The most-likely skyline problem for stochastic points

Akash Agrawal, Yuan Li, Jie Xue, Ravi Janardan

1 Introduction

In \( \mathbb{R}^d \), a point \( u \) dominates a point \( v \) if each coordinate of \( u \) is at least as large as the corresponding coordinate of \( v \), with strict inequality in at least one dimension. The skyline of a set of points consists of the subset of all points where no point is dominated by any other point of the set. (See Figure 1a.) The skyline (or Pareto set or maximal vector) is useful in multi-criteria decision-making as it yields a set of viable candidates for further exploration. It has been well-studied in the database, optimization, and computational geometry literature; e.g., [3, 5, 6].

We investigate skylines in a setting where there is uncertainty associated with the existence of the points. Such stochastic datasets can model, for instance, experimental observations with associated confidence values or physical entities that may not always be available (e.g., sensors whose activity level depends on battery life or hotel rooms where availability depends on demand).

We consider the problem of computing the skyline that has the greatest probability of being present, hence the one that the user is most likely to encounter and explore further. We call this the most-likely skyline. Our results include an optimal algorithm in the plane and hardness results in higher dimensions.

1.1 Problem formulation, contributions, and related work

Let \( O = \{o_1, o_2, \ldots, o_n\} \) be a set of points in \( \mathbb{R}^d \), where \( x_k(o_i) \) denotes the \( k \)th coordinate of \( o_i \). Point \( o_i \) dominates \( o_j \) (i.e., \( o_i > o_j \)) if \( x_k(o_i) \geq x_k(o_j) \) for \( 1 \leq k \leq d \), with strict inequality in at least one dimension.

Suppose that each \( o_i \in O \) has an associated real \( p_i \in (0, 1] \). (The \( p_i \)'s are known and independent of each other.) We call \( p_i \) (resp. \( q_i = 1 - p_i \)) the existence (resp. non-existence) probability of \( o_i \) and call \( O \) a stochastic set.

Let \( O' \subseteq O \), where no point of \( O' \) dominates another of \( O' \); thus, \( O' \) itself is the skyline of \( O' \). Now, when is \( O' \) also a skyline of \( O' \)? Let \( F(O') \subseteq O' \) be the points that are not dominated by any point of \( O' \); intuitively, these are the points “above” the staircase contour defined by \( O' \). Clearly, as long as no point of \( F(O') \) is present, \( O' \) is also the skyline of \( O \). (The points of \( O \setminus O' \) that are dominated by one or more points of \( O' \), i.e., the ones “below” the staircase, do not affect the skyline property of \( O' \).) Thus, for \( O' \) to be a skyline of \( O \), each point of \( O' \) must be present and no point of \( F(O') \) should be present. So, the probability that \( O' \) is a skyline of \( O \) is \( \text{PrSky}(O') = \prod_{o_i \in O'} p_i \times \prod_{o_i \notin F(O')} q_i \).

Our problem is to compute a skyline \( O' \) of \( O \) for which \( \text{PrSky}(O') \) is maximum. This skyline, called the most-likely skyline of \( O \), is denoted by \( \text{MLSky}(O) \). (See Figure 1b and Figure 1c for an example.)

Note that the introduction of uncertainty makes our problem challenging as there might be an exponential number of candidate skylines—as many as one for each possible subset of existent points. By contrast, in the non-stochastic setting, there is exactly one skyline for a given set of points.

We make three contributions to the most-likely skyline problem. We prove that computing such a skyline is \( \text{NP} \)-hard in \( \mathbb{R}^3 \), hence also in \( \mathbb{R}^d \) for \( d \geq 3 \) (Section 2). Furthermore, we prove that the most-likely skyline in \( \mathbb{R}^d \) (\( d \geq 3 \)) cannot even be well-approximated in polynomial-time unless \( P = \text{NP} \) (Section 3). We complement these results with an \( O(n \log n) \)-time and \( O(n) \)-space algorithm to compute the most-likely skyline in \( \mathbb{R}^2 \) (Section 4), which is optimal in the comparison model due to the known \( \Omega(n \log n) \) lower bound for the non-stochastic skyline problem [5].

To our knowledge, this paper is the first to consider skylines in the unipoint stochastic model, where
the points have fixed locations and associated existence probabilities. An alternative setting is the multipoint stochastic model, where each point is described by discretely-many locations, with associated existence probabilities, or by a continuous probability distribution. Examples of work here include computing skylines whose points have existence probabilities above a threshold \([7]\), computing for each dataset (or for a query point) the probability that it is not dominated by any other point \([1, 2]\), computing stochastic skyline operators to find a minimum set of candidate points with respect to a certain scoring function \([9]\), etc.

2 NP-Hardness of computing the most-likely skyline in \(\mathbb{R}^d, d \geq 3\)

We give a polynomial-time reduction from the minimum \(\varepsilon\)-ADR problem in \(\mathbb{R}^3\), which is known to be NP-hard \([4]\), to the most-likely skyline problem in \(\mathbb{R}^3\). (Here ADR stands for “Approximately Dominating Representatives” \([4]\).

An instance of the \(\varepsilon\)-ADR problem in \(\mathbb{R}^3\) consists of a set, \(S\), of \(n\) (non-stochastic) points and a real \(\varepsilon > 0\). An \(\varepsilon\)-ADR of \(S\) is a set \(S' \subseteq S\) such that every \(s \in S\) is dominated by some \(s' \in S'\) when \(s'\) is boosted by \(\varepsilon\), i.e., \((1+\varepsilon)\cdot s' > s\). The minimum \(\varepsilon\)-ADR problem seeks the smallest such set \(S'\).

The reduction: Given any \(\varepsilon\)-ADR instance in \(\mathbb{R}^3\), we compute the (conventional) skyline, \(Sky(S)\), of \(S\) and boost its points by \(\varepsilon\) to get a set \(\hat{S}\). To each point in \(Sky(S)\) (resp. \(\hat{S}\)) we assign an existence probability \(\beta\) (resp. \(\alpha\)), where \(1/3 < \alpha < 1/2 < \beta < 1\) and \(\beta(1-\alpha) < \alpha\); e.g., \(\alpha = 0.4\) and \(\beta = 0.6\). The set \(\hat{S} = Sky(S) \cup \hat{S}\) is an instance of the most-likely skyline problem in \(\mathbb{R}^3\).

The reduction takes polynomial time. We observe the following:

(i) The probability that the skyline of \(\hat{S}\) is empty is the probability that no point of \(\hat{S}\) exists, i.e., \((1 - \beta)^K(1 - \alpha)^K\), where \(K = |Sky(S)| = |\hat{S}|\). Since \(\beta > 1/2\), the probability of the empty skyline is less than \(\beta^K(1 - \alpha)^K\). The latter is the probability of that skyline of \(\hat{S}\) where the points of \(Sky(S)\) exist and those of \(\hat{S}\) do not. Thus, the most-likely skyline of \(\hat{S}\) is non-empty.

(ii) Consider the skyline of \(\hat{S}\) that consists of just \(Sky(S)\). Suppose that a point \(s \in Sky(S)\) is replaced by a point \(\hat{s} \in \hat{S}\) that dominates it. The probability expression for \(Sky(S)\) contains terms \(\beta\) and \(1 - \alpha\), since \(s\) is present in it and \(\hat{s}\) is not. In the probability expression for the new skyline, the term \(1 - \alpha\) is replaced by \(\alpha\), since \(\hat{s}\) is present and the term \(\beta\) is excluded since \(s\) is not present. (The term \(\beta\) is not replaced by \(1 - \beta\) since \(s\) is dominated by \(\hat{s}\). Indeed, \(\hat{s}\) may also dominate other points of \(Sky(S)\), so the term \(\beta\) for each such point is also excluded.) Since \(\beta(1 - \alpha) < \alpha\), the new skyline has a higher probability than the previous skyline of \(\hat{S}\). The replacement process is continued until a skyline consisting of only points drawn from \(\hat{S}\) is obtained. Since each replacement yields a skyline of higher probability, it follows that the most-likely skyline of \(\hat{S}\) consists of only points from \(\hat{S}\).

(iii) The points of \(\hat{S}\) are mutually non-dominating since their pre-images are in \(Sky(S)\). Thus, each point of \(\hat{S}\) that is not in the most-likely skyline of \(\hat{S}\) con-
tributes \((1 - \alpha)\) to the probability of this skyline, so its probability is \(\alpha^k(1 - \alpha)^{K-k} = (\alpha/(1 - \alpha))^k(1 - \alpha)^K\).

Since \(\alpha < 1/2\), we have \(\alpha/(1 - \alpha) < 1\). Since \((1 - \alpha)^K\) is fixed for a given set \(S\) and the skyline under consideration has maximum probability, it follows that \(k\) is minimum.

Suppose that there is a polynomial-time algorithm for the most-likely skyline problem in \(\mathbb{R}^3\). We generate \(S\) and compute its most-likely skyline, all in polynomial time. These skyline points, which are a subset of \(\hat{S}\) of some minimum size \(k\), dominate the points of \(Sky(S)\), hence also the points of \(S\). Therefore, the pre-images of these skyline points in \(S\) (i.e., prior to boosting) are an \(\varepsilon\)-ADR of \(S\) of minimum size \(k\) and can be computed in polynomial time. This contradicts the known NP-hardness of the \(\varepsilon\)-ADR problem in \(\mathbb{R}^3\) and yields the following theorem.

**Theorem 1** The most-likely skyline problem in \(\mathbb{R}^d\), \(d \geq 3\), is NP-hard.

### 3 Inapproximability in \(\mathbb{R}^d\), \(d \geq 3\)

Let \(A\) be an algorithm that computes a skyline whose probability is greater than \(c\) times the probability of the most-likely skyline, \(0 < c < 1\). We call \(A\) a \(c\)-approximation algorithm. Our first result is based on the reduction in Section 2.

**Theorem 2** For \(c > 1/2\), there exists no polynomial-time \(c\)-approximation algorithm, \(A\), for the most-likely skyline problem in \(\mathbb{R}^d\), \(d \geq 3\), unless \(P = NP\).

**Proof.** We show that \(A\) cannot exist for \(c = \alpha/(1 - \alpha)\), where \(1/3 < \alpha < 1/2\). (Recall that \(\alpha\) is the existence probability assigned to each point of \(\hat{S}\) in Section 2.) The theorem follows since \(\alpha > 1/3\) implies \(c > 1/2\). (A similar result is stated in [8] for the most-likely convex hull problem.)

Suppose \(A\) exists for \(c = \alpha/(1 - \alpha)\). We run \(A\) on \(\hat{S} = Sky(S) \cup \hat{S}\). The probability of the resulting skyline is greater than \(\alpha/(1 - \alpha)\) times the probability of the most-likely skyline of \(\hat{S}\), i.e., greater than \((\alpha/(1 - \alpha)) \cdot (\alpha^k(1 - \alpha))^{K-k} = \alpha^{k+1}(1 - \alpha)^{K-(k+1)}\). Assume without loss of generality that the computed skyline contains points from \(S\) only. (If it contains points of \(Sky(S)\), then each of these can be replaced, in polynomial time, by the corresponding boosted point in \(\hat{S}\) and the probability of the resulting skyline only increases.)

How many points does the computed skyline have? Note that a skyline with \(k+1\) points of \(\hat{S}\) has probability \(\alpha^{k+1}(1 - \alpha)^{K-(k+1)}\). For each additional point of \(\hat{S}\) that is included in the skyline, the probability gets multiplied by a factor \(\alpha/(1 - \alpha)\), which is less than 1 since \(\alpha < 1/2\). It follows that the skyline computed by \(A\) has fewer than \(k+1\) points of \(\hat{S}\). Thus, the \(\varepsilon\)-ADR of \(S\) corresponding to the pre-images of the points of the skyline computed by \(A\) has fewer than \(k+1\) points. This \(\varepsilon\)-ADR of \(S\) must be a minimum \(\varepsilon\)-ADR of \(S\), since the latter has size \(k\). This yields a polynomial-time algorithm for computing a minimum \(\varepsilon\)-ADR, which is not possible unless \(P = NP\).

We can use Theorem 2 and the notion of “product compositability” to show that, for any \(\delta > 0\), there is not even a polynomial-time \(2^{-\varepsilon(n - \delta)}\)-approximation algorithm for the most-likely skyline problem in \(\mathbb{R}^d\), \(d \geq 3\), unless \(P = NP\).

An optimization problem is product composable [8] if any given set of problem instances \(I_1, \ldots, I_k\) can be combined to yield a new instance \(I^*\) whose objective function is expressible as the product of the objective functions of \(I_1, \ldots, I_k\). We require that \(|I^*| = \sum_{i=1}^{k} |I_i|\), that \(I^*\) is constructible in time polynomial in \(|I^*|\), and that there exists a polynomial-time computable bijection between the set of feasible solutions of \(I^*\) and those of \(I_1, \ldots, I_k\).

The following lemma relates product compositability to inapproximability.

**Lemma 3** ([8]) If a maximization problem of size \(n\) is product composable and cannot be approximated within a constant \(c < 1\) in polynomial time, then it has no polynomial-time \(2^{-\varepsilon(n - \delta)}\)-approximation algorithm, for any \(\delta > 0\).

A proof of this lemma can be found in Appendix G of [8] (full version).

Intuitively, the lemma is proved by showing that the existence of a \(2^{-\varepsilon(n - \delta)}\)-approximation algorithm together with product compositability would imply the existence of a \(c\)-approximation algorithm. Recall that Theorem 2 has established that, for \(1/2 < c < 1\), no \(c\)-approximation algorithm exists for the most-likely skyline problem, unless \(P = NP\). In fact, the proof of Theorem 2 shows that this is true even for the subset of instances consisting of the set \(\hat{S} = Sky(S) \cup \hat{S}\) and the associated probabilities, as defined in Section 2.

So it suffices to show that the most-likely skyline problem consisting of instances \(S = Sky(S) \cup \hat{S}\) and the aforementioned probabilities is product composable. Specifically, we form the instances \(I_1, \ldots, I_k\) by partitioning \(\hat{S}\) using the \(k\) points on its most-likely skyline.

Let \(S' = \{s_1, \ldots, s_k\} \subseteq \hat{S}\) be the points on the most-likely skyline of \(\hat{S}\), sorted by non-increasing \(x_1\) coordinates. For each \(s_i \in S'\), we define two sets \(S_i\) and \(\hat{S}_i\). \(S_i\) contains points \(s_j \in Sky(S)\) such that \(s_i > s_j\) or \(s_i = s_j\). \(\hat{S}_i\) contains points \(s_j \in Sky(S)\) such that \(s_i > s_j\) or \(s_i = s_j\) and \((s_i, s_j) \not\in S'\). \(\hat{S}_i\) contains the boosted points of \(S_i\). For \(1 \leq i \leq k\), let \(I_i = S'_i \cup \hat{S}_i\). (See Figure 2.) It is easy to verify that \(I_1, \ldots, I_k\) can be combined to form a new instance, \(I^*\), in polynomial-time such that,
Theorem 4, there is no polynomial-time 0 time, and vice-versa. This establishes product composi-

tion of most-likely skyline of points, in 1,1), for a set S. Our algorithm runs in n n space, n n) space,

which is optimal in the comparison model.

Figure 2: Example of a partition of S to form the problem instances I1, I2, and I3. The points in Sky(S) are

represented by crosses (×) and the boosted points, i.e., points in S, are represented by disks (○). Points on the

most-likely skyline are circled.

given the most-likely skyline for I1, . . . , I k, the most-likely skyline for I* can be computed in polynomial-
time, and vice-versa. This establishes product composable. Lemma 3 now yields the following result.

Theorem 4 For any δ > 0, there is no polynomial-time 2−O(δlog δ)-approximation algorithm for computing the

most-likely skyline of n stochastic points in R d, d ≥ 3, unless P = NP.

Finally, we note that there is a simple, but uninteresting, polynomial-time 2−na-approximation algorithm for

the most-likely skyline problem: Simply compute the skyline of the points of S whose existence probability is

more than 1/2. (A similar observation appears in [8] for the most-likely convex hull problem.)

4 An efficient algorithm in R2

We now describe an algorithm to compute the most-likely skyline, MLSky(O), for a set O of n points, in R2.

Our algorithm runs in O(n log n) time and O(n) space, which is optimal in the comparison model.

We assume w.l.o.g. that all points of O are in the first quadrant and that no two points have the same x1-
or x2-coordinate. Let the points of O be o1, o2, . . . , on, in decreasing order of their x1-coordinates. For conve-
nience, we augment O with dummy points o0 = (∞, 0) and on+1 = (0, ∞), with p0 = p on+1 = 1. The proof of

the following lemma is fairly obvious, hence omitted.

Lemma 5 For any subset O′ of O, O′ = MLSky(O) iff O′ ∪ {o0, on+1} = MLSky(O ∪ {o0, on+1}).

Based on Lemma 5, we augment the input set with o0 and on+1, and, hereafter, we will focus on finding the

most-likely skyline for {o0, o1, . . . , on, on+1}. For notational convenience, let Oij = {o0, o1, . . . , oj}.

We sweep a vertical line from right to left over O, stopping at each point oj. At oj we compute the most-

likely skyline of Oj subject to the constraint that oj belongs to this skyline. We denote this optimal skyline by

S(oj). We initialize S(o0) = {o0} and report S(oj+1) = {o0, . . . , on+1} as MLSky(O).

Let F(S(oj)) be the set of points of Oj \ S(oj) that are not dominated by any points in S(oj). Then, the

probability of S(oj) being the skyline of Oj is

PrSky(S(oj)) = \prod_{o_k \in S(o_j)} p_k \times \prod_{o_k \in F(S(o_j))} q_k.

Let oj be the point in S(oj) with largest x2-coordinate smaller than x2(oj), and let R(oj) = (x1(oj), x1(2oj)) ×

(x2(oj), ∞). (Figure 3.) Let F(R(S(oj))) be the set of points of F(S(oj)) lying in R(oj, oj). Then,

PrSky(S(oj)) = p_j \times \left( \prod_{o_k \in F(R(S(o_j)))} q_k \times \prod_{o_k \in S(o_j \setminus \{o_j\})} p_k \times \prod_{o_k \in F(S(o_j) \setminus F(R(S(o_j))))} q_k \right).

Thus, we can write

PrSky(S(o_j)) = p_j \times \text{score}_o(o_j),

where \text{score}_o(o_j) is the expression inside the large parentheses in the above equation.

Figure 3: Illustrating S(o_j) and R(o_j, o_j). (Dummy points o0 and o on+1 are not shown.)

Since p_j is fixed for o_j and PrSky(S(o_j)) is maximum, it follows that \text{score}_o(o_j) must be maximum. For

the given pair (o_j, o_j), the first term in \text{score}_o(o_j), i.e., \left( \prod_{o_k \in F(R(S(o_j)))} q_k \right), is fixed. Thus, for \text{score}_o(o_j) to be maximum, the product of the second and third terms must be maximum. This product is nothing but the probability of the most-likely skyline of Oj subject to the constraint that oj belongs to the skyline, i.e., PrSky(S(oj)). Hence, \text{score}_o(o_j) = \left( \prod_{o_k \in F(R(S(o_j)))} q_k \right) \times PrSky(S(o_j)).
Thus the recurrence for $S(o_i)$ is:

$$S(o_i) = \begin{cases} 
    \{o_i\}, & \text{if } i = 0, \\
    \{o_i\} \cup S(o_j), & \text{otherwise},
\end{cases}$$

where $o_j = \max_{o_j \in O_{i-1}; x_2(o_i) > x_2(o_j)} \{\text{score}_o(o_j)\}$.

This recurrence leads naturally to a dynamic programming algorithm that can be implemented easily to run in $O(n^2)$ time, where the run time is dominated by time to maintain the first term, i.e., $(\prod \ q_k)$, in $\text{score}_o(o_j)$ for all relevant pairs $(o_i, o_j)$. The run time can be improved to $O(n \log n)$ through a more careful approach, as follows.

When the sweep is at $o_i$, we will, for the sake of brevity, refer to $(\prod \ q_k)$, $\text{score}_o(o_j)$, and $\alpha_k \in \mathcal{F}_n(S(o_i))$ $\text{PrSky}(S(o_j))$ as the $R$-value, $S$-value, and $P$-value of $o_j$, respectively. Note that the $S$-value of $o_j$ is the product of the $R$-value and the $P$-value of $o_j$. Note also that the $R$-value and $S$-value of $o_j$ depend on $o_i$, too, but since we refer to these when the sweep is at $o_i$, we omit the reference to $o_i$. (Similarly, if the sweep is later at some other point.)

Just after $o_i$ is processed in the right-to-left sweep, let $o$ be some point of $O$ for which we wish to compute $S(o)$. Let $o_j$ be any point of $O$ to the right of and below $o_i$. Then, if $o$ is above $o_j$ then $o_j$ will be in the rectangle $R(o, o_j)$ and so $q_i$ will need to be included in the $R$-value of $o_j$ when the sweep reaches $o$. Thus, when we have finished processing $o_i$, we preemptively multiply by $q_i$ the $R$-value of each $o_j$ to the right of and below $o_i$. All such points $o_j$ would have already been encountered in the sweep and their $y$-coordinates will lie in the range $[0, x_2(o_i)]$, so the multiplication can be done for all $o_j$ in this range efficiently by grouping them into a small number of sets. This observation along with a suitable data structure is the key to realizing the improved run time.

Let $\mathcal{D}$ be a data structure on $O$ which supports the following operations when the sweepline is at $o_i$.

- $\text{FindMax}_S$-$S$-value$(o_i)$: Returns the maximum of the $S$-values associated with the points whose $y$-coordinates are in the range $[0, x_2(o_i)]$, along with the corresponding point $o_j$.

- $\text{Set}_P$-$P$-value$(o_i, \mu)$: Sets the $P$-value of $o_i$ to $\mu$. (In our algorithm, $\mu$ will be $p_t$ times the $S$-value returned by $\text{FindMax}_S$-$S$-value$(o_i)$, which is executed just before $\text{Set}_P$-$P$-value$(o_i, \mu)$.)

- $\text{RangeMult}_R$-$R$-value$(o_i)$: Multiplies by $q_i$ the $R$-values of the points whose $y$-coordinates are in the range $[0, x_2(o_i)]$.

Note that the range $[0, x_2(o_i)]$ used in $\text{FindMax}_S$-$S$-value$(o_i)$ and $\text{RangeMult}_R$-$R$-value$(o_i)$ may include points that are below and to the left of $o_i$, hence have not yet been seen in the sweep. However, $\mathcal{D}$ is set up so that the $P$-value (hence the $S$-value) of any point that has not yet been seen in the sweep is zero. Thus, such points are effectively ignored when the sweep is at $o_i$. This approach obviates the need to make $\mathcal{D}$ dynamic.

In Section 4.1 we show that $\mathcal{D}$ can be implemented so that it supports the above operations in $O(n \log n)$ time using $O(n)$ space. Given this it should be clear that the algorithm runs in $O(n \log n)$ time and $O(n)$ space as it involves doing at each point of $o_i \in O$ (other than at $o_i$), one $\text{FindMax}_S$-$S$-value$(o_i)$, one $\text{Set}_P$-$P$-value$(o_i, \mu)$, and one $\text{RangeMult}_R$-$R$-value$(o_i)$ operation, in that order. (After each $\text{FindMax}_S$-$S$-value$(o_i)$, $S(o_i)$ is updated by including $o_i$ in $S(o_i)$.) This leads to the following conclusion.

**Theorem 6** The most-likely skyline of a set of $n$ stochastic points in $\mathbb{R}^2$ can be computed in $O(n \log n)$ time using $O(n)$ space.

### 4.1 Implementing the data structure $\mathcal{D}$

We implement $\mathcal{D}$ as a 1-dimensional range tree, i.e., a balanced binary search tree where the $x_2$-coordinates of the points of $O$ are stored at the leaves, in increasing order from left to right. We maintain several fields at the nodes of $\mathcal{D}$, whose meanings are defined relative to the position of the sweep at the current point $o_i$.

During the sweep, we need to keep track of the $R$-value of each point $o_i$ that is to the right of and below $o_i$. Rather than doing this explicitly for each such $o_i$, which would be expensive, we accumulate the $R$-value for $o_i$ as the product of certain real numbers stored at the leaf containing $o_i$ and the ancestors of this leaf. Specifically, let $\text{prod}(v)$ be a real-valued field at any node $v$. Then the $R$-value of $o_i$ (relative to the current position of the sweep at $o_i$) is the product of the $\text{prod}(\cdot)$ fields at the leaf containing $o_i$ and its ancestors. (Note that when the sweep is at $o_i$, the $R$-value is irrelevant for points that are to the right of and above $o_i$, and undefined for points that are to the left of $o_i$.) We initialize $\text{prod}(v)$ to 1 for all nodes of $\mathcal{D}$.

At each leaf $v$, besides $\text{prod}(w)$, we store three additional fields $\text{pt}(v)$, $\text{pval}(v)$, and $\text{val}(v)$. Here $\text{pt}(v)$ is the point whose $x_2$-coordinate is stored at $w$, $\text{pval}(w)$ is the $P$-value of the point if it has already been seen in the sweep and zero otherwise, and $\text{val}(w)$ is $\text{prod}(w) \times \text{pval}(w)$. We initialize $\text{pval}(w)$ to 1 for the leaf $w$ corresponding to $o_0$ and to zero for all other leaves.

At each non-leaf $v$, besides $\text{prod}(v)$, we store two additional fields $\text{val}(v)$ and $\text{pt}(v)$, whose meanings are as follows: Let $\mathcal{D}(v)$ be the subtree of $\mathcal{D}$ rooted at $v$. Among
the leaves of $D(v)$, let $w$ be the one for which the product of $val(w)$ and the $prod(\cdot)$ fields at $w$ and its ancestors, up to and including $v$, is maximum. Then $val(v)$ stores this maximum product and $pt(v)$ equals $pt(w)$. Thus the maximum $S$-value among the leaves in $D(v)$ is the product of $val(v)$ and the $prod(\cdot)$ fields at the proper ancestors of $v$. (Note that any leaf in $D(v)$ corresponding to a point that has yet to be seen in the sweep cannot realize the maximum product as its $val(\cdot)$ field is zero.)

As we will see below, when searching downwards in $D$ during any of the aforementioned operations, if we are at a non-leaf node $v$ then we will multiply the $prod(\cdot)$ field and the $val(\cdot)$ field of each child of $v$ by $prod(v)$ and then reset $prod(v)$ to 1. This ensures that (a) at any node on the search path, the maximum $S$-value among the leaves in its subtree is equal to the node’s $val(\cdot)$, and (b) the value that was originally in $prod(v)$ will continue to be applied to the points in the subtree of each of $v$’s children. This as-needed, lazy approach to propagating the values in the $prod(\cdot)$ fields allows us to implement the operations efficiently.

We now describe how to do the operations on $D$.

- **FindMaxS-value($o_i$):** We search downwards in $D$ with $x_2(o_i)$ and identify a set, $C$, of canonical nodes, as follows: Whenever the search at a node $v$ goes to the right child, we include the right child $v'$ in $C$. Thus, the leaves of the $D(v')$’s yield a grouping of the points of $O$ lying in the range $[0, x_2(o_i)]$ into $O(\log n)$ subsets.

  During the search down $D$, when we are at a non-leaf node $v$, we update $prod(u)$ to $prod(u) \times prod(v)$ (resp. $val(u)$ to $val(u) \times prod(v)$) for each child $u$ of $v$, and then reset $prod(v)$ to 1. Finally, we return the maximum of the $val(v')$’s, taken over all nodes $v'$ in $C$.

- **SetP-value($o_i, \mu$):** We search downwards in $D$ with $x_2(o_i)$ to find the leaf $w$ containing $o_i$. At each non-leaf node $v$ in the search, we update $prod(u)$ to $prod(u) \times prod(v)$ (resp. $val(u)$ to $val(u) \times prod(v)$) for each child $u$ of $v$, and then reset $prod(v)$ to 1.

  At $w$, we set $pt(w)$ to $o_i$, $prod(w)$ to 1, and both $val(w)$ and $val(v)$ to $\mu$. We then walk back up $D$ towards the root and, at each node $v$ visited, we update $val(v)$ to the larger of the $val(\cdot)$ of its children and update $pt(v)$ accordingly.

- **RangeMultR-value($o_i$):** We search downwards in $D$ with $x_2(o_i)$ and identify the set $C$ of canonical nodes, as we did in FindMaxS-value($o_i$). At each non-leaf node $v$ in the search, we update $prod(u)$ to $prod(u) \times prod(v)$ (resp. $val(u)$ to $val(u) \times prod(v)$) for each child $u$ of $v$, and then reset $prod(v)$ to 1. Next, for each node $v'$ in $C$, we update $prod(v')$ to $prod(v') \times q_i$. Finally, starting at the lowest node in $C$ we walk back up $D$ towards the root and, at each node $v$ visited, we update $val(v)$ to the larger of the $val(\cdot)$ of its children and update $pt(v)$ accordingly.

It should be evident from the preceding discussion that $D$ implements the operations correctly in $O(\log n)$ time apiece and uses $O(n)$ space.

### 5 Conclusion

Given a set of points in $\mathbb{R}^d$, where each point has a fixed probability of existence, we have considered the problem of computing the skyline that has the greatest probability of existing, i.e., the most-likely skyline. For $d > 2$, we have shown that the problem is NP-hard and, moreover, cannot even be well-approximated unless $P = NP$. For $d = 2$, we have given an optimal algorithm which runs in $O(n \log n)$ time and uses $O(n)$ space.

### Acknowledgement

The research of the first author was supported, in part, by a Doctoral Dissertation Fellowship from the Graduate School of the University of Minnesota.

### References


