Skyline Computation with Noisy Comparisons

Benoît Groz[1](✉), Frederik Mallmann-Trenn[2], Claire Mathieu[3], and Victor Verdugo[4,5]

[1] Université Paris-Saclay, CNRS, LRI, Gif-sur-Yvette, France
benoit.groz@lri.fr
[2] King's College London, London, UK
frederik.mallmann-trenn@kcl.ac.uk
[3] CNRS & IRIF, Paris, France
claire.m.mathieu@gmail.com
[4] London School of Economics and Political Science, London, UK
v.verdugo@lse.ac.uk
[5] Universidad de O'Higgins, O'Higgins, Chile
victor.verdugo@uoh.cl

Abstract. Given a set of n points in a d-dimensional space, we seek to compute the *skyline*, i.e., those points that are not strictly dominated by any other point, using few comparisons between elements. We adopt the noisy comparison model [15] where comparisons fail with constant probability and confidence can be increased through independent repetitions of a comparison. In this model motivated by Crowdsourcing applications, Groz and Milo [18] show three bounds on the query complexity for the skyline problem. We improve significantly on that state of the art and provide two output-sensitive algorithms computing the skyline with respective query complexity $O(nd\log(dk/\delta))$ and $O(ndk\log(k/\delta))$, where k is the size of the skyline and δ the expected probability that our algorithm fails to return the correct answer. These results are tight for low dimensions.

Keywords: Skyline · Noisy comparisons · Fault-tolerance

1 Introduction

Skylines have been studied extensively, since the 1960s in statistics [6], then in algorithms and computational geometry [22] and in databases [7,12,16,21]. Depending on the field of research, the *skyline* is also known as the set of *maximum vectors*, the *dominance frontier*, *admissible* points, or *Pareto frontier.* The skyline of a set of points consists of those points which are not strictly dominated by any other point. A point p is *dominated* by another point q if $p_i \leq q_i$ for every coordinate (attribute or dimension) i. It is *strictly dominated* if in addition the inequality is strict for at least one coordinate; see Fig. 1.

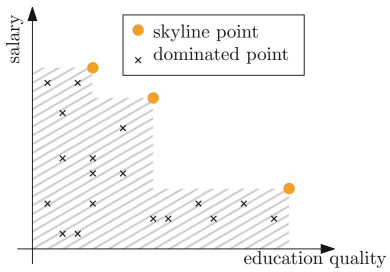


Fig. 1. Given a set of points X, the goal is to find the set of *skyline points*, i.e.,points are not dominated by any other points.

Noisy Comparison Model, and Parameters. In many contexts, comparing attributes is not straightforward. Consider the example of finding *optimal* cities from [18].

> *To compute the skyline with the help of the crowd we can ask people questions of the form "is the education system superior in city x or city y?" or "can I expect a better salary in city x or city y". Of course, people are likely to make mistakes, and so each question is typically posed to multiple people. Our objective is to minimize the number of questions that need to be issued to the crowd, while returning the correct skyline with high probability.*

Thus, much attention has recently been given to computing the skyline when information about the underlying data is uncertain [25], and comparisons may give erroneous answers. In this paper we investigate the complexity of computing skylines in the noisy comparison model, which was considered in [18] as a simplified model for crowd behaviour: we assume queries are of the type *is the i-th coordinate of point p (strictly) smaller than that of point q?*, and the outcome of each such query is independently correct with probability greater than some constant better than 1/2 (for definiteness we assume probability 2/3). As a consequence, our confidence on the relative order between p and q can be increased by repeatedly querying the pair on the same coordinate. Our complexity measure is the number of comparison queries performed.

This noisy comparison model was introduced in the seminal paper [15] and has been studied in [8,18]. There are at least 2 straightforward approaches to reduce noisy comparison problems to the noiseless comparison setting. One approach is to take any "noiseless" algorithm and repeat each of its comparisons $\log(f(n))$ times, where n is the input size and $f(n)$ is the complexity of the algorithm. The other approach is to sort the n items in all d dimensions at a cost of $nd\log(nd)$, then run some noiseless algorithm based on the computed orders. The algorithms in [15,18] and this paper thus strive to avoid the logarithmic overhead of these straightforward approaches.

Three algorithms were proposed in [18] to compute skylines with noisy comparisons. Figure 2 summarizes their complexity and the parameters we consider.

The first algorithm is the reduction through sorting discussed above. But skylines often contain only a small fraction of the input items (points), especially when there are few attributes to compare (low dimension). This leads to more efficient algorithms because smaller skylines are easier to compute. Therefore, [18] and the present paper expresses the complexity of computing skylines as a function of four parameters that appear on Fig. 2: δ, the probability that the algorithm could fail to return the correct output, and three parameters wholly determined by the input set X: the number of input points $n = |X|$, the dimension d of those points, and the size $k = |\text{skyline}(X)|$ of the skyline (output). There is a substantial gap between the lower bounds and the upper bounds achieved by the skyline algorithms in [18]. In particular, the authors raised the question whether the skyline could be computed in $o(nk)$ for any constant d. In this paper, we tighten the gap between the lower and upper bounds and settle this open question.

Contributions. We propose 2 new algorithms that compute skylines with probability at least $1 - \delta$ and establish a lower bound:

- Algorithm **SkyLowDim**(X, δ) computes the skyline in $O(nd\log(dk/\delta))$ query complexity and $O(nd\log(dk\delta) + ndk)$ overall running time.
- Algorithm **SkyHighDim**(X, δ) computes the skyline in $O(ndk\log(k/\delta))$
- $\Omega(nd\log k)$ queries are necessary to compute the skyline when $d \leq k$.
- Additionally, we show that Algorithm **SkyLowDim** can be adapted to compute the skyline with $O(nd\log(dk))$ comparisons in the noiseless setting.

Our first algorithm answers positively the above question from [18]. Together with the lower bound, we thus settle the case of low dimensions, i.e., when there is a constant c such that $d \leq k^c$. Our 2 skyline algorithms both shave off a factor k from the corresponding bounds in the state of the art [18], as illustrated in Fig. 2 with respect to query complexity. **SkyLowDim** is a randomized algorithm that samples the input, which means it may fail to compute the skyline within the bounds even when comparisons are guaranteed correct. However, we show that our algorithm can be adapted to achieve deterministic $O(nd\log(dk))$ for this specific noiseless case.

As a subroutine for our algorithms, we developed a new algorithm to evaluate disjunctions of boolean variables with noise. Algorithm **NoisyFirstTrue** is, we believe, interesting in its own right: it returns the index of the first positive variable in input order, with a running time that scales linearly with the index.
Technical Core of Our Algorithms. The algorithm underlying the two bounds for $k \ll n$ in [18] recovers the skyline points one by one. It iteratively adds to the skyline the maximum point, in lexicographic order, among those not dominated by the skyline points already found.[1] However, the algorithm in [18] essentially considers the whole input for each iteration. Our two algorithms, on the opposite, can identify and discard some dominated points early. The idea behind our

[1] The difference between those two bounds is due to different subroutines to check dominance.

[18]	$O(nd\log(nd/\delta))^\dagger$	$O(ndk\log(dk/\delta))$	$O(ndk^2\log(k/\delta))$	d : dimension
this paper	——	$O(nd\log(dk/\delta))^\dagger$ **SkyLowDim**	$O(ndk\log(k/\delta))$ **SkyHighDim**	n : # input points k : # skyline points δ : error rate tolerated
best when:	$k \in \Omega(\log(dn))$	$d < k^c < n$	$k \ll d$	

Fig. 2. Query complexity of skyline algorithms depending on the values of k. For †-labeled bounds, the running time is larger than the number of queries.

algorithm **SkyHighD − param** is that it is more efficient to separate the two tasks: (i) finding a point p not dominated by the skyline points already found, on the one hand, and (ii) computing a maximum point (in lexicographic order) *among those dominating* p, on the other hand. Whenever a point is considered for step (i) but fails to satisfy that requirement, the point can be discarded definitively. The $O(ndk)$ skyline algorithm from [13] for the noiseless setting also decomposes the two tasks, although the point they choose to add to the skyline in each of the k iteration is not the same as ours.

Our algorithm **SkyLowD − param** can be viewed as a 2-steps algorithm where the first step prunes a huge fraction of dominated points from the input through discretization, and the second step applies a cruder algorithm on the surviving points. We partition the input into buckets for discretization, identify "skyline buckets" and discard all points in dominated buckets. The bucket boundaries are defined by sampling the input points and sorting all sample points in each dimension. In the noisy comparison model, the approach of sampling the input for some kind of discretization was pioneered in [8] for selection problems, but with rather different techniques and objectives. One interesting aspect of our discretization is that a fraction of the input will be, due to the low query complexity, incorrectly discretized yet we are able to recover the correct skyline.

Our lower bound constructs a technical reduction from the problem of identifying null vectors among a collection of vectors, each having at most one non-zero coordinate. That problem can be studied using a two-phase process inspired from [15].

Related Work. The noisy comparison model was considered for sorting and searching objects [15]. While any algorithm for that model can be reduced to the noiseless comparison model at the cost of a logarithmic factor (boosting each comparison so that by union bound all are correct), [15] shows that this additional logarithmic factor can be spared for sorting and for maxima queries, though it cannot be spared for median selection. [17,26] and [8] investigate the trade-off between the total number of queries and the number of rounds for (variants of) top-k queries in the noisy comparison model and some other models. The noisy comparison model has been refined in [14] for top-k queries, where the probability of incorrect answers to a comparison increase with the distance between the two items.

Other models for uncertain data have also been considered in the literature, where the location of points is determined by a probability distribution, or when data is incomplete. Some previous work [3,27] model uncertainty about the output by computing a ρ-skyline: points having probability at least ρ to be in the skyline. We refer to [5] for skyline computation using the crowd and [23] for a survey in crowdsourced data management.

Our paper aims to establish the worst-case number of comparisons required to compute skylines with output-sensitive algorithms, i.e., when the cost is parametrized by the size of the result set. While one of our algorithm is randomized, we do not make any further assumption on the input (we do not assume input points are uniformly distributed, for instance). In the classic *noiseless* comparison model, the problem of computing skylines has received a large amount of attention [7,20,22]. For any constant d, [20] show that skylines can be computed in $O(n\log^{d-2}k)$. In the RAM model, the fastest algorithms we are aware of run in $O(n\log^{d-3}n)$ expected time [10], and $O(n\log\log_{n/k}n(\log_{n/k}n)^{d-3}$ deterministic time [2]. When $d \in \{2,3\}$, the problem even admits "instance-optimal" algorithms [4]. [11] investigates the constant factor for the number of comparisons required to compute skyline, when $d \in \{2,3\}$. The technique does not seem to generalize to arbitrary dimensions, and the authors ask among open problems whether arbitrary skylines can be computed with fewer than $dn\log n$ comparisons. To the best of our knowledge, our $O(nd\log(dk))$ is the first non-trivial output-sensitive upper bound that improves on the folklore $O(dnk)$ for computing skylines in arbitrary dimensions. Many other algorithms have been proposed that fit particular settings (big data environment, particular distributions, etc), as evidenced in the survey [19], but those works are further from ours as they generally do not investigate the asymptotic number of comparisons. Other skyline algorithms in the literature for the noiseless setting have used bucketing. In particular, [1] computes the skyline in a massively parallel setting by partitioning the input based on quantiles along each dimension. This means they define similar buckets to ours, and they already observed that the buckets that contain skyline points are located in hyperplanes around the "bucket skyline", and therefore those buckets only contain a small fraction of the whole input.

Organization. In Sect. 2, we recall standard results about the noisy comparison model and introduce some procedure at the core of our algorithms. Section 3 introduces our algorithm for high dimensions (Theorem 4) and Sect. 4 introduces the counterpart for low dimensions (Theorem 6). Section 5 establishes our lower bound (Theorem 7).

2 Preliminaries

The complexity measured is the number of comparisons in the worst case. Whenever the running time and the number of comparisons differ, we will say so. With respect to the probability of error, our algorithms are supposed to fail with probability at most δ. Following standard practice we only care to prove that our algorithms have error in $O(\delta)$: 5δ, for instance, because the asymptotic complexity of our algorithms would remain the same with an adjusted value for the parameter: $\delta' = \delta/5$.

Given two points, $p = (p_1, p_2 \ldots, p_d)$ and $q = (q_1, q_2 \ldots, q_d)$ point p is *lexicographically* smaller than q, denoted by $p \leq_{\text{lex}} q$, if $p_i < q_i$ for the first i where p_i and q_i differ. If there is no such i, meaning that the points are identical, we use the id of the points in the input as a tie-breaker, ensuring that we obtain a total order. We next describe and name algorithms that we use as subroutines to compute skylines.

Algorithm **NoisySearch** takes as input an element y, an ordered list $(y_1, y_2, \ldots, y_m)$, accessible by comparisons that each have error probability at most p, and a parameter δ. The goal is to output the interval $I = (y_{i-1}, y_i]$ such that $y \in I$.

Algorithm **NoisySort** relies on **NoisySearch** to solve the **noisy sort problem**. It takes as input an unordered set $Y = \{y_1, y_2, \ldots, y_m\}$, and a parameter δ. The goal is to output an ordering of Y that is the correct non-decreasing sorted order. In the definition above, the order is kept implicit. In our algorithms, the input items are d-dimensional points, so **NoisySort** will take an additional argument i indicating on which coordinate we are sorting those points.

Algorithm **NoisyMax** returns the maximum item in the unordered set Y whose elements can be compared, but we will rather use another variant: algorithm **MaxLex** takes as input an unordered set $Y = \{y_1, y_2, \ldots, y_m\}$, a point x and a parameter δ. The goal is to output the maximum point in lexicographic order among those that dominate x. Algorithm **SetDominates** is the boolean version whose goal is to output whether there exists a point in Y that dominates x.

Algorithm **NoisyFirstTrue** takes as input a list $(y_1, y_2, \ldots, y_m)$ of boolean elements that can be compared to `true` with error probability at most p (typically the result of some comparison or subroutines such as **SetDominates**). The goal is to output the index of the first element with value `true` (and $m+1$, which we assimilate to `false`, if there are none).[2]

Theorem 1 ([15,18]). *When the input comparisons have error probability at most $p = 1/3$, the table below lists the number of comparisons performed by the algorithms to return the correct answer with success probability $1 - \delta$:*

Algorithm	**NoisyMax**	**NoisySort**	**NoisySearch**	**SetDominates**	**MaxLex**
Comparisons	$O(m\log\frac{1}{\delta})$	$O(m\log\frac{m}{\delta})$	$O(\log\frac{m}{\delta})$	$O(md\log\frac{1}{\delta})$	$O(md\log\frac{1}{\delta})$

We denote by CheckVar(x, δ) the procedure that checks if $x =$ `true` with error probability δ by majority vote, and returns the corresponding boolean.

Theorem 2. *Algorithm* **NoisyFirstTrue** *solves the first positive variable problem with success probability $1-\delta$ in $O(j \cdot \log(1/\delta))$ where j is the index returned.*

[2] As in [18] (but with stronger bounds), this improves upon an $O(m\log(1/\delta))$ algorithm from [15] that only answers whether at least one of the elements is true.

Proof. The proof, left for the long version [24], shows that the error (resp. the cost) of the whole algorithm is dominated by the error (resp. the cost) of the last iteration.

Algorithm NoisyFirstTrue$(x_1, \ldots, x_n, \delta)$ (see Theorem 2)
input: $\{x_1, \ldots, x_n\}$ set of boolean random variables, δ error probability
output: the index j of the first positive variable, or $n+1$ (=`false`).

```
1: $i \leftarrow 1$
2: $\delta' \leftarrow \delta/2$
3: while $i \leq n$ do
4:    $j \leftarrow$ NoisyOr$(x_1, \ldots, x_i, \delta')$
5:    if CheckVar$(x_j, \delta'/2^i)$ then
6:       return $j$
7:    else
8:       $i \leftarrow 2 \cdot i$
9: return false
```

3 Skyline Computation in High Dimension

We first introduce Algorithm **SkyHighD** – **param** which assumes that an estimate $\hat{k}$ of k is known in advance. We will show afterwards how we can lift that assumption.

Theorem 3. *Given* $\delta \in (0, 1/2)$ *and a set* X *of data items,* **SkyHighD**–**param**(X, δ) *outputs* $\min(|X|, \hat{k})$ *skyline points, with probability at least* $1 - \delta$*. The running time and number of queries is* $O(nd\hat{k}\log(\hat{k}/\delta))$.

Proof. Each iteration through the loop adds a point to the skyline S with probability of error at most $\delta/\hat{k}$. The final result is therefore correct with success probability $1 - \delta$. The complexity is $O((i' - i) * d\hat{k}\log(\hat{k}/\delta))$ to find a non-dominated point $p_{i'}$ at line 3, and $O(nd\log(\hat{k}/\delta))$ to compute the maximal point above $p_{i'}$ at line 4. Summing over all iterations, the running time and number of queries is $O(nd\hat{k}\log(\hat{k}/\delta))$.

Algorithm **SkyHighD** – **param**(X, δ) needs a good estimate of the skyline cardinality $\hat{k} \in O(k)$ to return the skyline in $O(ndk\log(k/\delta))$. To guarantee that complexity, algorithm **SkyHighDim** exploits the classical trick from Chan [9] of trying a sequence of successive values for $\hat{k}$ – a trick that we also exploit in algorithms **NoisyFirstTrue** and **SkyLowDim**. The sequence grows exponentially to prevent failed attempts from penalizing the complexity.

Theorem 4. *Given* $\delta \in (0, 1/2)$ *and a set* X *of data items,* **SkyHighDim**(X, δ) *outputs a subset of* X *which, with probability at least* $1 - \delta$*, is the skyline. The running time and number of queries is* $O(ndk\log(k/\delta))$.

Proof. The proof is relatively straightforward and left for the long version.

Algorithm SkyHighD − param(k, X, δ) (see Theorem 3)
input: $X = \{p_1, \ldots, p_n\}$ set of points, $\hat{k}$ upper bound on skyline size, δ error probability
output: $\min\{\hat{k}, \text{skyline}(X)\}$ skyline points w.p. $1 - \delta$

```
1: Initialize $S \leftarrow \emptyset$, $i \leftarrow 1$
2: while $i \neq -1$ and $|S| < \hat{k}$ do
3:     $i' \leftarrow$ index of the first point $p_{i'}$ not dominated by current skyline points.[a]
       {Find a skyline point dominating $p_i$}
4:     Compute $p^* \leftarrow$ MaxLex$(p_{i'}, \{p_i, \ldots, p_n\}, \delta/(2\hat{k}))$
5:     $S \leftarrow S \cup \{p^*\}$
6:     $i \leftarrow i'$
7: Output $S$
```

[a]This point can be computed using algorithm **NoisyFirstTrue** on the boolean variables: $\neg$**SetDominates**$(S, p_i, \delta/(2\hat{k}))$,..., $\neg$**SetDominates**$(S, p_n, \delta/(2\hat{k}))$, where we denote by $\neg$ the negation. This means that $\neg$**SetDominates** $(S, p_n, \delta/(2\hat{k}))$ returns true when the procedure **SetDominates**$(S, p_n, \delta/(2\hat{k}))$ indicates that p_n is not dominated.

Algorithm SkyHighDim(X, δ) (see Theorem 4)
input: X set of points, δ error probability
output: skyline(X) w.p. $1 - \delta$

```
1: Initialize $j \leftarrow 0$, $\hat{k} \leftarrow 1$
2: repeat
3:     $j \leftarrow j + 1$ ; $\hat{k} \leftarrow 2\hat{k}$ ; $S \leftarrow$ SkyHighD − param$(\hat{k}, X, \delta/2^j)$
4: until $|S| < \hat{k}$
5: Output $S$
```

4 Skyline Computation in Low Dimension

Let us first sketch our algorithm **SkyLowD − param**(k, X, δ). The algorithm works in 3 phases. The first phase partitions input points in buckets. We sort the i-th coordinate of a random sample to define $s + 1$ intervals in each dimension $i \in [d]$, hence $(s + 1)^d$ *buckets*, where each bucket is a product of intervals of the form $\prod_i I_i$; then we assign each point p of X to a bucket by searching in each dimension for the interval I_i containing p_i. Of course we do not materialize buckets that are not assigned any points.

The second phase eliminates irrelevant buckets: those that are dominated by some non-empty bucket and therefore have no chance of containing a skyline point. In short, the idea is to identify the "skyline of the buckets", and use it to discard the dominated buckets, as defined in Sect. 4.2. With high probability the bucketization obtained from the first phase will be "accurate enough" for our purpose: it will allow to identify efficiently the irrelevant buckets, and will also guarantee that the points in the remaining buckets form a small fraction of the input (provided k and d are small).

In phase 3, we thus solve the skyline problem on a much smaller dataset, calling Algorithm $\mathbf{SkyHighD-param}$ to find the skyline of the remaining points.[3] The whole purpose of the bucketization is to discard most input points while preserving the actual skyline points, so that we can then run a more expensive algorithm on the reduced dataset.

Figure 3 illustrates our algorithm: the grey buckets (c, f, g) are dominated by some (non-empty) orange buckets (a, b, d, e) so they cannot contain skyline points: in phase 3, the algorithm solves the reduced problem on the points contained in the orange buckets.

4.1 Identifying "Truly Non-empty" Buckets

Our bucketization does not guarantee that all points are assigned to the proper bucket, because it would be too costly with noisy comparisons. In particular, empty buckets may erroneously be assumed to contain some points (e.g., the buckets above a, b on Fig. 3). Those empty buckets also are irrelevant, even if they are not dominated by the "skyline" buckets. To drop the irrelevant buckets, we thus design a subroutine **First-Nonempty-Bucket** that processes a list of buckets, and returns the first bucket that really contains at least one point. Incidentally, we will not double-check the emptiness of every bucket using this procedure, but will only check those that may possibly belong to the skyline: those that we will define more formally as buckets of type (i), (ii) and (iv) in the proof of Theorem 5. We could not afford to "fix" the whole assignment as it may contain too many buckets.

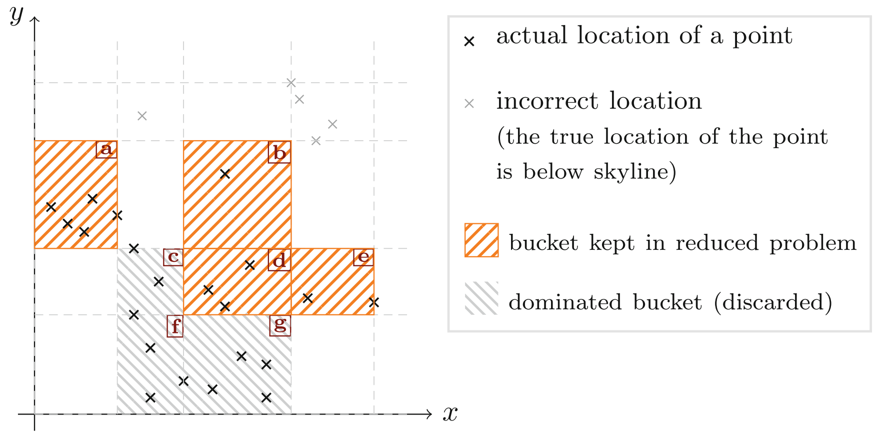
In the **First-Nonempty-Bucket** problem, the input is a sequence of pairs $[(B_1, X_1), \ldots, (B_n, X_n)]$ where B_i is a bucket and X_i is a set of points. The goal is to return the first i such that $B_i \cap X_i \neq \emptyset$ with success probability $1-\delta$. The test $B_i \cap X_i \neq \emptyset$ can be formulated as a DNF with $|X_i|$ conjunctions of $O(d)$ boolean variables each. To solve **First-Nonempty-Bucket**, we can flatten the formulas of all buckets into a large DNF with conjunctions of $O(d)$ boolean variables (one conjunction per bucket point). We call $\mathbf{FirstBucket}([(B_1, X_1), \ldots, (B_n, X_n)], \delta)$ the algorithm that executes **NoisyFirstTrue** to compute the first true conjunction, while keeping tracks of which point belongs to which bucket with pointers:

Lemma 1. *Algorithm* $\mathbf{FirstBucket}([(B_1, X_1), \ldots, (B_n, X_n)], \delta)$ *solves problem* **First-Nonempty-Bucket** *in* $O(\sum_{i \leq j} d \cdot |X_i| \log(1/\delta))$ *with success probability* $1-\delta$*, where* j *is the index returned by the algorithm.*

4.2 Domination Relationships Between Buckets

In the second phase, Algorithm $\mathbf{SkyLowD-param}(k, X, \delta)$ eliminates irrelevant buckets. To manage ties, we need to distinguish two kinds of intervals: the

[3] Alternatively, one could use an algorithm provided by Groz and Milo [18], it is only important that the size of the input set is reduced to n/k to cope with the larger runtime of the mentioned algorithms.



y
x
a
b
c
d
e
f
g
× actual location of a point
× incorrect location (the true location of the point is below skyline)
▨ bucket kept in reduced problem
▨ dominated bucket (discarded)

Fig. 3. An illustration of the bucket dominance and its role in **SkyLowD − param**. Here bucket b dominates c and f but not a, d, e or g. Buckets c, f, g are dominated by some non-empty bucket and therefore cannot contain a skyline point. Bucket a does not contain a skyline point, but this cannot be deduced from the bucket assignments, therefore points in bucket a are passed on to the reduced problem. In this figure we may assume to simplify that a bucket contains its upper boundary. But in our algorithm bucket a would actually contain only the 4 leftmost points, and the fifth point would belong to a distinct bucket with a trivial interval on x. (Color figure online)

trivial intervals that match a sample coordinate: $I = [x, x]$, and the non-trivial intervals $I =]a, b[$ $(a < b)$ contained between samples (or above the largest sample, or below the smallest sample). To compare easily those intervals, we adopt the convention that for a non-trivial interval $I =]a, b[$, $\min\ I = a + \epsilon$ and $\max\ I = b - \epsilon$ for some infinitesimal $\epsilon > 0$: $\epsilon = (b - a)/3$ would do. We say that a bucket $B = \prod_i I_i$ is *dominated* by a different bucket $B' = \prod_i I'_i$ if in every dimension $\max\ I_i \leq \min\ I'_i$. Equivalently: we say that B' dominates B if every point (whether in the dataset or hypothetical) in B' dominates every point in B. The idea is that no skyline point belongs to a bucket dominated by a non-empty bucket. We observe that the relative position of buckets is known by construction, so deciding whether a bucket dominates another one may require time $O(d)$ but does not require any comparison query.

Figure 3 illustrates the relevant and discarded buckets. On that figure, we depicted a few empty buckets above the skyline that are erroneously assumed to contain some points as a result of noise during the assignment. Of course, there are also incorrect assignments of points into empty or non-empty buckets below the skyline, as well as incorrect assignments into the "skyline buckets". These incorrect assignments are not an issue as long as there are not too many of them: dominated buckets will be discarded as such, whether empty or not, and the few irrelevant points maintained into the reduced dataset will be discarded in phase 3, when the skyline of this dataset is computed.

4.3 Algorithm and Bounds for Skyline Computation in Low Dimension

Theorem 5. *Given* $\delta \in (0, 1/2)$*,* $\hat{k} > 0$*, and a set* X *of data items, algorithm* $\mathbf{SkyLowD-param}(\hat{k}, X, \delta)$ *outputs* $\min(|X|, \hat{k})$ *skyline points, with probability at least* $1 - \delta$*. The number of queries is* $O(nd\log(d\hat{k}/\delta))$*. The running time is* $O(nd\log(d\hat{k}/\delta) + nd \cdot \min(\hat{k}, |skyline(X)|))$

Proof. The proof, left for the long version, first shows by Chernoff bounds that the assignment satisfies with high probability some key properties: (1) few points are erroneously assigned to incorrect buckets (2) the skyline points are assigned to the correct bucket, and (3) there are at most $O(n/(d\hat{k}^2))$ points on any hyperplane (i.e., in buckets that are ties on some dimension). The proof then shows that:

- there are at most $O(n/\hat{k})$ points in the reduced problem. This is because those points belong to skyline buckets or buckets that are tied with a skyline bucket on at least one dimension (every other non-empty bucket is dominated), and property (3) of the assignment guarantees that the union of all such buckets has at most $O(n/\hat{k})$ points.
- the buckets above the skyline buckets which are erroneously assumed to contain points can quickly be identified and eliminated since they contain few points.

Algorithm $\mathbf{SkyLowD-param}(\hat{k}, X, \delta)$ needs a good estimate of the skyline cardinality to return the skyline in $O(nd\log(dk/\delta))$: we must have $\hat{k} \geq k$ and $\log(\hat{k}) \in O(\log(k))$. Algorithm $\mathbf{SkyLowDim}(X, \delta)$ (left for the long version) guarantees the complexity by trying a sequence of successive values for $\hat{k}$. The successive values in the sequence grow super exponentially (similarly to [9,18]) to prevent failed attempts from penalizing the complexity.

Theorem 6. *Given* $\delta \in (0, 1/2)$ *and a set* X *of data items,* $\mathbf{SkyLowDim}(\hat{k}, X, \delta)$ *outputs a subset of* X *which, with probability at least* $1 - \delta$*, is the skyline. The number of queries is* $O(nd\log(dk/\delta))$*. The running time is* $O(nd\log(dk/\delta) + ndk)$*.*

Proof. For iteration j, the algorithm bounds the probability of error by $\delta/2^j$, and the corresponding cost is given by Theorem 5, hence the complexity we claim by summing those terms over all iterations.

Remark 1. In the noiseless setting, we could adopt the same sampling approach to assign points to buckets and reduce the input size. On line 18 we could use any noiseless skyline algorithm such as the $O(ndk)$ algorithm from [13], or our own similar **SkyHighDim** which can clearly run in $O(dnk)$ in the noiseless case. The cost of the bucketing phase remains $O(nd\log(d\hat{k}/\delta))$. The elimination phase becomes rather trivial since all points get assigned to their proper bucket, and therefore there is no need to check buckets for emptiness as in Line 13. By

Algorithm SkyLowD – param($\hat{k}, X, \delta$) (see Theorem 5)
input: $\hat{k}$ integer, X set of points, δ error probability
output: $\min\{\hat{k}, |\text{skyline}(X)|\}$ points of skyline(X)
error probability: δ

```
 1: if $\hat{k}^5 \geq n$ or $d^5 \geq n$ or $(\log(1/\delta))^5 \geq n$ then
 2:     Compute the skyline by sorting every dimension, as in [18]. Return that skyline.
 3: $\delta' \leftarrow \delta/(2d\hat{k})^5$ and $s \leftarrow d\hat{k}^2 \log(d^2\hat{k}^2/\delta')$
    {Phase (i): bucketing}
 4: for each dimension $i \in \{1, 2, \ldots, d\}$ do
 5:     $S_i \leftarrow$ NoisySort(sample of $X$ of size $s, i, \delta'/d$)
 6:     Remove duplicates so that, with prob. $1 - \delta'/d$, the values in $S_i$ are all distinct.[a]
 7: for each point $p \in X$ do
 8:     Place $p$ in set $X_B$ associated to $B = \prod_{i=1}^{d} I_i$, with $I_i$ = NoisySearch($p_i, S_i$,
        $\delta'/(d\hat{k})$).
 9: Drop all empty buckets (those that were assigned no point).
10: Sort buckets into a sequence $B_1, \ldots, B_h$ so that each bucket comes before buckets
    it dominates.
    {Phase (ii): eliminating irrelevant buckets}
11: Initialize $X' \leftarrow \emptyset$, $i \leftarrow 1$
12: while $i \neq -1$ do
13:     $i \leftarrow$ FirstBucket($[(B_1, X_{B_1}), \ldots, (B_h, X_{B_h})], \delta'/\hat{k}$))
14:     $X' \leftarrow X' \cup X_{B_i}$
15:     if $|X'| > 8n/\hat{k}$ then
16:         Raise an error.
17:     Drop from $B_1, \ldots, B_h$ all buckets dominated by $B_i$, and also buckets $B_1$ to $B_i$.
    {Phase (iii): solve reduced problem}
18: Output SkyHighDim($X', \delta'$).
```

[a]Note that X can contain points sharing the same coordinate meaning that the S_i are not necessarily distinct.

setting $\delta = 1/k$ failures are scarce enough so that the higher cost of $O(ndk)$ in case of failure is covered by the cost of an execution corresponding to a satisfying sample. Consequently, the expected query complexity is $O(nd\log(dk))$, and the running time $O(nd\log(dk) + ndk)$.

Better yet: we can replace random sampling with quantile selection to obtain a deterministic algorithm with the same bounds. Algorithms for the *multiple selection* problem are surveyed in [11]. Actually, our algorithm can be viewed as some kind of generalization to higher dimensions of an algorithm from [11] which assigns points to buckets before recursing, the buckets being the quantiles along one coordinate.

5 Skyline Lower Bound

To achieve meaningful lower bounds (that do not reduce to the noiseless setting), we assume here that the input comparisons have a probability of error at least 1/3. Of course, we just need the probability to be bounded away from 0.

Theorem 7. *For any $n \geq k \geq d > 0$, any algorithm that recovers with error probability at most $1/10$ the skyline for any input having exactly k skyline points, requires $\Omega(nd\log(k/\delta))$ queries in expectation on a worst-case input.*

Proof. The proof is left for the long version.

6 Conclusion and Related Work

We introduced 2 algorithms to compute skylines with noisy comparisons. The most involved shows that we can compute skylines in $O(nd\log dk/\delta)$ comparisons. We also show that this bound is optimal when the dimensions is low ($d \leq k^c$ for some constant c), since computing noisy skylines requires $\Omega(dn\log k)$ comparisons. All our algorithms but **SkyLowDim** in $O(nd\log dk/\delta)$ are what we call *trust-preserving*([18]), meaning that when the probability of errors in input comparisons is already at most $\delta < 1/3$, we can discard from the complexity the dependency in δ (replacing δ by some constant).

We leave open the question of the optimal number of comparisons required to compute skylines for arbitrarily large dimensions. Even in the noiseless case, it is not lear whether the skyline could be computed in $O(dn\log k)$ comparisons. Our algorithm is output sensitive (the running time is optimized with respect to the output size) but we did not investigate its instance optimality. However, knowing the input set up to a permutation of the points does not seem to help identifying the skyline points in the noisy comparison model, so we believe that for every k and on any input of skyline cardinality k, even with this knowledge any skyline algorithm would still require $\Omega(dn\log k)$ comparisons. We leave open the question of establishing such a stronger lower bound.

References

1. Afrati, F.N., Koutris, P., Suciu, D., Ullman, J.D.: Parallel skyline queries. In: 15th International Conference on Database Theory, ICDT 2012, Berlin, Germany, 26–29 March 2012, pp. 274–284 (2012). https://doi.org/10.1145/2274576.2274605
2. Afshani, P.: Fast computation of output-sensitive maxima in a word RAM. In: Chekuri, C. (ed.) Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, 5–7 January 2014, pp. 1414–1423. SIAM (2014). https://doi.org/10.1137/1.9781611973402.104
3. Afshani, P., Agarwal, P.K., Arge, L., Larsen, K.G., Phillips, J.M.: (approximate) uncertain skylines. In: Proceedings of the 14th International Conference on Database Theory, ICDT 2011, pp. 186–196. ACM (2011). https://doi.org/10.1145/1938551.1938576
4. Afshani, P., Barbay, J., Chan, T.M.: Instance-optimal geometric algorithms. J. ACM **64**(1), 3:1–3:38 (2017)
5. Asudeh, A., Zhang, G., Hassan, N., Li, C., Zaruba, G.V.: Crowdsourcing pareto-optimal object finding by pairwise comparisons. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pp. 753–762. ACM (2015)

6. Barndorff-Nielsen, O., Sobel, M.: On the distribution of the number of admissible points in a vector random sample. Theory Prob. Appl. **11**(2), 249–269 (1966). http://search.proquest.com/docview/915869827?accountid=15867
7. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th International Conference on Data Engineering, pp. 421–430. IEEE Computer Society (2001). http://dl.acm.org/citation.cfm?id=645484.656550
8. Braverman, M., Mao, J., Weinberg, S.M.: Parallel algorithms for select and partition with noisy comparisons. In: Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing, STOC 2016, pp. 851–862 (2016)
9. Chan, T.M.: Optimal output-sensitive convex hull algorithms in two and three dimensions. Discrete Comput. Geom. **16**(4), 361–368 (1996). https://doi.org/10.1007/BF02712873
10. Chan, T.M., Larsen, K.G., Patrascu, M.: Orthogonal range searching on the ram, revisited. In: Hurtado, F., van Kreveld, M.J. (eds.) Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, 13–15 June 2011, pp. 1–10. ACM (2011). https://doi.org/10.1145/1998196.1998198
11. Chan, T.M., Lee, P.: On constant factors in comparison-based geometric algorithms and data structures. Discrete Comput. Geom. **53**(3), 489–513 (2015). https://doi.org/10.1007/s00454-015-9677-y
12. Chomicki, J., Ciaccia, P., Meneghetti, N.: Skyline queries, front and back. SIGMOD Rec. **42**(3), 6–18 (2013). https://doi.org/10.1145/2536669.2536671
13. Clarkson, K.L.: More output-sensitive geometric algorithms (extended abstract). In: 35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20–22 November 1994, pp. 695–702 (1994). https://doi.org/10.1109/SFCS.1994.365723
14. Davidson, S.B., Khanna, S., Milo, T., Roy, S.: Top-k and clustering with noisy comparisons. ACM Trans. Database Syst. **39**(4), 35:1–35:39 (2014). https://doi.org/10.1145/2684066
15. Feige, U., Raghavan, P., Peleg, D., Upfal, E.: Computing with noisy information. SIAM J. Comput. **23**(5), 1001–1018 (1994). https://doi.org/10.1137/S0097539791195877
16. Godfrey, P., Shipley, R., Gryz, J.: Algorithms and analyses for maximal vector computation. VLDB J. **16**(1), 5–28 (2007). https://doi.org/10.1007/s00778-006-0029-7
17. Goyal, N., Saks, M.: Rounds vs. queries tradeoff in noisy computation. Theory of Computing **6**(1), 113–134 (2010)
18. Groz, B., Milo, T.: Skyline queries with noisy comparisons. In: Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 15, pp. 185–198. ACM (2015). https://doi.org/10.1145/2745754.2745775
19. Kalyvas, C., Tzouramanis, T.: A survey of skyline query processing (2017). CoRR abs/1704.01788, http://arxiv.org/abs/1704.01788
20. Kirkpatrick, D.G., Seidel, R.: Output-size sensitive algorithms for finding maximal vectors. In: Proceedings of the First Annual Symposium on Computational Geometry, SCG 1985, pp. 89–96. ACM (1985). https://doi.org/10.1145/323233.323246
21. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: an online algorithm for skyline queries. In: Proceedings of the 28th International Conference on Very Large Data Bases, VLDB 2002, VLDB Endowment, pp. 275–286 (2002). http://dl.acm.org/citation.cfm?id=1287369.1287394

22. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. J. ACM **22**(4), 469–476 (1975). https://doi.org/10.1145/321906.321910
23. Li, G., Wang, J., Zheng, Y., Franklin, M.J.: Crowdsourced data management: a survey. In: 33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, 19–22 April 2017, pp. 39–40. IEEE Computer Society (2017). https://doi.org/10.1109/ICDE.2017.26
24. Mallmann-Trenn, F., Mathieu, C., Verdugo, V.: Skyline computation with noisy comparisons (2017). CoRR abs/1710.02058, http://arxiv.org/abs/1710.02058
25. Marcus, A., Wu, E., Karger, D., Madden, S., Miller, R.: Human-powered sorts and joins. Proc. VLDB Endow. **5**(1), 13–24 (2011). https://doi.org/10.14778/2047485.2047487
26. Newman, I.: Computing in fault tolerant broadcast networks and noisy decision trees. Rand. Struct. Algorithms **34**(4), 478–501 (2009)
27. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic skylines on uncertain data. In: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB 2007, VLDB Endowment, pp. 15–26 (2007). http://dl.acm.org/citation.cfm?id=1325851.1325858