Simple Multi-Pass Streaming Algorithms for Skyline Points and Extreme Points

Timothy M. Chan
Dept. of Computer Science, University of Illinois at Urbana-Champaign, IL, USA

Saladi Rahul
Dept. of Computer Science and Automation, Indian Institute of Science Bangalore, India

Abstract

In this paper, we present simple randomized multi-pass streaming algorithms for fundamental computational geometry problems of finding the skyline (maximal) points and the extreme points of the convex hull. For the skyline problem, one of our algorithm occupies $O(h)$ space and performs $O(\log n)$ passes, where h is the number of skyline points. This improves the space bound of the currently best known result by Das Sarma, Lall, Nanongkai, and Xu [VLDB'09] by a logarithmic factor. For the extreme points problem, we present the first non-trivial result for any constant dimension greater than two: an $O(h \log^{O(1)} n)$ space and $O(\log^d n)$ pass algorithm, where h is the number of extreme points. Finally, we argue why randomization seems unavoidable for these problems, by proving lower bounds on the performance of deterministic algorithms for a related problem of finding maximal elements in a poset.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases multi-pass streaming algorithms, skyline, convex hull, extreme points, randomized algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2021.22

Funding *Timothy M. Chan*: Work supported in part by NSF Grant CCF-1814026.
Saladi Rahul: Work supported by IISc start-up research grant.

1 Introduction

1.1 Multi-pass streaming model

The *streaming* model has emerged as a popular model to handle massive data. Unfortunately, streaming algorithms for geometric problems that make a single pass over the input and work with a small amount of space are typically unable to give exact solutions. This motivates the *multi-pass* streaming model, where the algorithm is allowed to make multiple passes over the input. The input sequence remains unchanged in each pass. The goal is to minimize the amount of working space (or memory) and the number of passes. The data is assumed to be explicitly stored either in a disk or in a cloud, which facilitates multiple passes over it, but since each pass is costly it is essential to minimize the passes.

Summarization queries are the most widely studied class of problems in the streaming model. The focus of this paper is *geometric* summarization queries in the multi-pass streaming model. Specifically, we study two fundamental geometric summarization problems: the *skyline* problem, asking for the "dominating" points in the data, and the *extreme points* problem, asking for the vertices of the convex hull, which succinctly represents the shape of the point cloud. The goal is to design algorithms which are *output-sensitive* in space (i.e., near $O(h)$ space when there are h skyline/extreme points) and perform few passes. Note that $O(h)$ space[1] is indeed the best possible if we want to store the skyline/extreme points in memory, rather than write to an output stream.

[1] Throughout the paper, all space bounds are measured in words, not bits. A word may store one input point, or an $O(\log n)$-bit number.

© Timothy M. Chan and Saladi Rahul;
licensed under Creative Commons License CC-BY 4.0

We will focus on algorithms which are optimized to work efficiently even for the worst-case input. For the problems of interest in this paper (such as the skyline problem), one could argue that an incremental algorithm which updates the skyline as each new element is inserted will work well for randomly generated inputs (for example, for points uniformly distributed in a square, any prefix of the input has an expected $O(\log n)$ number of skyline points, and so it is not difficult to obtain a solution using $O(\log n)$ expected space, with just one pass). However, point sets encountered in practice may not be randomly distributed.

1.2 Skyline points

Let P be a set of n points lying in $\mathbb{R}^d$ for a constant d. A point $p = (p_1, \ldots, p_d)$ *dominates* another point $q = (q_1, \ldots, q_d)$ if $p_i > q_i$ for all $i \in \{1, \ldots, d\}$. In the *skyline* (also called *maxima*) problem, the goal is to find all points $p \in P$ such that p is not dominated by any other point in P. The problem has been extensively studied by the computational geometry [20] and the database community (e.g., see [22] and the references therein). Currently, the best known result in the word-RAM model is an $O(n \log^{d-3} n)$-time algorithm by Chan, Larsen and Pătraşcu [6] (also see [1] and [14] for the best-known output-sensitive algorithms in the word-RAM model and the I/O model, respectively, and [2] for instance-optimal algorithms in $\mathbb{R}^2$ and $\mathbb{R}^3$).

The formal study of the skyline problem in the multi-pass streaming model was initiated by Das Sarma, Lall, Nanongkai, and Xu [11]. The naive $O(nh)$-time output-sensitive algorithm (e.g., see [8]) can be implemented in the multi-pass setting with $O(h)$ space but requires $O(h)$ passes, where h is the number of skyline points of P. Das Sarma et al. proposed a new randomized algorithm using significantly fewer number of passes: it requires $O(h \log n)$ space and just $O(\log n)$ passes, with high probability,[2] for any constant dimension d.

Alternatively, it is not difficult to obtain a deterministic algorithm with $O(h \log n)$ space and $O(\log^{d-1} n)$ passes, by adapting Kirkpatrick and Seidel's output-sensitive skyline algorithm [16] in the multi-pass setting, similar to Chan and Chen's multi-pass adaptation [5] of Kirkpatrick and Seidel's output-sensitive 2-d convex hull algorithm [17]; see the appendix. However, with this approach, the number of logarithmic factors grows as the dimension increases.

New randomized algorithms. Our first result is a variant of Das Sarma et al.'s algorithm that solves the d-dimensional skyline problem using $O(\log n)$ passes and $O(h)$ space – this improves space by a logarithmic factor. Our bounds also hold with high probability. Although the improvement is not big, the highlight here is the simplicity of our analysis compared to the longer and more complicated analysis by Das Sarma et al. [11]. (The simpler analysis is well-suited for teaching purposes.) Also, unlike their analysis, which is specialized to the skyline problem, our analysis naturally extends to the extreme points problem, as we will discuss later.

Our algorithm can also achieve a trade-off: by increasing the space bound to $O(bh)$ for a parameter b, the number of passes can be lowered to $O(\log_b n)$. For example, setting $b = n^\delta$ gives $O(hn^\delta)$ space and $O(1/\delta)$ passes, for any $\delta \in (0, 1]$. Setting $b = \log^\delta n$ for an arbitrarily small constant $\delta > 0$ gives $O(h \log^\delta n)$ space and $O\left(\frac{\log n}{\log \log n}\right)$ passes.

In the $O(h)$-space regime, we also describe a refinement of the algorithm that further reduces the number of passes from $O(\log n)$ to $O\left(\log h + \frac{\log n}{\log \log n}\right)$, which is slightly sublogarithmic, assuming that h is not too big. These bounds hold in expectation.

[2] Throughout this paper, "with high probability" will imply "with probability at least $1 - \frac{1}{n^c}$", where c is a sufficiently large constant.

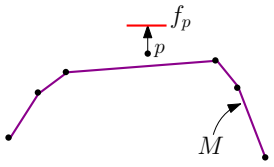
Our randomized algorithms for skyline points, like previous work [11], extends (with the same bounds) to the general setting of a partially ordered set, or *poset*. A poset is a pair $(P, \succ)$, where P is the set of n elements and $\succ$ is an irreflexive, transitive binary relation on the elements of P. The problem here is to find all the *maximal* elements in a poset, i.e., elements $a \in P$ such that there is no element $b \in P$ with $b \succ a$. We only assume an oracle that can test whether $a \succ b$ for any two given elements a and b.

Is randomization essential? A natural question is whether randomization is essential for the algorithms proposed in this paper. At least for the poset problem we can answer this question. We show that any deterministic algorithm which uses $O(h)$ space to find all maximal points in a poset has to perform $\Omega(h)$ passes. In other words, among the class of deterministic algorithms to compute maximal elements in a poset, the naive idea of finding one maximal element per pass is the best possible algorithm. Therefore, randomization is not just necessary, but in fact leads to dramatically improved results. Our lower bound proof is based on a new, interesting adversarial argument.

1.3 Extreme points

Given a set P of n points lying in $\mathbb{R}^d$, a point $p \in P$ is an *extreme point* if $conv(P) \neq conv(P \setminus \{p\})$, where $conv(P)$ denotes the convex hull of P. In the multi-pass setting, this problem was first studied in $\mathbb{R}^2$ by Chan and Chen [5], who obtained an algorithm with $O(h \log^2 n)$ space and $O(\log^2 n)$ passes, and then recently by Farach-Colton, Li, and Tsai [12], who improved the bounds to $O(h \log^2 n)$ space and $O(\log n)$ passes. These solutions are based on the output-sensitive divide-and-conquer algorithms of Kirkpatrick and Seidel [17] and Chan, Snoeyink, and Yap [7], and hence, they inherently work only in $\mathbb{R}^2$ (and possibly in $\mathbb{R}^3$) – in dimension greater than three, these divide-and-conquer algorithms have complexity at least the number of hull facets, which can be much larger than the number of hull vertices.

In this work, we present the first non-trivial result for any constant dimension greater than two: an $O(h \log^{O(1)} n)$-space, $O(\log^d n)$-pass algorithm. Our solution requires extending our skyline algorithm in a nontrivial fashion.



In the non-streaming setting, $O(nh)$-time output-sensitive algorithms for the extreme points problem were reported independently by Clarkson [8], Chan [3], Ottmann et al. [19], and Dula and Helagason [15]. These algorithms are all similar and work by incrementally building a subset $M \subseteq P$ of the extreme points of the upper hull. For each input point $p \in P$, we first check if p lies below or above the current upper hull of M – this step reduces to a linear program with $O(h)$ constraints. If p lies below, then it can be removed. Otherwise, p may not necessarily be extreme in P, but we can shoot an upward ray from p to hit a facet $f_p \in conv(P)$ – this step reduces to a linear program with n constraints.[3] The $d+1$ vertices defining f_p are extreme in P and can be added to M. The whole algorithm requires solving

[3] A version of the algorithm by Clarkson [8] and Dula and Helagason [15] avoids linear programming in this step and instead finds a point extreme in the direction orthogonal to f_p, and adds to M. If p is still above the current upper hull of M, we repeat.

$O(n)$ linear programs with $O(h)$ constraints and $O(h)$ linear programs with n constraints, and thus takes $O(nh)$ time by known linear-time linear programming algorithms in constant dimensions [4, 9, 21].[4] Because it finds a constant number of extreme points per iteration, the algorithm is inherently *sequential* and requires $\Omega(h)$ passes.

In our solution, we use randomization instead to find more extreme points at each iteration, by generating $O(h)$ linear programs with n constraints that can be solved *in parallel*, for each of logarithmically many rounds.

The rest of the paper is organized as follows. In Sections 2 and 3, we present our algorithms for computing skyline points and extreme points, respectively. In Section 4 we prove a lower bound on the performance of any deterministic algorithm for finding the maximal elements in a poset.

2 Randomized algorithms for skyline points in $\mathbb{R}^d$

2.1 Main algorithm

In this section we will prove the following result.

▶ **Theorem 1.** *Let P be a set of n points in $\mathbb{R}^d$, and h be the number of skyline points. Then, with high probability, there is an $O(bh)$-space, $O(\log_b n)$-pass algorithm to compute the skyline points, where b is a parameter in the range $[2, n/h]$.*

Our algorithm is a refinement of Das Sarma et al.'s algorithm [11]. The general idea is to use random sampling to somehow prune a fraction of the input points in each round. The right sample size requires knowledge of h, which we guess by repeated doubling.

Let P be a set of n points in $\mathbb{R}^d$. The algorithm consists of two stages. In Stage I, the goal is to handle the case when $h \leq \log_b n$. Here, we can just run a known naive algorithm (e.g., see [8]) which finds one skyline point per pass and thus requires $h \leq \log_b n$ passes. For example, having found skyline points $p_1, \ldots, p_{i-1}$ in the first $i-1$ passes, we can find the point, $p_i \in P$, with the largest x-coordinate value among the points of P not dominated by $p_1, \ldots, p_{i-1}$, in the next pass. If some skyline points have not been found after $\log_b n$ passes, we proceed to Stage II. Let c be a sufficiently large constant.

The set M maintains the current list of skyline points found by the algorithm. In Stage-II, an *iteration* will consist of two passes. Let P_i denote the set of unclassified points at the beginning of the i-th iteration (where $i \geq 1$), i.e., the points of P_i which have not yet been labeled as skyline or non-skyline. In the first pass of the i-th iteration (step 3a), we independently sample each point of P_i with probability $\frac{cr_i}{|P_i|}$, where c is a sufficiently large constant. Let $R_i \subseteq P_i$ be the sampled set of points. In the second pass of the i-th iteration (step 3b), the goal is to find a set of skyline points R_i^+ which dominate *all* the points in R_i (i.e., every point in R_i is dominated by some point in R_i^+). This is achieved as follows: initially, set $R_i^+ \leftarrow R_i$. Then for each point $p \in P_i$, if p does not dominate any point in R_i^+, then we do nothing. Otherwise, we add p to R_i^+ and remove the points of R_i^+ which are dominated by p. Note that this step will not increase the size of R_i^+, although it could potentially decrease the size of R_i^+. At the end of the second pass, the claim is that the points in R_i^+ are all skyline points (see Lemma 2). The algorithm terminates when all the points have been classified.

[4] With range searching data structures, the overall running time can be lowered to $O(n \log^{O(1)} h + (nh)^{1-1/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n)$ [3]. We will ignore bounds of this flavor, since they do not improve upon $O(nh)$ by much as d increases.

Algorithm 1 Finding Skyline Points.

0. $M \leftarrow \emptyset$.
//Stage I to handle $h \leq \log_b n$.
1. For the first $\log_b n$ passes, run the naive algorithm of finding one skyline point per pass.

//Stage II to handle $h > \log_b n$.
2. $P_1 \leftarrow P$, $r_1 \leftarrow c \log_b n$, and $i \leftarrow 1$.
3. Repeat till $|P_i| < r_i$:
3a. Find a sample $R_i \subseteq P_i$ of size r_i. Set $R_i^+ \leftarrow R_i$.
3b. For each point $p \in P_i$: *//Finding new skyline points.*
- Add p to R_i^+ if p dominates at least one point in R_i^+.
- Remove the points in R_i^+ which are dominated by p.
3c. $M \leftarrow M \cup R_i^+$.
3d. $P_{i+1} \leftarrow P_i \setminus (R_i^+ \cup \{\text{points of } P_i \text{ dominated by } R_i^+\})$.
3e(i). If $|P_{i+1}| \geq |P_i|/b$, then $r_{i+1} \leftarrow br_i$. *// Ineffective iteration.*
3e(ii). Otherwise, $r_{i+1} \leftarrow r_i$.
3f. $i \leftarrow i + 1$.

4. Compute the skyline points of the $O(r_i)$ points in P_i and add them to M.

▶ Remark. Since the size of the sets P_i can be significantly larger than h, we cannot afford to store them explicitly. To overcome this issue, we will instead use M to *implicitly* maintain P_i: observe that whenever a point p in the stream arrives, if none of the points in M dominate p, then $p \in P_i$.

Analysis. Stage I of the algorithm requires $O(\log_b n)$ space and $O(\log_b n)$ passes. From now on we will focus on Stage II and assume $h > \log_b n$. The i-th iteration is labeled *effective* if $|P_{i+1}| < |P_i|/b$. We start by proving a simple fact.

▶ **Lemma 2.** *At the end of the i-th iteration, all the points in R_i^+ are skyline points.*

Proof. For the sake of contradiction, assume that a point $p \in R_i^+$ is not a skyline point and let $q \in P_i$ be a point dominating it. If p appears before q in the stream, then it is easy to observe that p will not survive in R_i^+. On the other hand, if q arrives before p in the stream, then there are two cases:

- $p \in R_i^+$ at the end of the first pass. In that case, we know that in the second pass there is a point which will come before p and remove p from R_i^+. Once that happens, then p cannot be added back to R_i^+ in the second pass.
- $p \notin R_i^+$ in the first pass, but $p \in R_i^+$ at the end of the second pass. This implies that there is a point $p' \in R_i^+$ at the end of the first pass and p' is dominated by p; but p' would have been dominated by q as well, and hence p cannot be part of R_i^+ at the end of the second pass. ◀

The following lemma is the crux of our argument.

▶ **Lemma 3.** *When $r_i \geq cb^2h$ for a sufficiently large constant c, then all further iterations will be effective with high probability.*

Proof. Let M^* be the skyline points of P. Consider any i-th iteration in which $r_i \geq cb^2h$. Given a sample R_i, in step 3b we have constructed a set $R_i^+ \subseteq M^*$ which dominates all the points in R_i. The main question is this:

What is the probability that the number of points of $P_i \setminus R_i^+$ not dominated by R_i^+ is more than $|P_i|/b$?

This probability seems hard to bound directly. We turn the question around:

Fix a subset $A \subseteq M^*$, where the number of points of $P_i \setminus A$ not dominated by A is more than $|P_i|/b$. What is the probability that $R_i^+ = A$?

Observe that if any point $p \in P_i \setminus A$ not dominated by A is chosen to be in R_i, then p would be dominated by some point in R_i^+, making it impossible for $R_i^+ = A$. Using this observation, we get the following upper bound:

$$\begin{aligned}\Pr[R_i^+ = A] &\leq \Pr[\text{every point of } P_i \setminus A \text{ not dominated by } A \text{ is not in } R_i] \\ &\leq \left(1 - \frac{r_i}{|P_i|}\right)^{|P_i|/b} \\ &\leq e^{-r_i/b}.\end{aligned}$$

A trivial upper bound on the number of candidates for A is 2^h, since $A \subseteq M^*$. It turns out that this trivial bound is sufficient for our purposes. By the union bound,

$$\begin{aligned}&\Pr[\text{the number of points of } P_i \setminus R_i^+ \text{ not dominated by } R_i^+ \text{ is more than } |P_i|/b] \\ &\quad\leq 2^h \cdot e^{-r_i/b} \\ &\quad< 2^{-\Omega(cbh)} \qquad \text{since } r_i \geq cb^2h \\ &\quad= n^{-\Omega(c)} \qquad \text{since } h > \log_b n.\end{aligned}$$

It follows that an iteration is effective with high probability. ◀

Readers familiar with ε-nets or standard geometric Clarkson–Shor-style sampling analysis [10, 18] may find the preceding analysis similar to known arguments, but there is one interesting, key difference: the set system we are dealing with does not have constant VC dimension, but rather has dimension $\Theta(h)$. We use the 2^h upper bound on the number of possible sets A, instead of a more usual polynomial bound. (In dimension 2 and 3, one could decompose the region not dominated by $O(h)$ points into $O(h)$ cells of constant complexity, and could therefore use a more standard Clarkson-Shor–style argument, but the size of such decomposition blows up in dimension beyond 3.)

▶ **Lemma 4.** *The number of passes performed by the algorithm is $O(\log_b n)$ and the space occupied by the algorithm is $O(bh)$. Both bounds hold with high probability.*

Proof. The number of effective iterations is $O(\log_b n)$, since in each effective iteration the number of unclassified points go down by a factor of at least b. Now we will bound the number of iterations which are *ineffective*. By Lemma 3, with high probability, an ineffective iteration can only happen when $r_i < cb^2h$. Since the value of r is increased by a factor of b after each ineffective iteration, the total number of ineffective iterations will be bounded by $O(\log_b(b^2h))$. Therefore, with high probability the number of passes performed by the algorithm is $O(\log_b n + \log_b(b^2h)) = O(\log_b n)$.

Since an ineffective iteration can only happen when $r_i < cb^2h$, with high probability, the space occupied will be $O(b^2h)$. By choosing $b' = \sqrt{b}$, the space becomes $O(b'h)$ and the number of passes becomes $O(\log_{b'} n)$. ◀

Comparison. Compared to Das Sarma et al.'s algorithm [11], our use of a different algorithm in Stage I ensures that Stage II is only invoked to handle the case where h is sufficiently large, and as a result, we could afford a sample size smaller by a logarithmic factor than the sample size used by [11] and still obtain high probability bounds.

Compared to our analysis, Das Sarma et al.'s analysis [11] is longer and more complicated. It starts by constructing a graph consisting of h components (one per skyline point). To argue that in each pass a constant fraction of the points gets classified, the h components are categorized into *heavy* and *light*, and then a separate analysis is performed on the heavy and the light components. Also, their analysis is specialized to the skyline problem, whereas our analysis will extend to the extreme points problem as well.

Running time. Naively, each pass in Stage II can be implemented in $O(b^2nh)$ time, yielding a total running time of $O(b^2nh \log_b n)$, with high probability. In constant dimensions, we can use orthogonal range searching data structures to implement step 3b, and the total running time can be reduced to $O(n \log^{O(1)} h \log_b n)$.

Posets. The algorithm for skyline points naturally extends to the problem of finding maximal points in a poset by suitably adapting the definition of domination. The only modification needed is in Stage I: we used geometry (the x-coordinate values) to find one skyline point per pass. For poset, replace it with another naive algorithm which finds one maximal point per pass.

2.2 Further refinement

In this subsection, we present an interesting variant of the algorithm which slightly reduces the expected number of passes to sublogarithmic, if h is not too big, while using only $O(h)$ expected space.

▶ **Theorem 5.** *Let P be a set of n points in $\mathbb{R}^d$, and h be the number of skyline points. Then, there is an $O(h)$-space, $O\left(\log h + \frac{\log n}{\log\log n}\right)$-pass algorithm to compute the skyline points. Both bounds hold in expectation.*

The new algorithm is similar to our algorithm in Section 2.1, but with one key difference. An iteration will now be considered ineffective if the sample does not prune away a large number of points (this is as before), *nor* does it discover a large number of new skyline points (this is new). Another minor difference is that in case of an ineffective iteration, we will double the sample size (instead of lying by b). More precisely, the only change in the pseudocode is to replace step 3e(i) with the following:

3e(i). If $|P_{i+1}| \geq |P_i|/b$ and $|R_i^+| < h/b^2$, then $r_{i+1} \leftarrow 2r_i$ // *Ineffective iteration.*

We will fix the parameter b so that $\log_b n = b^2$ (and thus $b = \Theta\left(\sqrt{\frac{\log n}{\log\log n}}\right)$).

The following lemma shows that an ineffective iteration is not very likely to happen when the sample size is sufficiently large.

▶ **Lemma 6.** $\Pr[\text{iteration } i \text{ is ineffective} \mid r_i \geq h] \leq e^{-\Omega(b)}$.

Proof. We modify the proof of Lemma 3.

We have already shown that $\Pr[R_i^+ = A] \leq e^{-r_i/b}$. Before, we trivially bound the number of candidates for A by 2^h. This time, we will give a sharper upper bound. An ineffective iteration guarantees that $|A| < h/b^2$, and hence, the number of candidates for A is at most

$$\sum_{k=1}^{h/b^2} \binom{h}{k} \leq \frac{h}{b^2} \cdot \binom{h}{h/b^2} \leq \frac{h}{b^2} \cdot \left(\frac{eh}{h/b^2}\right)^{h/b^2} = b^{O(h/b^2)}.$$

By the union bound,

$$\begin{aligned}
&\Pr\left[\text{the number of points of } P_i \setminus R_i^+ \text{ not dominated by } R_i^+ \text{ is more than } |P_i|/b\right] \\
&\leq\ b^{O(h/b^2)} \cdot e^{-r_i/b} \\
&\leq e^{-\Omega(\frac{h}{b})} \quad \text{since } r_i \geq h \\
&\leq e^{-\Omega(b)} \quad \text{since } h \geq b^2.
\end{aligned}$$

◀

▶ **Lemma 7.** *The expected number of passes performed by the algorithm is* $O\left(\log h + \frac{\log n}{\log\log n}\right)$.

Proof. An effective iteration with $|R_i^+| \geq h/b^2$ can happen at most b^2 times. An effective iteration with $|P_{i+1}| < |P_i|/b$ can happen only $O(\log_b n)$ times. Therefore, effective iterations happen $O(\log_b n + b^2) = O\left(\frac{\log n}{\log\log n}\right)$ times.

Let us classify the ineffective iterations into two categories: (a) when $r_i < h$, and (b) when $r_i \geq h$.

The number of ineffective iterations of category (a) is $O(\log h)$, since r_i doubles during each ineffective iteration.

By Lemma 6, in expectation, between two consecutive effective iterations, there can be only $O(1)$ ineffective iterations of category (b). Therefore, the expected number of ineffective iterations of category (b) is $O\left(\frac{\log n}{\log\log n}\right)$. This finishes the proof. ◀

▶ **Lemma 8.** *The expected space used by the algorithm is* $O(h)$.

Proof. Let Y be the number of ineffective iterations in which $r_i \geq h$. The space used is bounded by $r^* = \max_i r_i$, which is at most $h \cdot 2^Y$. It thus remains to show that $\mathbf{E}[2^Y] = O(1)$.

To this end, we consider the following probability exercise:

> Let t be an integer and $\rho \leq 1/(8t)$. Consider a sequence of independent tosses of a biased coin, where the probability of heads is ρ. Stop the process when we encounter t tails. Let H be the number of heads encountered. Show that $\mathbf{E}[2^H] = O(1)$.

It is straightforward to see that $\Pr[H = j] \leq \binom{t+j}{j}\rho^j$. If $j < t$, this probability is at most $(2t)^j\rho^j \leq 1/4^j$. If $j \geq t$, the probability is at most $(2j)^t\rho^j \leq 1/4^j$, since the function $f(x) = (2x)^t\rho^x \cdot 4^x$ is decreasing for $x \geq t$ and has value at most 1 at $x = t$. Thus, $\mathbf{E}[2^H] \leq \sum_j 2^j \cdot 1/4^j = O(1)$.

The result now follows, by associating heads with ineffective iterations and tails with effective iterations, where $t = O(\log_b n + b^2) = O\left(\frac{\log n}{\log\log n}\right)$ (from the proof of Lemma 7) and $\rho = e^{-\Omega(b)}$ (by Lemma 6). ◀

3 Extreme points in $\mathbb{R}^d$

In this section we build on the ideas used for the skyline algorithm to solve the extreme points problem. The following result is obtained.

▶ **Theorem 9.** *Let P be a set of n points in $\mathbb{R}^d$, and h be the number of extreme points. Then, with high probability, there is an $O(h \log^{O(1)} n)$ space and $O(\log^d n)$ pass algorithm to compute the extreme points.*

It suffices to focus on computing the extreme points on the upper hull of P (finding the extreme points on the lower hull is symmetric). Our algorithm for the extreme points problem will also work in two stages. In Stage I, we will use the expensive $O(nh)$-time algorithm mentioned in the Introduction and let it run for $O(\log n)$ passes. If $h < \log n$, then $O(\log n)$ passes will be enough to find all the extreme points. Otherwise, we go to Stage II.

Algorithm 2 Finding Extreme Points.

0. $M \leftarrow \emptyset$.

//Stage I to handle $h \leq \log n$.

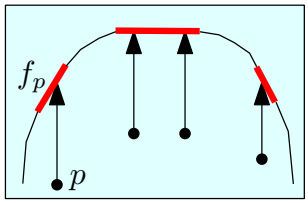
1. For the first $O(\log n)$ passes, run the expensive $O(nh)$-time algorithm.

//Stage II to handle $h > \log n$.

2. $P_1 \leftarrow P$, $r \leftarrow c \log n$ and $i \leftarrow 1$.
3. Repeat till $|P_i| < r$:
3a. Find a sample $R_i \subseteq P_i$ of size r. Set $R_i^+ \leftarrow \emptyset$.
3b. For each point $p \in R_i$: *//Finding new extreme points.*
- Shoot a vertical ray upwards from p to hit a facet $f_p \in conv(P)$.
- Add the $d+1$ vertices defining f_p into R_i^+.

3c. $M \leftarrow M \cup R_i^+$.
3d. $P_{i+1} \leftarrow P_i \setminus \{$points of P_i that are strictly below the upper hull of $R_i^+\}$.
3e. If $|P_{i+1}| \geq |P_i|/2$, then $r \leftarrow 2r$. *// Ineffective iteration.*
3f. $i \leftarrow i + 1$.

4. Output the extreme points of the $O(r)$ points in P_i.



Unlike the skyline algorithm, there is no notion of domination for the extreme points problem. Therefore, step 3b of the skyline algorithm cannot be used here. Instead, we perform the following operation: from each point $p \in R_i$, shoot a vertical ray upwards to hit a facet $f_p \in conv(P)$, where $conv(P)$ is the convex hull of P. This operation reduces to linear programming on the dual halfspaces of P. There is a known multi-pass streaming algorithm of Chan and Chen [5] which can solve a linear program in any constant dimension d using $O(\log^{O(1)} n)$ space and $O(\log^{d-1} n)$ passes. We can execute all the r linear programming queries *simultaneously*. This will not hurt the number of passes, but instead increase the space to $O(r \log^{O(1)} n)$. At the end of step 3b, we ensure that the upper hull of R_i^+ "covers" the points in R_i.

Analysis. The analysis follows the same steps as in our analysis of the skyline algorithm. The space used is $O(h \log^{O(1)} n)$, and since $O(\log n)$ iterations are performed, the total number of passes required are $O(\log^d n)$.

To prove an equivalent statement as Lemma 3, let M^* be the extreme points on the upper hull of P and let $b \leftarrow 2$. Fix a subset $A \subseteq M^*$, where the number of points of P_i above the upper hull of A is more than $|P_i|/2$. If any point $p \in P_i \setminus A$ above the upper hull of A is chosen to be in R_i, then p will be "covered" by the upper hull of R_i^+, making $R_i^+ = A$ impossible. So, the same argument as before shows $\Pr[R_i^+ = A] \leq e^{-\Omega(ch)}$. Thus, as before, the probability that the number of points of P_i above the upper hull of R_i^+ is more than $|P_i|/2$ is at most $n^{-\Omega(c)}$.

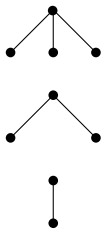
Running time and trade-offs. Each pass in Stage II can be implemented in $O(nh \log^{O(1)} n)$ time, since with Chan and Chen's algorithm [5], the r linear programs take $O(nr \log^{O(1)} n)$ time. Recall that P_i is represented implicitly; in each pass, we can test whether a point p is in P_i by testing whether p is covered by the upper hull of M, which reduces to solving a linear program on $O(h)$ points. The extra cost is $O(nh)$ per pass. The total time is thus $O(nh \log^{O(1)} n)$. (In the traditional non-streaming setting, the total running time is actually $O(nh)$, as it can be bounded by a geometric series.)

As before, it is possible to adapt the algorithm to achieve a trade-off, with $O(b^{O(1)} h \log^{O(1)} n)$ space and $O((\log_b n)^{O(1)})$ passes for a parameter b, since Chan and Chen's multi-pass linear programming algorithm [5] supports a trade-off. For example, setting $b = n^{\Theta(\delta)}$ gives $O(hn^{\delta})$ space and $O((1/\delta)^{O(1)})$ passes.

4 Why randomized algorithms?

We finish by proving that there does *not* exist any efficient deterministic algorithm for the problem of finding maximal elements of a poset. This justifies the use of randomization in the paper (at least for the poset problem). Our lower bound proof is based on a new and self-contained adversarial argument.

▶ **Theorem 10.** *Let $h = \Omega(1)$ and $p \cdot h \ll n$, where p is the number of passes made by an algorithm. Assume that the only operations on the input elements are pairwise comparisons. Then any deterministic algorithm which uses $O(h)$ space has to perform $p \geq \frac{h}{3} + 1$ passes to decide whether the number of maximal elements in a poset is $h + 1$, or $h + 2, \ldots$, or $h + 6$.*



Let P be the elements in our partially ordered set (poset). The queries asked by the algorithm will be of the form $q(a, b)$, where $a \in P$ is currently stored in the memory and $b \in P$ is the current element in the stream. The response of the adversary will either be $a \succ b$ which implies a *dominates* b, or $b \succ a$ which implies b dominates a, or $a \not\sim b$ which implies a and b are *incomparable*. The responses of the adversary has be *consistent*, i.e., once it responds to a query $q(a, b)$, then the answer to it cannot change later. Before the algorithm begins, the adversary will maintain that all the elements are incomparable. Each time, after seeing $n/3$ elements in the stream, the adversary will create dominance relationship between some pairs of elements by revealing a two-level tree (examples of two-level trees shown in

the figure on the right), where the root element dominates its child elements. Therefore, the root of each tree is a maximal element, and hence, the number of maximal points in P will be equal to the number of trees constructed by the adversary.

If an element belongs to a tree revealed till now by the adversary, then it will be labelled *locked*; otherwise, it will be labelled *unlocked*. We will show that if the number of passes performed by the algorithm is less than or equal to $h/3$, then the adversary can arrange the unlocked elements in at least two consistent ways, each having different number of maximal elements. This implies that the execution of the algorithm is exactly the same for two different inputs, which is a contradiction. Now we will present the technical details.

Adversary's strategy. We will need a couple of definitions to set up adversary's strategy. A *time-unit* corresponds to processing a single element in the stream. Each pass is divided into three *phases*, with each phase lasting $n/3$ time-units. For a p-pass algorithm, this naturally leads to a labelling of the phases as $1, 2, 3, \ldots, 3p-2, 3p-1, 3p$. At the end of the i-th phase, a tree T_i is created by the adversary.

The elements are partitioned into three equal-sized slabs: slab 0 consists of the first $n/3$ elements in the stream, slab 1 consists of the middle $n/3$ elements in the stream, and slab 2 consists of the last $n/3$ elements in the stream. Before the algorithm begins, all the elements are called *short-lived*, and if at any point an element remains in memory continuously for $n/3$ time-units, then we start calling it *long-lived*.

Now we are ready to describe the construction of a tree T_i. The dominated elements in T_i will be those elements in slab $(i-2) \bmod 3$ which were short-lived till the end of the $(i-1)$-th phase, but became long-lived at the end of the i-th phase. Next, we describe the strategy for picking the maximal element of T_i. The adversary will arbitrarily pick *one* among all the elements which satisfy the following conditions. The element should

1. belong to slab ($i \bmod 3$),
2. not belong to any of the trees already constructed, and
3. not be present in the memory at the end of the previous phases.

The reason for imposing these conditions will become clear in the proof of Lemma 12.

▶ **Lemma 11.** *There always exists an element which satisfies the above conditions. In fact, at least $n/6$ elements in a slab satisfy the above conditions.*

Proof. We claim that the number of long-lived elements are $O(ph)$. The key observation is that for an element to become long-lived, it has to be stored in memory at the end of at least one phase. Since the number of phases are $O(p)$, there can be at most $O(ph)$ long-lived elements. Therefore, the number of elements of slab $i \bmod 3$ which belong to the trees already constructed are $O(ph) + O(h) \ll n/12$. Also, the number of elements in slab $i \bmod 3$ which are present in the memory at the end of any phase is $O(ph) \ll n/12$. Since slab $i \bmod 3$ consists of $n/3$ elements, there will be at least $n/3 - n/6 = n/6$ elements satisfying the above conditions. ◀

When the algorithm asks a query $q(a, b)$, the adversary reports $a \succ b$ or $b \succ a$ if that relation holds in any of the trees constructed till now; otherwise it reports $a \not\sim b$. Next, we argue that the responses of the adversary to the queries are consistent.

▶ **Lemma 12.** *If the adversary places a relation $a \succ b$ in the poset, then the algorithm must not have asked the query $q(a, b)$ or $q(b, a)$ till then. This ensures that responses of the adversary are consistent.*

Proof. Without loss of generality, assume that the element b is in slab 0 and the element a in slab 2 (the other cases can be handled symmetrically). Note that this satisfies the condition that an element from slab $(i-2) \bmod 3$ is dominated only by an element from slab $i \bmod 3$. For a query $q(a,b)$ to be asked during the j-th pass, a should be stored in memory at the beginning of the j-th pass. This implies that a is stored in memory at the end of the $3(j-1)$-th phase, which violates condition 3 for picking the maximal element.

Now we prove that the query $q(b,a)$ was not asked. In a given pass, at the end of which phase does b newly become long-lived? It turns out to be the end of the second phase. Then, let j be the smallest index such that at the end of the second phase in the j-th pass, b was still in memory. This is when $a \succ b$ will be created by the adversary, since b has newly become long-lived. Now, if $q(b,a)$ was asked (say, in the i-th pass) before $a \succ b$ was placed, then b should be in memory at the end of the second phase of the i-th pass and in fact, it should be in memory in the third phase of the i-th pass till a is processed. This implies that b becomes long-lived in the i-th pass, which contradicts that j is the smallest index. ◀

Handling unlocked elements. Since each tree corresponds to revealing only one maximal element, this strategy of the adversary will ensure that the algorithm is forced to perform $h/3$ passes at which point h maximal elements will be revealed. If further passes are not performed, then the algorithm will have the same outcome for more than two inputs. The details follow next.

Let b, b' be any two unlocked elements in slab 2 which satisfy the three conditions stated above for being a maximal element (by Lemma 11 we know at least two such elements in slab 2 still exist). Also, observe that all the unlocked elements in slab 0 are short-lived and by using an argument similar to Lemma 12, it can be shown that queries of the form $q(b,a)$ or $q(a,b)$ or $q(b',a)$ or $q(a,b')$ were not asked by the algorithm, where a is a short-lived unlocked element in slab 0. As a result, the adversary will be consistent if it declares that the unlocked elements in slab 0 are dominated by either b or b'. Now the adversary has two choices: (i) either make only b or b' the root of a tree and all the unlocked elements in slab 0 the leaves of that tree, or (ii) make two trees with b and b' as the root of those trees, and the unlocked elements in slab 0 are partitioned to be the leaves of the two trees.

A similar argument holds when maximal elements are chosen from slab 0 and slab 1. Therefore, the number of maximal elements can be anywhere in the range $(h, h+6]$, if the number of passes performed are at most $h/3$.

▶ Remark. The assumption that $p \cdot h \ll n$ is needed, since for large h, there is a trivial deterministic algorithm with $O(h)$ space and $O(n/h)$ passes (we can divide the input sequence into $O(n/h)$ blocks of h elements, and in the i-th iteration, load the i-th block in memory and test which of the h elements in the block are maximal).

References

1. Peyman Afshani. Fast computation of output-sensitive maxima in a word RAM. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1414–1423, 2014.
2. Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. Instance-optimal geometric algorithms. *Journal of the ACM*, 64(1):3:1–3:38, 2017.
3. Timothy M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete & Computational Geometry*, 16(4):369–387, 1996.
4. Timothy M. Chan. Improved deterministic algorithms for linear programming in low dimensions. *ACM Trans. Algorithms*, 14(3):30:1–30:10, 2018. doi:10.1145/3155312.

5 Timothy M. Chan and Eric Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007. doi:10.1007/s00454-006-1275-6.

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

7 Timothy M. Chan, Jack Snoeyink, and Chee-Keng Yap. Primal dividing and dual pruning: Output-sensitive construction of four-dimensional polytopes and three-dimensional Voronoi diagrams. *Discrete & Computational Geometry*, 18(4):433–454, 1997.

8 Kenneth L. Clarkson. More output-sensitive geometric algorithms. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 695–702, 1994.

9 Kenneth L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM*, 42(2):488–499, 1995. doi:10.1145/201019.201036.

10 Kenneth L. Clarkson and Peter W. Shor. Application of random sampling in computational geometry, II. *Discret. Comput. Geom.*, 4:387–421, 1989. doi:10.1007/BF02187740.

11 Atish Das Sarma, Ashwin Lall, Danupon Nanongkai, and Jun (Jim) Xu. Randomized multi-pass streaming skyline algorithms. *PVLDB*, 2(1):85–96, 2009. doi:10.14778/1687627.1687638.

12 Martin Farach-Colton, Meng Li, and Meng-Tsung Tsai. Streaming algorithms for planar convex hulls. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 47:1–47:13, 2018. doi:10.4230/LIPIcs.ISAAC.2018.47.

13 Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 58–66, 2001.

14 Xiaocheng Hu, Cheng Sheng, Yufei Tao, Yi Yang, and Shuigeng Zhou. Output-sensitive skyline algorithms in external memory. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 887–900, 2013.

15 J.H. Dula J and R.V. Helgason. A new procedure for identifying the frame of the convex hull of a finite collection of points in multidimensional space. *European Journal of Operational Research*, 92(2):352–367, 1996.

16 David G. Kirkpatrick and Raimund Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 89–96, 1985. doi:10.1145/323233.323246.

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

18 Ketan Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms*. Prentice Hall, 1994.

19 Thomas Ottmann, Sven Schuierer, and Subbiah Soundaralakshmi. Enumerating extreme points in higher dimensions. *Nordic Journal of Computing*, 8(2):179–192, 2001.

20 F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer–Verlag, 1985.

21 Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6:423–434, 1991. doi:10.1007/BF02574699.

22 Cheng Sheng and Yufei Tao. On finding skylines in external memory. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 107–116, 2011. doi:10.1145/1989284.1989298.

A A deterministic algorithm for skyline points in $\mathbb{R}^d$

In this appendix, we briefly describe a deterministic algorithm to compute skyline points in $\mathbb{R}^d$ using $O(h \log n)$ space and $O(\log^{d-1} n)$ passes (as we have mentioned in the introduction). Though we are not aware of an explicit reference of this result, it follows from a straightforward adaptation of Kirkpatrick and Seidel's output-sensitive divide-and-conquer algorithm [16], but reimplemented in the multi-pass setting (analogous to Chan and Chen's multi-pass reimplementation [5] of Kirkpatrick and Seidel's output-sensitive 2-d convex hull algorithm [17]).

Given point sets P and M in $\mathbb{R}^d$, let $P \ominus M$ denote the "filtered" subset of all points $p \in P$ that are not dominated by any points in M. Let $p\downarrow$ denote the projection of p onto the first $d-1$ coordinates, and let $P\downarrow = \{p\downarrow : p \in P\}$.

Below is a variant or reinterpretation of Kirkpatrick and Seidel's algorithm for computing the skyline of $P \ominus M$ (initially, we set $M = \emptyset$):

Algorithm 3 $\mathbf{Skyline}_d(P, M)$.

1. If $|P \ominus M| \le 1$, then return $P \ominus M$.
2. Partition P into the left and the right halves P_ℓ and P_r using an approximate median d-th coordinate.
3. Compute $M_r = \mathbf{Skyline}_{d-1}((P_r \ominus M)\downarrow, \emptyset)$. Add $\{p : p\downarrow \in M_r\}$ to M.
4. Return $\mathbf{Skyline}_d(P_\ell, M) \cup \mathbf{Skyline}_d(P_r, M) \cup \{p : p\downarrow \in M_r\}$.

(In the original algorithm, points dominated by M_r are pruned from P_ℓ before recursion. With the filtering operation $\ominus$, explicit pruning is avoided.)

In the multi-pass setting, we will execute the recursion level by level. The recursion tree for $\mathbf{Skyline}_d$ has $O(\log n)$ levels. We maintain one global set M and do filtering with respect to this global set M (this does not affect correctness); the size of the set is $O(h)$. Consider the next level of the tree. There are at most $O(h)$ nodes in the level. Each subset P can be encoded by an interval in the d-th coordinate. As we make a pass over the input and encounter a point p, we can identify the subset P containing p, and test whether it is in $P \ominus M$ by checking whether it is dominated by any point in M (in $O(h)$ time naively, or in polylogarithmic time by storing M in an orthogonal range searching data structure). The approximate median computation in step 2 can be done by a known one-pass, $O(\log n)$-space algorithm of Greenwald and Khanna [13]. All $O(h)$ invocations to this approximate median algorithm are done simultaneously, and so the total space used is $O(h \log n)$. Step 3 invokes a $(d-1)$-dimensional skyline algorithm. Again, these invocations are done simultaneously; the total output size in these calls is $O(h)$.

Let $P_d(n)$ be the number of passes in our d-dimensional skyline algorithm, and let $s_d(n)$ be the space used per output point (i.e., the total space is $h \cdot s_d(n)$). Then

$$P_d(n) = O(\log n) \cdot (P_{d-1}(n) + O(1)) \qquad \text{and} \qquad s_d(n) = s_{d-1}(n) + O(\log n).$$

With the base case $P_1(n) = 1$ and $s_1(n) = O(1)$, we get $P_d(n) = O(\log^{d-1} n)$ and $s_d(n) = O(\log n)$ as desired.