, we can perform these removals implicitly, spending only logarithmic time per retrieved point.

Reconstructing from squares. When F consists of unit squares, our goal is to achieve universal optimality. This is a very strong performance requirement: in particular, even spending $O(\log n)$ time per retrieved point is already too expensive. To reason about universal

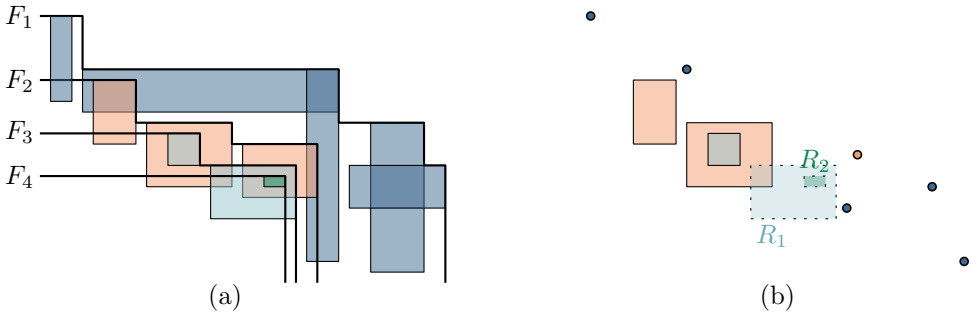

Figure 5 (a) The partition into layers F_1, F_2, F_3, F_4. (b) The active layer $L_i(F)$, $i \in [4]$, consists of the regions with solid boundaries. R_1 and R_2 are not in $L_3(F)$ and $L_4(F)$ because they are dominated, point-regions are never in $L_i(F)$.

optimality, fix an input F^0 and consider the number of combinatorially distinct Pareto fronts $\mathsf{PF}(P)$ that can arise over all realizations $P \sim F^0$. Let $U(F^0)$ denote this number. Universal optimality restricts us to a total running time of $\Theta(\log(U(F^0)))$. Equivalently, every unit of running time must be charged to a set of realizations $\{P_j\} \sim F^0$ that collectively generate sufficiently many distinct Pareto fronts $\mathsf{PF}(P_j)$ to pay for that cost. Establishing this running time requires a highly technical charging and counting scheme discussed in the full version [11].

5 An efficient reconstruction program for rectangles

In this section we present a reconstruction program that executes Algorithm 1 in $O(\log n)$ time per retrieval using $O(n \log n)$ preprocessing time. We first adapt Algorithm 1 slightly, such that we have more control over the order the dependent regions, i.e. the order that regions in $F^0 - I^0$, are retrieved in. To this end, we partition $F^0 - I^0$ into several *layers*.

▶ **Definition 11** (Figure 5(a)). *The family $F = F^0 - I^0$ is partitioned into* layers $F_1, \ldots, F_k$*:*

$$F_i := \begin{cases} \{R_a \mid R_a \in \mathrm{OPF}(F^0)\} & \text{if } i = 1 \\ \left\{R_a \mid R_a \in \mathrm{OPF}\left(F^0 - \bigcup_{j=1}^{i-1} F_j\right)\right\} & \text{if } i > 1 \text{ and this family is non-empty.} \end{cases}$$

This layer decomposition is *static*, it is solely defined based on the original regions $(F^0 - I^0)$ and remains fixed during the reconstruction phase. The first layer contains all regions of F^0 that are on $\mathrm{OPF}(F^0)$. The subsequent layers contain all regions that are on the outer Pareto front after removing all regions in earlier layers.

▶ **Observation 12.** *A region R_a in layer F_i is never dominated by a point $p \in R_b$ with $R_b \in F_j$, where $j \geq i$.*

5.1 An adapted reconstruction strategy

In the adapted reconstruction strategy, we go through the layers one by one, and retrieve some regions from each layer along the way. However, for the current family of regions F, we might no longer be interested in some of the regions in a layer. We therefore define the following subfamily of each layer F_i called the *active layer*.

▶ **Definition 13** (Figure 5(b)). *For the current family of regions F, an* active layer $L_i(F)$ *is the subfamily of regions in F_i, where $R_a \in L_i(F)$ if and only if the following conditions hold:*

- $R_a \in F_i$, *and*
- R_a *is not dominated in* F, *and*
- R_a *is not retrieved.*

To obtain an adapted reconstruction strategy, we replace Stage 1 of Algorithm 1 by the subroutine Algorithm 2. Specifically, instead of retrieving any region on $\mathrm{OPF}(F)$ with an outgoing dependency in F^0, we first retrieve the regions in the active layer $L_1(F)$, then the regions in $L_2(F)$, etc. We denote this adapted strategy by Algorithm 1-2.

Algorithm 2 A subroutine for Algorithm 1 to retrieve regions on $\mathrm{OPF}(F)$.

```
for i = 1 up to k do
  while L_i(F) ≠ ∅ do
    Retrieve any region R_a ∈ L_i(F).
```

When processing an active layer $L_i(F)$, all regions in $L_i(F)$ are on the outer Pareto front:

▶ **Lemma 14.** *When Algorithm 1-2 considers an active layer $L_i(F)$, then $L_i(F) \subseteq \mathrm{OPF}(F)$.*

Proof. Let $R_a \in L_i(F)$ with top-right corner r. We have that $L_j(F) = \emptyset$ for all $j < i$, and thus every region in $\bigcup_{j=0}^{i-1} F_j$ is either retrieved or dominated. It follows that R_a is not dominated by the top-right corner of a region in $\bigcup_{j=0}^{i-1} F_j$. By Observation 12, the point r is also not dominated by the top-right corner of any region in $\bigcup_{j=i}^{k} F_j$. It follows that r is on the staircase of $\mathrm{OPF}(F)$, and thus $R_a \in \mathrm{OPF}(F)$. ◀

Together with Observation 12 this implies that Algorithm 1-2 executes Algorithm 1.

▶ **Corollary 15.** *Algorithm 1-2 is an instance-optimal retrieval strategy.*

5.2 An efficient reconstruction program

To compute the independent regions I^0 efficiently, we derive a more convenient characterization of independent regions. In Definition 3, there are infinitely many choices for the point $q \in R_b$. We show that in most cases only two of these choices are relevant: For a region $R \in F^0$, let $\ell(R)$ be the leftmost point on the top facet of R that lies above $\mathrm{IPF}(F^0)$, and $r(R)$ the bottommost point on the right facet of R that lies above $\mathrm{IPF}(F^0)$, see Figure 6(a).

▶ **Lemma 16.** *R_a has an outgoing dependency to R_b if and only if there is a point q on the top or right facet of R_b that satisfies Definition 3.*

Proof. The "if" direction is trivial, so let's prove the "only if" direction. Let R_a have an outgoing dependency to R_b, certified by a point $\tilde{q}$ that satisfies Definition 3. Consider the closed line segment S from $\tilde{q}$ to the top-right corner of R_a. Then all points on S satisfy conditions II and III of Definition 3. Let q be the top-rightmost point of $S \cap R_b$. If q is on the top or right facet of R_b, this q shows the "only if" direction. Otherwise, q is the top-right corner of R_a. But then, the top-right corner of R_b dominates R_a – a contradiction to condition I of Definition 3. ◀

▶ **Corollary 17** (Figure 6(b)). *R_a has an outgoing dependency to R_b if and only if either*
(1) *there is a point $q \in \{\ell(R_b), r(R_b)\}$ such that the top-right corner of R_a dominates q; or*
(2) *the top or right facet of R_a intersects a top or right facet of R_b at a point above* $\mathrm{IPF}(F^0)$.

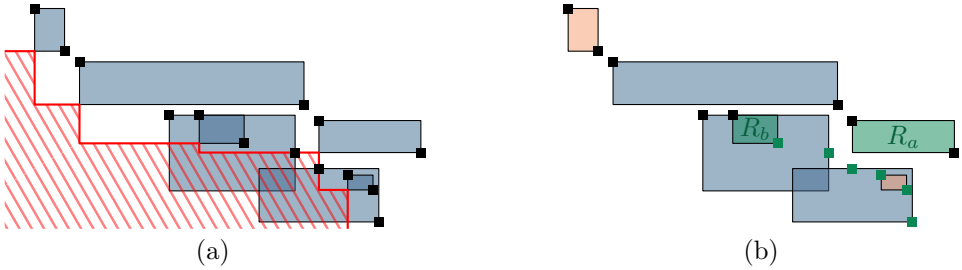

Figure 6 (a) The squares indicate $\ell(R)$ and $r(R)$. (b) R_a has a case (1) outgoing dependency because of the green points. R_b has a case (2) outgoing dependency. Orange regions are independent.

Proof. We first show the "only if" direction. By Lemma 16, there is a point q on the, without loss of generality, top facet of R_b that satisfies Definition 3. Let q be the leftmost such point. Consider condition III: If q lies on the top or right facet of R_a, then case (2) applies. Otherwise $q \in \{\ell(R_b), r(R_b)\}$, and case (1) applies.

For the "if" direction, in both cases, we first verify condition I of Definition 3. Indeed, in neither case can the top-right corner of R_b dominate R_a. Then, in case (1), the point q satisfies conditions II and III. In case (2), any intersection point satisfies these conditions. ◀

What remains is to turn Corollary 17 into an algorithm for finding the independent regions. We first compute $\mathrm{IPF}(F^0)$ and build an orthogonal range query data structure over all $\ell(R), r(R)$ in $O(n \log n)$ time and $O(n)$ space [9]. For every region R_a, we query the open area dominated by the top-right corner of R_a to check whether case (1) of Corollary 17 applies. Next, we think of the top and right facet of each region as closed rectangles of width 0, and build a rectangle-rectangle intersection counting data structure [9] over these. For every region R_a, we query the top and right facet. In total, there are always at least 4 intersections (from R_a itself). Case (2) of Corollary 17 applies if and only if there are more than four. The independent regions are precisely those for which neither case applies, so:

▶ **Lemma 18.** *Given F^0, we can compute I^0 in $O(n \log n)$ time and $O(n)$ space.*

Preprocessing phase. We first invoke Lemma 18 to find I^0 and then construct $\mathrm{OPF}(I^0)$ in $O(n \log n)$ time. We also store I^0 in an orthogonal ray shooting data structure to prepare for Stages 2 and 3 of Algorithm 1. Set $F = F^0 - I^0$. Next, we partition F into a layer decomposition which finds the partition $\{F_i\}$ and constructs $\mathrm{OPF}(F_i)$ for each index i. We augment it with a planar point location data structure that for any query point q returns the layer F_i such that the downward vertical ray from q hits $\mathrm{OPF}(F_i)$. This allows us to efficiently determine between which layers a retrieved point lies. The layer decomposition and its point location data structure can be computed in $O(n \log n)$ time and $O(n)$ space by a straightforward adaptation of Chazelle's algorithm to compute the convex layers of a planar point set [10] on the top-right corners of the regions.

▶ **Definition 19.** *We store for each layer i a tree T_i that stores the regions in* $\mathrm{OPF}(F_i)$ *in-order as a leaf-based leaf-linked balanced binary tree. Moreover, we store a list of* some *maximal intervals along* $\mathrm{OPF}(F_i)$ *that are dominated by points in F_j for $j < i$.*

This tree will later allow us to efficiently disregard all maximal intervals of consecutive leafs that are dominated in F. Specifically, it will allow us to maintain a doubly linked list storing all regions in $\mathrm{OPF}(F_i)$ that are not dominated in F.

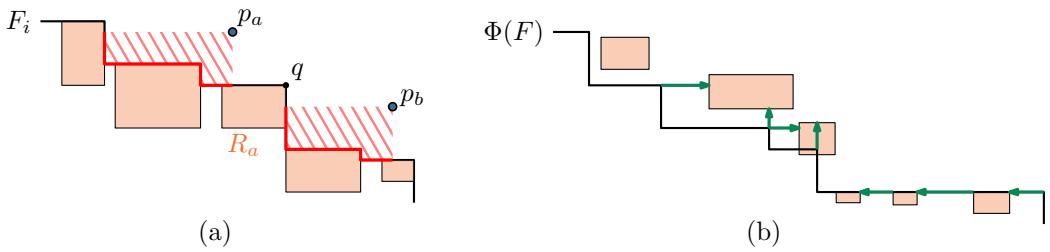

Figure 7 (a) Intervals I_{p_a} and I_{p_b} not overlapping implies there is a top-right corner q between them. (b) The ray shooting queries that imply which regions in I have to be retrieved.

Reconstruction phase. As we execute the subroutine in Algorithm 2, we retrieve all regions in the active layers, layer-by-layer. We maintain the following invariant:

> When we process the active layer $L_i(F)$, we store *all* maximal intervals of points on $\mathrm{OPF}(F_i)$ dominated by points in F_j for $j < i$, and a doubly linked list of $L_i(F)$.

Note that this requirement only has to hold when processing $L_i(F)$. We achieve this invariant as follows. When we process an active layer $L_i(F)$, we assume the invariant to be true. Note that this is trivially true when we start processing $L_1(F)$. Moreover, observe that if it is true at the start of processing $L_i(F)$ then it remains true throughout by Observation 12.

We rely on the following lemma, which intuitively states two consecutive maximal intervals of dominated points along $\mathrm{OPF}(F_i)$ can be merged, *if* there is no $R \in F_i$ between them:

▶ **Lemma 20.** *Let I_{p_a} and I_{p_b} be two intervals along $\mathrm{OPF}(F_i)$ that are dominated by points p_a and p_b, respectively. Then either $I_{p_a} \cap I_{p_b} \neq \emptyset$, or there is region $R_a \in F_i$ in $\mathrm{OPF}(F_i)$ between I_{p_a} and I_{p_b}.*

Proof. Suppose that $I_{p_a} \cap I_{p_b} = \emptyset$, see Figure 7(a). Then there must be a top-right corner q of the staircase $\mathrm{OPF}(F_i)$ between the two intervals and there is a region $R_a \in \mathrm{OPF}(F_i)$ whose top-right corner is q. It follows that there is a region in $\mathrm{OPF}(F_i)$ between I_{p_a} and I_{p_b}. ◀

Our retrieval update. Whenever our algorithm retrieves a region $R_a \in L_i(F)$, it updates the list of intervals on $\mathrm{OPF}(F_{j+1})$ for exactly *one* j. A merging strategy then ensures that the invariant holds for the active layer as we process it. Specifically, when we retrieve a region $R_a \in L_i(F)$ we perform a point location with p_a on the layer decomposition. If p_a is in the inner Pareto front, we discard it. Otherwise, suppose that p_a lies between $\mathrm{OPF}(F_j)$ and $\mathrm{OPF}(F_{j+1})$. The point p_a then dominates a contiguous interval along $\mathrm{OPF}(F_{j+1})$ (which corresponds to a contiguous interval of leaves in T_{j+1}). We search along T_{j+1} to identify this interval in $O(\log n)$ time. We then insert this interval I_{p_a} into the ordered list of maximally dominated intervals, iteratively merging I_{p_a} with any interval I_{p_b} in the list that is intersected by I_{p_a}. A merge between two intervals replaces both by their union. Note that this merge takes amortized constant time since there are at most $O(1)$ such intervals I_{p_b} that are not contained in I_{p_a} and all intervals contained in I_{p_a} are permanently removed.

Maintaining our invariant. When we have finished processing the active layer $L_i(F)$, we must ensure that our invariant holds for $L_{i+1}(F)$. We can now rely on Lemma 20. Per our invariant, we stored the set of maximal intervals along $\mathrm{OPF}(F_i)$ that are dominated by a point in F_j for $j < i$. Per our merging strategy, we merged these intervals upon inserting them such that between any two consecutive intervals there is a (non-dominated) region R in F_i (which is thus a region of $L_i(F)$). Every such region R is retrieved by our

retrieval strategy, and we charge $O(\log n)$ to every such interval. Each interval along $\mathrm{OPF}(F_i)$ corresponds to some fictive point q that dominates the interval and we search the balanced tree storing $\mathrm{OPF}(F_{i+1})$ in $O(\log n)$ time to find the maximal interval along $\mathrm{OPF}(F_{i+1})$ dominated by q. With this total charge, we search for every such fictive q for the maximal interval along $\mathrm{OPF}(F_{i+1})$ dominated by q. We merge these new intervals with the intervals already stored along $\mathrm{OPF}(F_{i+1})$ in the exact same manner as above at amortized constant overhead. Every point in F_j with $j < i$ that dominates any point along $\mathrm{OPF}(F_{i+1})$ must have had an interval either along $\mathrm{OPF}(F_{i+1})$ or $\mathrm{OPF}(F_i)$ by our invariant. It follows that this merging procedure maintains our invariant for $\mathrm{OPF}(F_{i+1})$ and charges logarithmic time to each retrieved region in $L_i(F)$. The instance-optimality of our retrieval strategy then implies we spend $O(r(F^0, P) \log n)$ total time on these retrievals.

The independent regions. For each region R_a that we retrieve in Stage 1, we record the point p_a that is retrieved. Let F be the current family of regions then we denote by $\Phi(F)$ the Pareto front of all points that we retrieved, which we can maintain in $O(\log n)$ time per retrieved point. By Lemma 6, $\mathrm{OPF}(F) = \Phi(F)$ by the end of Stage 1.

We then execute Stage 2 and Stage 3 using orthogonal range searching queries on I^0. In particular, we note that for each edge e of the staircase of $\Phi(F)$, only one region in I can intersect e in its interior, or bottom or left facet (if there are two such regions then one of the two has an outgoing dependency to the other). However, there may be many regions whose top or right facet intersect e, see Figure 7(b). We thus shoot a ray from each vertex of $\Phi(F)$ in each cardinal direction, charging to the retrieved vertex, spending $O(r(F^0, P) \log n)$ time. If a ray hits a region that intersects the corresponding edge only in its top (or right) facet, then we continue to the next region along $\mathrm{OPF}(I^0)$ as long as this next region also intersects the same edge only in its top (or right) facet. As a consequence, we obtain a superset of all regions retrieved in Stage 2 and 3. For each of these regions, we check if they meet the requirement for retrieval at constant overhead and retrieve them if needed. It follows that we execute Algorithm 1-2 after $O(n \log n)$ preprocessing in $O(r(F^0, P) \log n)$ time.

Finally, we note that we can slightly strengthen the result. We can not only find a set of regions that is *finished*, we can also output a pointer to a balanced tree that stores the Pareto front in order. Specifically, we merge the fronts $\mathrm{OPF}(I)$ and $\Phi(F)$ into a staircase of regions that has a 1:1 correspondence to $\mathsf{PF}(P)$. As we retrieve regions from I the left-to-right order along $\mathrm{OPF}(I)$ does not change. We can therefore use any balanced tree over $\mathrm{OPF}(I^0)$ as a balanced tree over $\mathrm{OPF}(I)$. Our next key observation is that each edge of a Pareto front intersects another Pareto front exactly once. For each edge of the staircase of $\Phi(F)$, we binary search in $O(\log n)$ time to find the edge of $\mathrm{OPF}(I)$ that is intersected by $\Phi(F)$ (if any). Given $O(|\Phi(F)|) \subset O(r(F^0, P))$ intersection points, we compute the joint maximum of $\mathrm{OPF}(I)$ and $\Phi(F)$ in $O(r(F^0, P))$ time which corresponds to $\mathsf{PF}(P)$.

▶ **Theorem 21.** *Let F be a family of n axis-aligned rectangles. We can process F in $O(n \log n)$ time and $O(n)$ space, such that given $P \sim F$ we reconstruct the Pareto front $\mathsf{PF}(P)$ in $O(r(F, P) \log n)$ time.*

In the full version [11] we prove the following even stronger result for unit-squares.

▶ **Theorem 22.** *Let F be a family of n axis-aligned unit squares. We can process F in $O(n \log n)$ time and $O(n)$ space, such that given $P \sim F$ we reconstruct the Pareto front $\mathsf{PF}(P)$ in universally optimal time.*

References

1 Ankush Acharyya, Ramesh K. Jallu, Vahideh Keikha, Maarten Löffler, and Maria Saumell. Minimum color spanning circle of imprecise points. *Theoretical Computer Science*, 930:116–127, 2022. doi:10.1016/j.tcs.2022.07.016.

2 Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. Instance-Optimal Geometric Algorithms. *Journal of the ACM (JACM)*, 2017. doi:10.1145/3046673.

3 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.

4 Sarita de Berg, Ivor van der Hoog, Eva Rotenberg, Daniel Rutschmann, and Sampson Wong. Instance-Optimal Imprecise Convex Hull. In *European Symposium on Algorithms (ESA)*, volume 351 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.25.

5 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.

6 Kevin Buchin and Wolfgang Mulzer. Delaunay triangulations in O(sort(n)) time and more. *Journal of the ACM (JACM)*, 58(2), 2011. doi:10.1145/1944345.1944347.

7 Jean Cardinal, Samuel Fiorini, Gwenaël Joret, Raphaël Jungers, and J. Ian Munro. Sorting under partial information (without the ellipsoid algorithm). *Combinatorica*, 33(6):655–697, 2013. doi:10.1007/s00493-013-2821-5.

8 Timothy M. Chan. Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996. doi:10.1007/BF02712873.

9 Bernard Chazelle. A Functional Approach to Data Structures and Its Use in Multidimensional Searching. *SIAM Journal on Computing*, 17(3):427–462, 1988. doi:10.1137/0217026.

10 Bernard Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31(4):509–517, 2006. doi:10.1109/TIT.1985.1057060.

11 Sarita de Berg, Nynne Maria Foldager Bække, Frida Astrup Eriksen, Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. Instance and universally optimal bounds for imprecise pareto fronts. *arXiv preprint*, 2026. doi:10.48550/arXiv.2605.07523.

12 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.

13 Klim Efremenko, Gillat Kol, Raghuvansh R. Saxena, and Zhijun Zhang. Universally Optimal Streaming Algorithm for Random Walks in Dense Graphs. In *Innovations in Theoretical Computer Science Conference (ITCS)*, volume 362 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 55:1–55:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPIcs.ITCS.2026.55.

14 David Eppstein, Michael T. Goodrich, Abraham M. Illickan, and Claire A. To. Entropy-Bounded Computational Geometry Made Easier and Sensitive to Sortedness. In *Canadian Conference on Computational Geometry (CCCG)*, 2025. doi:10.48550/arXiv.2508.20489.

15 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.

16 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.

17 Gianni Franceschini, S. Muthukrishnan, and Mihai Pătraşcu. Radix Sorting With No Extra Space. In *European Symposium on Algorithms (ESA)*, pages 194–205, 2007. doi:10.1007/978-3-540-75520-3_19.

18 Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters (IPL)*, 1:132–133, 1972. doi:10.1016/0020-0190(72)90045-2.

19 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.

20 Bernhard Haeupler, Richard Hladík, John Iacono, Václav Rozhoň, Robert E. Tarjan, and Jakub Tětek. Fast and Simple Sorting Using Partial Information. *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3953–3973, 2025. doi:10.1137/1.9781611978322.134.

21 Yijie Han and Mikkel Thorup. Integer Sorting in $O(n\sqrt{\log\log n})$ Expected Time and Linear Space. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 135–144, 2002. doi:10.1109/SFCS.2002.1181890.

22 Martin Held and Joseph Mitchell. Triangulating Input-constrained Planar Point Sets. *Information Processing Letters (IPL)*, 2008. doi:10.1016/j.ipl.2008.09.016.

23 Richard Hladík and Jakub Tětek. Near-Universally-Optimal Differentially Private Minimum Spanning Trees. In *Foundations of Responsible Computing (FORC)*, volume 329 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.FORC.2025.6.

24 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.

25 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. doi:10.1137/1.9781611977073.122.

26 Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. A Combinatorial Proof of Universal Optimality for Computing a Planar Convex Hull. In *European Symposium on Algorithms (ESA)*, volume 351 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 102:1–102:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.102.

27 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.

28 Ivor van der Hoog, Eva Rotenberg, and Daniel Rutschmann. Tight Universal Bounds for Partially Presorted Pareto Front and Convex Hull. *arXiv preprint*, 2025. doi:10.48550/arXiv.2512.06559.

29 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.

30 Jeff Kahn and Jeong Han Kim. Entropy and sorting. In *ACM symposium on Theory of Computing (STOC)*, pages 178–187. Association for Computing Machinery, 1992. doi:10.1145/129712.129731.

31 Jeff Kahn and Michael Saks. Balancing poset extensions. *Order*, 1(2):113–126, 1984. doi:10.1007/BF00565647.

32 David G Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986. doi:10.1137/0215021.

33 Marc van Kreveld, Maarten Löffler, and Joseph Mitchell. Preprocessing Imprecise Points and Splitting Triangulations. *SIAM Journal on Computing (SICOMP)*, 2010. doi:10.1137/090753620.

34 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.

35 Maarten Löffler and Benjamin Raichel. Preprocessing Disks for Convex Hulls, Revisited. *arXiv preprint*, 2025. doi:10.48550/arXiv.2502.03633.

36 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.

37 Raimund Seidel. A Method for Proving Lower Bounds for Certain Geometric Problems. In *Computational Geometry*, volume 2 of *Machine Intelligence and Pattern Recognition*, pages 319–334. North-Holland, 1985. doi:10.1016/B978-0-444-87806-9.50017-7.