Skyline Computation with Noisy Comparisons*

Benoît Groz[1], Frederik Mallmann-Trenn[2,3], Claire Mathieu[2,3], and Victor Verdugo[3]

[1] Université Paris-Saclay, CNRS, LRI
benoit.groz@lri.fr
[2] King's College London
frederik.mallmann-trenn@kcl.ac.uk
[3] CNRS & IRIF
claire.m.mathieu@gmail.com
[4] London School of Economics
v.verdugo@lse.ac.uk

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 ([13]) 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 & Milo [16] 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 $O(nd\log(dk))$ and $O(ndk\log(k))$, where k is the size of the skyline. These results are tight for low dimensions.

Keywords: Skyline · Noisy comparisons · Fault-tolerance· CrowdSourcing.

1 Introduction

Skylines have been studied extensively, since the 1960s in statistics [5], then in algorithms and computational geometry [21] and in databases [6, 10, 14, 20]. 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 Figure 1.

Noisy comparison model, and parameters. In many contexts, comparing attributes is not straightforward. Consider the example of finding *optimal* cities from [16].

> *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*

* This paper is NOT eligible for the best student paper award.

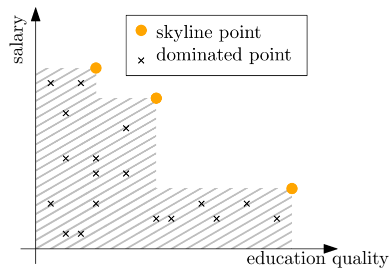

Fig. 1: Given a set of points X, the goal is to find the set of *skyline points*, i.e.,points are not dominated by any other points.

> *"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 [24], 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 [16] 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 [13] and has been studied in [7, 16]. There are at least 2 straightforward approaches to reduce problems in this model 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 [13, 16] and this paper thus strive to avoid the logarithmic overhead of these straightforward approaches.

Three algorithms were proposed in [16] 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, [16] and the present paper investigate the complexity of computing skylines expressed as a function of three parameters: $n=|X|$, the number of input points; d, the number of dimensions; and $k=|\text{skyline}(X)|$, the size of the skyline (output). There is a substantial gap between the lower bounds and the upper bounds achieved by the skyline algorithms in [16]. In particular, the authors raised the question whether the skyline could be computed in $o(nk)$ for any constant k when $k \ll n$. 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-Search**(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-Search**(X,δ) computes the skyline in $O(ndk\log(k/\delta))$
- $\Omega(nd\log k)$ queries are necessary to compute the skyline.
- Additionally, we show that Algorithm **SkyLowDim-Search** 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 [16]. Together with the lower bound, we thus settle the case of low dimensions, i.e., when there is a constant c such that $d \le k^c$. Our 2 skyline algorithms both shave off a factor k from the corresponding bounds in the state of the art [16], as illustrated in Figure 2 with respect to query complexity. We point out that **SkyLowDim-Search** is a randomized algorithm: it needs to sample the input. In our bounds we guarantee that the combined probability of incorrect comparisons and poor sample choice is low: this is because we tailor the sample size to the desired accuracy. But having a randomized algorithm is still a weakness in the sense that our approach cannot yield a "trust-preserving" algorithm: even in the extreme case where comparison queries all return a correct answer (noiseless setting), our algorithm still relies on sampling and therefore has some probability of failing to return the skyline within the running time bound. However, we show that for the specific case of the noiseless setting, our algorithm can be adapted to compute the skyline in $O(nd\log(dk))$.

As a subroutine for our algorithms, we developped a new algorithm to evaluate disjunctions of boolean variables with noise ("OR"). We believe algorithm **NoisyFirstTrue** to be interesting in its own right: it returns the first positive variable in input order, with a running time that scales linearly with the position of that variable in the input order.

[16]	$O(nd\log(nd/\delta))^\dagger$	$O(ndk\log(dk/\delta))$	$O(ndk^2\log(k/\delta))$
this paper	—	$O(nd\log(dk/\delta))^\dagger$	$O(ndk\log(k/\delta))$
best when:	$k \in \Omega(n)$	$d \le k^c \le n$	$k \ll d$

d : dimension
n : # input points
k : # skyline points
δ : error rate tolerated

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.

Technical core of our algorithms. The algorithm underlying the two bounds for $k \ll n$ in [16] 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. [5] However, the algorithm in [16] essentially considers the whole input for each iteration. Our two algorithms, on the opposite, can identify and thus discard some dominated points early. The idea behind our algorithm **SkylineHighDim** 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 [11] for the noiseless setting also decomposes the two tasks, although the point they add to the skyline in each of the k iteration is not the lexicographic maximum.

Our algorithm **SkylineLowDim** 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 eliminate 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 [7] 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 [13].

Related work. The noisy comparison model was considered for sorting and searching objects [13]. 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 the comparisons required are correct), [13] shows that this additional logarithmic factor can be spared for sorting and for maxima queries, though it cannot be spared for median selection. [25], [15] and [7] 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 [12] 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: in some, the location of each point is determined by a probability distribution over a set of locations, whereas in other models the data is incomplete [18, 23]. Some previous work [2, 26] model uncertainty about the output by computing a ρ-skyline: points having probability at least ρ to be in the skyline. We refer to [4] for skyline computation using the crowd and [22] 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

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

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 [6, 19, 21]. For any constant d, [19] show that skylines can be computed in $O(n\log^{d-2} k)$. When $d \in \{2,3\}$, Barbay et al. [3] provide stronger efficiency guarantees with "instance-optimal" algorithms. [9] 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 [17], 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 Section 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 Section 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. This is because we can run the algorithm with an adjusted value for the parameter ($\delta' = \delta/5$) while maintaining the asymptotic complexity of our algorithms.

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.

In the noisy comparison model, we call an algorithm *trust-preserving*([16][6]) if for every $\delta < 1/3$ it is guaranteed to return the correct answer with probability at least

[6] [25] calls such algorithms *fault-tolerant*

$1-\delta$ whenever the input comparisons are correct with probability at least $1-\delta$. 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 $(y_1,y_2,...,y_m)$, 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 $Y=\{y_1,y_2,...,y_m\}$, and a parameter δ. The goal is to output an ordering of Y that is the correct non-decreasing sorted order.

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 $Y=\{y_1,y_2,...,y_m\}$, 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 **NoisyOr** takes as input a list $(y_1,y_2,...,y_m)\}$ 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 original goal was to output whether at least one of the elements is true. But we rather adopt the enhanced version discussed in [16] which solves the *first positive variable problem*. 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).

Theorem 1 ([13], [16]). *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	**NoisyOr**	**NoisyMax**	**NoisySort**	**NoisySearch**	**SetDominates**	**MaxLex**
Comparisons	$O(m\log\frac{1}{\delta})$	$O(m\log\frac{1}{\delta})$	$O(m\log\frac{m}{\delta})$	$O(\log\frac{m}{\delta})$	$O(md\log\frac{1}{\delta})$	$O(md\log\frac{1}{\delta})$

Furthermore, these algorithms are trust preserving. This means that when the input comparisons already have error probability at most δ, we can discard from the complexity the dependency in δ (replacing δ by some constant).

We first refine the complexity of **NoisyOr** and call **NoisyFirstTrue** the refined algorithm which only spends constant time per variable it processes, and which identifies correctly all processed variables with high probability.

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 output by the algorithm. Furthermore, the algorithm is trust-preserving.*

Proof. The proof, left for the Appendix, 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$(x_1,...,x_n,\delta)$ (see Theorem 2)
input: $\{x_1,...,x_n\}$ set of boolean random variables, δ error probability
output: the index j of the first positive variable, or $m+1$ (=`false`).

$i \leftarrow 1$
$\delta' \leftarrow \delta/2$
while $i \leq n$ **do**
  $j \leftarrow$ **NoisyOr**$(x_1,...,x_i,\delta')$
  if CheckVar$(x_j,\delta'/2^i)$ **then**
    return j
  else
    $i \leftarrow 2 \cdot i$
return `false`

3 Skyline computation in high dimension

We first assume that an estimate $\hat{k}$ of k is known in advance. We will show afterwards how we can lift that assumption.

We are now ready to give the full description of our algorithm **SkylineHighDim**.

Algorithm SkylineHighDim(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. $1-\delta$

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

Theorem 3. *Given* $\delta \in (0,1/2)$ *and a set* X *of data items,* **SkylineHighDim**(X,δ) *outputs a subset of* X *which, with probability at least* $1-\delta$*, is the first* $\min(|X|,\hat{k})$ *skyline points. 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 $1-\delta$.

[7] 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.

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 **SkylineHighDim**(X,δ) can only return the skyline in $O(ndk\log(k/\delta))$ if it is provided with a good estimate of the skyline cardinality $\hat{k}\in O(k)$. We next show how to guarantee the complexity by trying a sequence of successive values for $\hat{k}$. The successive values in the sequence grow exponentially to prevent failed attempts from penalizing the complexity.

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

```
Initialize j ← 0, k̂ ← 1
repeat
    j ← j+1 ; k̂ ← 2k̂ ; S ← SkylineHighDim(k̂,X,δ/2^j)
until |S| < k̂
Output S
```

Theorem 4. *Given $\delta\in(0,1/2)$ and a set X of data items,* **SkyHighDim-Search**(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 Appendix.

4 Skyline computation in low dimension

Let us first sketch our algorithm **SkylineLowDim**(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.

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. With high probability the bucketization obtained from the first phase will be "accurate enough" so that we will be able 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 is small). In phase 3, we thus solve the skyline problem on a much smaller dataset, calling Algorithm **SkylineHighDim** to find the skyline of the remaining points. [8]

[8] Alternatively, one could use an algorithm provided by Groz and Milo [16], 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.

4.1 Emptiness testing, and domination relationships between buckets

Our bucketization does not guarantee that all points are assigned to the proper bucket. In particular, empty buckets may erroneously be assumed to contain some points. 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 $[(B_1,X_1),...,(B_n,X_n)]$ 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). Using **NoisyFirstTrue** to compute the first true conjunction (while keeping tracks of which point belongs to which bucket with pointers) yields the following complexity:

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

In the second phase, Algorithm **SkylineLowDim**(k,X,δ) uses elimination. To manage ties, we need to distinguish two kinds of intervals: the 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. See Figure 3 for an illustration. 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.

4.2 Properties satisfied by the bucket assignments

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

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

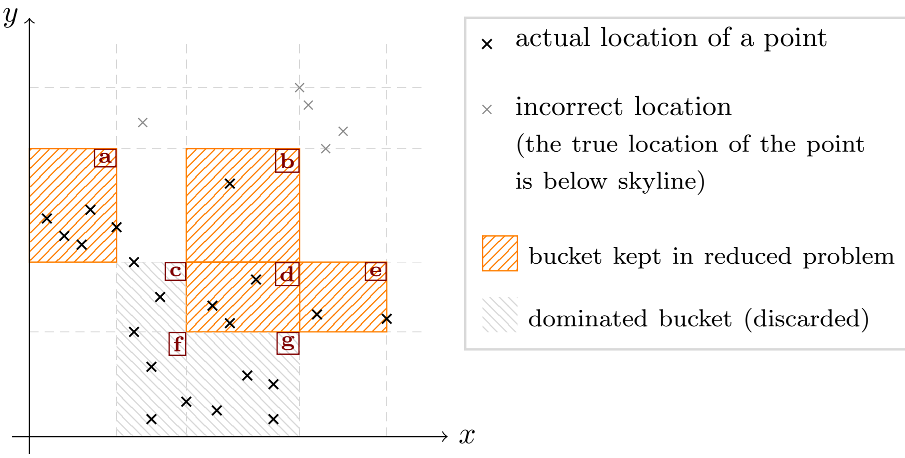

Fig. 3: An illustration of the bucket dominance and its role in **SkylineLowDim**. Here bucket b dominates c and f but not a, d, e or g. Buckets c,f,g are dominated by some non-empty bucket and therefore cannot contain a skyline point. Bucket a does not contain a skyline point, but this cannot be deduced 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...

Proof. The proof, left for the Appendix, 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 **SkylineLowDim**(X,δ) can only return the skyline in $O(nd\log(dk/\delta))$ if it is provided with a good estimate of the skyline cardinality: we must have $\hat{k} \geq k$ and $\log(\hat{k}) \in O(\log(k))$. We next show how to guarantee the complexity by trying a sequence of successive values for $\hat{k}$. The successive values in the sequence grow super exponentially (similarly to [8, 16]) to prevent failed attempts from penalizing the complexity.

Algorithm SkylineLowDim($\hat{k}$,X,δ) (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

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

Algorithm SkyLowDim-Search(X,δ) (see Theorem 6)
input: X set of points, δ error probability
output: skyline(X)
error probability: δ

$\hat{k} \leftarrow (\lfloor d/\delta \rfloor)^2$
repeat
  $\delta \leftarrow \delta/2$; $\hat{k} \leftarrow \hat{k}^2$; $S \leftarrow$ **SkylineLowDim**($\hat{k}$,X,δ)
until $|S| < \hat{k}$
Output S

Theorem 6. *Given $\delta \in (0,1/2)$ and a set X of data items,* **SkyLowDim-Search**(X,δ) *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 probability of error is $\delta/2^j$, and the cost is given by Theorem 5. Consequently, we obtain 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 [11], or our own similar **SkyHighDim-Search** 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 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 [9]. Actually, our algorithm can be viewed as some kind of generalization to higher dimensions of an algorithm from [9] which assigns points to buckets before recursing, the buckets being the quantiles along one coordinate.

5 Skyline Lower Bound

Theorem 7. *Let A be an algorithm that computes the skyline with error probability at most $1/2$. Then the expected number of queries of A is $\Omega(dn\log k)$.*

Proof. The proof is left for the Appendix.

References

1. Afrati, F.N., Koutris, P., Suciu, D., Ullman, J.D.: Parallel skyline queries. In: 15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012. pp. 274–284 (2012). https://doi.org/10.1145/2274576.2274605, https://doi.org/10.1145/2274576.2274605
2. 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. pp. 186–196. ICDT '11, ACM (2011). https://doi.org/10.1145/1938551.1938576, http://doi.acm.org/10.1145/1938551.1938576
3. Afshani, P., Barbay, J., Chan, T.M.: Instance-optimal geometric algorithms. J. ACM **64**(1), 3:1–3:38 (2017)
4. 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)
5. Barndorff-Nielsen, O., Sobel, M.: On the distribution of the number of admissible points in a vector random sample. Theory of Probability and its Applications **11**(2), 249–21 (1966), http://search.proquest.com/docview/915869827?accountid=15867
6. 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

7. 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. pp. 851–862. STOC '16 (2016)
8. Chan, T.M.: Optimal output-sensitive convex hull algorithms in two and three dimensions. Discrete & Computational Geometry **16**(4), 361–368 (1996). https://doi.org/10.1007/BF02712873, http://dx.doi.org/10.1007/BF02712873
9. Chan, T.M., Lee, P.: On constant factors in comparison-based geometric algorithms and data structures. Discrete & Computational Geometry **53**(3), 489–513 (2015). https://doi.org/10.1007/s00454-015-9677-y, https://doi.org/10.1007/s00454-015-9677-y
10. Chomicki, J., Ciaccia, P., Meneghetti, N.: Skyline queries, front and back. SIGMOD Rec. **42**(3), 6–18 (Oct 2013). https://doi.org/10.1145/2536669.2536671, http://doi.acm.org/10.1145/2536669.2536671
11. 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, https://doi.org/10.1109/SFCS.1994.365723
12. 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, https://doi.org/10.1145/2684066
13. Feige, U., Raghavan, P., Peleg, D., Upfal, E.: Computing with noisy information. SIAM Journal on Computing **23**(5), 1001–1018 (1994). https://doi.org/10.1137/S0097539791195877
14. Godfrey, P., Shipley, R., Gryz, J.: Algorithms and analyses for maximal vector computation. The VLDB Journal **16**(1), 5–28 (2007). https://doi.org/10.1007/s00778-006-0029-7, http://dx.doi.org/10.1007/s00778-006-0029-7
15. Goyal, N., Saks, M.: Rounds vs. queries tradeoff in noisy computation. Theory of Computing **6**(1), 113–134 (2010)
16. Groz, B., Milo, T.: Skyline queries with noisy comparisons. In: Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. pp. 185–198. PODS '15, ACM (2015). https://doi.org/10.1145/2745754.2745775, http://doi.acm.org/10.1145/2745754.2745775
17. Kalyvas, C., Tzouramanis, T.: A survey of skyline query processing. CoRR **abs/1704.01788** (2017), http://arxiv.org/abs/1704.01788
18. Khalefa, M.E., Mokbel, M.F., Levandoski, J.J.: Skyline query processing for incomplete data. In: 2008 IEEE 24th International Conference on Data Engineering. pp. 556–565. IEEE (2008)
19. Kirkpatrick, D.G., Seidel, R.: Output-size sensitive algorithms for finding maximal vectors. In: Proceedings of the First Annual Symposium on Computational Geometry. pp. 89–96. SCG '85, ACM (1985). https://doi.org/10.1145/323233.323246, http://doi.acm.org/10.1145/323233.323246
20. 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. pp. 275–286. VLDB '02, VLDB Endowment (2002), http://dl.acm.org/citation.cfm?id=1287369.1287394
21. 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, http://doi.acm.org/10.1145/321906.321910
22. Li, G., Wang, J., Zheng, Y., Franklin, M.: Crowdsourced data management: A survey (2016)

23. Lofi, C., El Maarry, K., Balke, W.T.: Skyline queries in crowd-enabled databases. In: Proceedings of the 16th International Conference on Extending Database Technology. pp. 465–476. ACM (2013)
24. 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, http://dx.doi.org/10.14778/2047485.2047487
25. Newman, I.: Computing in fault tolerant broadcast networks and noisy decision trees. Random Struct. Algorithms **34**(4), 478–501 (2009)
26. 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. pp. 15–26. VLDB '07, VLDB Endowment (2007), http://dl.acm.org/citation.cfm?id=1325851.1325858

Appendix for the upper bounds

Proof of Theorem 2

Proof. We denote by CheckVar(x,δ) the procedure that checks if $x = \texttt{true}$ with er. pr. δ by majority vote, and returns the corresponding boolean. We finally denote by j' the true index of the first positive variable in $x_1,\dots,x_m$. We assume the input comparison oracles with error probability δ.

The probability that **NoisyOr** fails to identify j' for $i = 2^{\lceil \log k \rceil}$ (i.e., the first time it faces variable $x_{j'}$) is at most δ'. The probability that an incorrect index is returned (before $i \geq j'$) is at most $\sum_i \delta'/2^i$. The algorithm thus returns an incorrect index with probability at most $\delta' + \sum_i \delta'/2^i \leq \delta$.

NoisyOr requires $O(i)$ comparisons at line 4, whereas CheckVar requires $O(i)$ comparisons at line 5. Replacing i with 2^h, the total cost on a successful execution is therefore $\sum_{h=1}^{\lceil \log j' \rceil} 2^h = O(j')$.

Proof of Theorem 4

Proof. For iteration j, the probability of error is $\delta/2^j$, and the cost is $O(nd\hat{k}\log(\hat{k}/\delta))$. Consequently, the probability that the algorithm fails to return the correct answer is at most: $\sum_j \delta/2^j \leq \delta$, and the running time is $O(\sum_{j=1}^{\lfloor \log k \rfloor + 1} nd2^j \log(2^j \times 2^j/\delta)) \in O(ndk\log(k/\delta))$. The complexity is $O((i'-i) * d\hat{k}\log(\hat{k}/\delta))$ to find a non-dominated point $p_{i'}$ at line 4, and $O(nd\log(\hat{k}/\delta))$ to compute the maximal point above $p_{i'}$ at line 6. Summing over all iterations, the running time and number of queries is $O(nd\hat{k}\log(\hat{k}/\delta))$.

Proof of Theorem 5

The following Lemma lists properties that our bucketing assignment satisfies with high probability. We will show in Theorem 5 that our algorithm can compute the skyline efficiently for any assignment satisfying those properties.

Lemma 2. *Assume that the samples have been correctly ordered at line 5. With error probability $\delta/\hat{k}$, the assignment performed at line 8 satisfies the following two properties:*

- *if I is a non-trivial interval (i.e., unless it matches the coordinate of a sample point), $|\{p\colon I = \mathbf{NoisySearch}(p_j, S_j, \delta'/(d\hat{k}))\}| \leq 4n/(d\hat{k}^2)$*
- *less than $2n/(d\hat{k}^2)$ points are (erroneously) assigned to buckets above the real skyline buckets.*
- *the skyline points are assigned to their correct bucket.*

Proof. Recall that $\delta' = \delta/(2d\hat{k})^5$, and that p_j denotes the j^{th} coordinate of point j. Assume the points of X are ordered w.r.t. to their j^{th} coordinate, breaking ties arbitrarily. Consider these ordered points to be divided into blocks, each one having $\ell = n/(d\hat{k}^2)$ consecutive points, except the last which may have less. In particular, the number of blocks is $d\hat{k}^2$.

Consider now the samples after line 5. Each block (but the last) contains at least one sample with probability at least $1-(1-\ell/n)^s \geq 1-\delta'/d^2\hat{k}^2$. If one sample is indeed taken from every block (except maybe the last), the distance between any two samples is at most 2ℓ. As a consequence, the number of points p that should be assigned to any given bucket is bounded by 2ℓ, except for buckets with a trivial interval because several such buckets can be merged when removing duplicates at line 6. By Chernoff bounds, the number of points assigned to wrong buckets is at most $2n/(d\hat{k}^2)$ w.p. at least $1-\delta'$. By union bound over all d dimensions and over all s intervals, we therefore have probability at least $1-3\delta'$ that one sample is taken from each block and that the total number w of points assigned to wrong buckets (over all dimensions and blocks) is less than $2n/(d\hat{k}^2)$. Consequently, with probability at least $1-3\delta'$ the assignment satisfies the first property. Indeed, for each dimension j and interval I, the number of points in I is bounded by 2ℓ (maximum distance between two samples) plus $2n/(d\hat{k}^2)$ (incorrect assignments into buckets):

$$|\{p\colon p \text{ was sorted into } I \text{ in line } 8\}| \leq 2\ell + \frac{2n}{d\hat{k}^2} = \frac{4n}{d\hat{k}^2}.$$

As for the number of buckets erroneously assumed to be non-empty, it is bounded by the number of points assigned to wrong buckets and is therefore at most $2n/(d\hat{k}^2)$.

This concludes the proof of the Lemma. We next turn to the proof of Theorem 5:

Proof. When $\hat{k}^5 \geq n$, $d^5 \geq n$ or $(\log(1/\delta))^5 \geq n$, the bounds can clearly be achieved by the other algorithms discussed previously, so we assume w.l.o.g. that $\hat{k}^5 < n$ and $d^5 < n$ and $(\log(1/\delta))^5 < n$. We evaluate the cost of the algorithm assuming that (a) the samples are correctly sorted at Line 5, (b) the assignment satisfies the properties in Lemma 2, and (c) no mistakes are made at lines 13 and 18. In other words, we only accept a few mistakes at Line 8.

Phase (i) Bucketing. Line 5: by Theorem 1 (noisy sorting) the sample is sorted in $d\cdot O(s\log(sd/\delta')) = O(nd\log(d\hat{k}/\delta))$. Line 8: by Theorem 1 (noisy search) the points are assigned to their bucket in $nd\cdot O(\log(sd\hat{k}/\delta')) = O(nd\log(d\hat{k}/\delta))$.

We will distinguish 4 kinds of (presumably) non-empty buckets (all other buckets are dropped at line 9): (i) those above the skyline that have been erroneously assigned some points, (ii) the buckets containing skyline points, (iii) the buckets that are dominated by buckets of type (ii), and (iv) the other (non-empty) buckets: they are not above the skyline but we do not have sufficient information to realize that they have no skyline points, because they are not dominated by any non-empty bucket. The algorithm is obviously not able to distinguish buckets of type (ii) and (iv), hence both are passed on to **SkylineHighDim** at line 18.

The number h of non-empty buckets is not necessarily much smaller than n as h may grow exponentially with d. Line 10 does not contribute to query complexity, but contributes $O(hd) \in O(nd)$ to the running time, using radix sort. Everything considered, the query complexity and running time of the bucketing phase are $O(nd\log(d\hat{k}/\delta))$.

Phase (ii) Eliminating irrelevant buckets. The buckets that are tested for emptiness are those of type (i), (ii) and (iv) because buckets of type (iii) are dropped at line 17. The number of buckets of type (ii) is at most k. Furthermore, a bucket can be of type (iv) iff there is one dimension i such that they share the same coordinates I_i as a skyline bucket on dimension i, and the interval I_i is not trivial. Consequently, by Lemma 2, there are at most $(dk)\times 4n/(d\hat{k}^2)$ points that belong to buckets of type (iv). The number of points in buckets of type (ii) is even smaller: when such a bucket is trivial it contains only skyline points, and when it is not trivial, there is a dimension on which it is a non-trivial interval and therefore by Lemma 2 it has at most $4n/(d\hat{k}^2)$ points, hence a total of at most $k\times 4n/(d\hat{k}^2)$ points in buckets of type (ii). When the estimate $\hat{k}$ is large enough ($k\in O(\hat{k})$), the number of points in buckets of type (ii) or (iv) is therefore $O(n/\hat{k})$. The case when this is not $O(n/\hat{k})$ because the estimate is not large enough is handled on line 15. Similarly, Lemma 2 guarantees that $O(n/\hat{k})$ points have been assigned to buckets of the first kind. Therefore, the total number of points ever considered on line 13 is $O(n/\hat{k})$. The contribution of line 13 to the complexity is therefore $O(d(n/\hat{k})\log(\hat{k}/\delta'))$ by Lemma 1. Line 17 does not contribute to query complexity, but contributes $hd\hat{k}\in O(nd\cdot\min(\hat{k},|\text{skyline}(X)|))$ to the running time.

Actually, we need to optimize a bit the algorithm to achieve that running time. There can be much more than $\hat{k}$ iterations, but there are only $\min(\hat{k},|\text{skyline}(X)|)$ "relevant" iterations in which we need to drop buckets. So we first strengthen the requirement on the order at line 10, so that a bucket comes before buckets it *weakly* dominates, where B' weakly dominates B (using the notation above) if in every dimension $\max_i\leq\max I'_i$. At line 17, if B_i has already been marked as weakly dominated, we move on to the next iteration (any bucket that B_i would dominate has already been dropped). Otherwise, we iterate through the list of remaining buckets, and we perform the following operations at a cost of $O(d)$ per bucket: we drop the buckets that B_i dominates, and mark the other buckets that B_i weakly dominates. There are only $\min(\hat{k},|\text{skyline}(X)|)$ buckets that are not weakly dominated, hence the running time.

Phase (iii) Solving the reduced problem. Finally, at line 18 the size of X' is $O(n/\hat{k})$, so its skyline can be computed in $O(nd\log(\hat{k}/\delta'))$ by **SkyHighDim-Search**.

We next show that the correct answer is returned with high probability. First, the probability that the algorithm fails to satisfy our requirements (a) to (c) above are respectively $d\cdot\delta'/d$, $\delta/\hat{k}$ and $\hat{k}\cdot\delta'/\hat{k}+\delta'$. So the conditions are met — hence the algorithm returns the correct output — with probability at least $1-4\delta$.

Appendix for the lower bounds

In this section, we exhibit an $\Omega(dn\log k)$ lower bound on the query complexity in the noisy skyline problem, denoted Skyline. To that end, we define a noisy vector problem, in which one is given k vectors each of length ℓ and needs to decide for each vector whether it is the all-zero vector. We prove a lower bound for this problem and reduce it to Skyline yielding the desired result.

5.1 (k,ℓ)-Null-Vectors: Definition and Lower Bound

In the (k,ℓ)-Null-Vectors the input S is a collection $\{\mathbf{v}^1,\mathbf{v}^2,...,\mathbf{v}^k\}\subseteq\{0,2\}^\ell$ of vectors such that for each $i\in[k]$, $\sum_{j=1}^{\ell}\mathbf{v}_j^i\leq 2$, and the output is a vector $(w_1,w_2,...,w_k)\in\{0,2\}^k$ such that for each $i\in[k]$, $w_i=\sum_{j=1}^{\ell}\mathbf{v}_j^i$. We define the distribution μ over vectors of $\{0,2\}^\ell$ as follows. For each $j\in[\ell]$, $\mu(2e_j)=1/(2\ell)$, where e_j is the canonical vector with a 1 in the j-th entry and zero elsewhere; $\mu(0,...,0)=1/2$. For inputs to (k,ℓ)-Null-Vectors, we will consider the product distribution μ^k.

Lemma 3. *For (k,ℓ)-Null-Vectors under the product distribution μ^k, if A is a deterministic algorithm with success probability at least $3/4$, then the worst case number of queries of A is $\Omega(\ell k\log k)$.*

Proof. The proof is by contradiction. Assume that A is an algorithm with success probability at least 3/4 and worst case number of queries $T\leq(\ell k\log_3 k)/1000$. We assume that the adversary is *generous*, *i.e.* the adversary tells the truth for every entry (i,j) such that $v_j^i=0$, and that lies with probability 1/3 otherwise.

Generalizing the 2-phase computational model by Feige, Peleg, Raghavan and Upfal [13], we will give the algorithm more leeway and study a 4-phase computation model, defined as follows. In the first phase, the algorithm queries every entry v_j^i $(\log_3 k)/100$ times. In the second phase, the adversary reveals to the algorithm all remaining hidden entries (i,j) such that $v_j^i=2$, except for a single random one. In the third phase, the algorithm can strategically and adaptively choose $kl/10$ entries, and the adversary reveals their true value at no additional cost. Finally, in phase 4, the algorithm outputs $w_i=2$ for every vector where it found an entry equal to 2, and $w_i=0$ for the rest of the vectors.

To see how the two models are related, observe that since $T\leq(\ell k\log_3 k)/20$, by Markov's inequality at most a set S of $\ell k/10$ entries are queried by algorithm A more than $(\log_3 k)/2$ times, so at the end of the first phase we have queried every entry at least as many times as A, except for those $\ell k/10$ entries, and in the beginning of the third phase there is all the necessary information to simulate the execution of A, adaptively finding S (and getting those values correctly), hence the success probability of the three-phase algorithm is greater than or equal to the success probability of A. Also observe that, thanks to the definition of μ and to the generosity of the adversary, any execution where all queries to a vector lead to 0 answers must lead to an output where $w_i=0$—else the algorithm would be incorrect when μ selects the null vector.

We now sketch the analysis of the success probability of the three-phase algorithm. Due to the definition of μ, with probability at least 9/10 the ground-truth input drawn from μ^k has $k/2\pm O(\sqrt{k})$ vectors that contain an entry equal to 2. At the end of the first phase, and due the fact that the adversary is generous, we have that at most of them have been identified. There remain $k/2\pm O(\sqrt{k})$ vectors that appear to be all zeroes, and about $(k/2)(1/3)^{(\log_3 k)/2}=(1/2)\sqrt{k}$ of those vectors contain a still-hidden entry whose true value is 2. During the third phase, all of those hidden 2's are revealed except for one. At that point, there still remain $k/2\pm O(\sqrt{k})$ vectors whose entries appear to be all zeroes, there is a 2 hidden somewhere uniformly at random, but all entries have been queried an equal number of times, all in vain. To find that remaining hidden entry (and therefore decide which w_i is equal to 2), the algorithm has no information to distinguish between the $\ell(k/2 \pm O(\sqrt{k}))$ remaining entries. Since, the algorithm may only select $\ell k/10$ elements to query further, the algoirthm's success probability after the fourth phase cannot be better than $(k\ell/10)/(\ell(k/2 \pm O(\sqrt{k}))) < 1/4$, a contradiction.

5.2 Reduction: Proof of Theorem 7

Step 1. Assume, for simplicity, that $d-2$ divides k. From an input $S = \{\mathbf{u}^1, \mathbf{u}^2, \ldots, \mathbf{u}^k\}$ to the (k,ℓ)-Null-Vectors, we first show how to construct an input $\mathcal{I}_S$ for Skyline with n points in d dimensions and a skyline that is likely to be of size k, where $n = (\ell + d - 2)k/(d-2)$. We first randomly permute the entries of each $\mathbf{u}^i$, by using k independent permutations, resulting in $S_\pi = \{\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^k\}$. Partition S_π into $k/(d-2)$ blocks of $d-2$ vectors, where for $j \in \{0, 1, \ldots, k/(d-2)-1\}$, block $S_\pi^j = \{\mathbf{v}^{j(d-2)+i} : i \in [d-2]\}$. For each block, define $\ell + d - 2$ points, as displayed (one point per row) on Figure 4, and the union over all blocks is the input $\mathcal{I}_S$ to the Skyline. Formally, we define point $\mathbf{p}^{(t)}$ with $t = j\ell + i$ as follows.

$$\mathbf{p}_r^{(t)} := \begin{cases} j & \text{if } r = d-1, \\ n-j & \text{if } r = d, \\ 1 & \text{if } r = i-\ell \text{ and } \ell \le i\ , \\ v_i^{j(d-2)+i} & \text{if } r = i \text{ and } i \in [d-2], \\ 0 & \text{otherwise.} \end{cases}$$

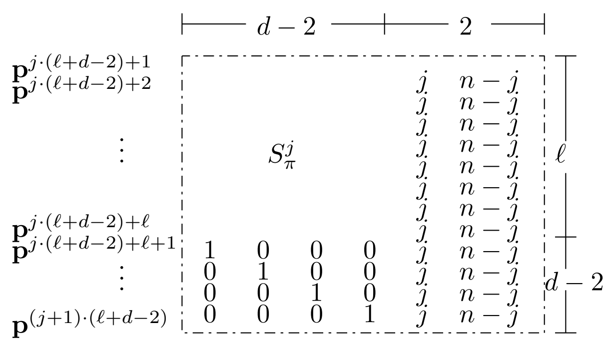

Fig. 4: Block $(j, n-j)$ of the reduction. The vectors of S_π^j placed in this block are $\mathbf{v}^{j(d-2)+1}, \mathbf{v}^{j(d-2)+2}, \ldots, \mathbf{v}^{j(d-2)+(d-2)}$.

Step 2. Because of the non-domination implied by the last two coordinates of any point, the skyline of the set of points is the sum over all blocks of the skyline of each block. Fix an arbitrary block and focus on the first $d-2$ dimensions. For each dimension, the corresponding column (whose first ℓ coordinates are those of some vector $\mathbf{v}^i$) contains exactly one 1 (on the row of some point $\mathbf{p}$) and possibly one 2 , the remaining entries being all 0. Thus it is easy to verify that $\mathbf{p}$ is part of the skyline if and only if $\mathbf{v}^i = \mathbf{0}$.

From the output skyline(I) it is now easy to construct the output of the (k,ℓ)-Null-Vectors: For all blocks, for all dimensions $\leq d-2$, if $\mathbf{p} \in \text{skyline}(I)$ then $w_i \leftarrow 0$ else $w_i \leftarrow 2$. This yields the correct output $\mathbf{w} = (w_1, w_2, ..., w_k)$. Thus we derive the following observation.

Observation 1 *Given the set of points skyline($\mathcal{I}_S$), one can recover the solution to the (k,ℓ)-*Null-Vectors *without further queries.*

Furthermore, in the following we prove that the construction is likely to have k skyline points.

Lemma 4. *Let $\mathcal{E}$ be the event that the input $\mathcal{I}_S$ has exactly k skyline points. Then, $\mathbb{P}(\mathcal{E}) \geq 1 - 1/k$ as long as $k^5 \leq n$.*

Proof. First observe that, by construction, regardless of whether $\mathcal{E}$ holds, every block contains at most $d-2$ skyline points: Consider an arbitrary block. The last two dimensions are identical for each point belonging to that block and we focus thus on the first $d-2$ dimensions. There are exactly $d-2$ points with one coordinate being 1 and all of these points are potential skyline points. In particular, take any such point $\mathbf{p}$ and assume that the i'th coordinate of $\mathbf{p}$ is 1. Then $\mathbf{p}$ is part of the skyline if and only if the vector $\mathbf{v}^i$ is the null vector. Moreover, every block can have at most $d-2$ entries with value 2 and each such 2 eliminating one potential skyline point. Thus, there are at most $d-2$ skyline points per block.

Consider the vertices $\mathbf{v}^{i_1}, \mathbf{v}^{i_1+1}, ..., \mathbf{v}^{i_2}$ of any block. We say they are *collision free* if the following holds: if $\mathbf{v}^j_{j^*} = 2$ for $j \in [i_1, i_2]$, then $\mathbf{v}^{j'}_{j^*} = 0$ for all $j' \in [i_1, i_2] \setminus \{j\}$. Observe that if the vertices of any block are collision free, then each of the first $d-2$ dimensions is dominated by a distinct skyline point and thus there $d-2$ skyline points in that block. Thus, if the vectors of every block are collision free, then there $d-2$ skyline points per block and summing up over all $k/(d-2)$ blocks, we get that there are thus k skyline points in total.

Thus, in order to bound $\mathbb{P}(\mathcal{E})$ it suffices to bound the probability that all blocks are collision free.. Recall that the random permutations $\pi_1, \pi_2, ..., \pi_k$ permute each vector $\mathbf{v}^i$ independently. Since in a block at most k^2 pairs may collide, and each collision happens with probability $1/\ell$, the expected number of *collisions* per block is at most k^2/ℓ. The expected number of collisions over all blocks is thus, by the union bound, at most $(k/(d-2)) \cdot k^2/\ell \leq 1/k$, by assumption on k. Thus, the claim follows by applying Markov inequality.

Proof (Proof of Theorem 7). Suppose for the sake of contradiction that there exists an algorithm $\mathcal{A}$ recovering the skyline for any input with exactly k skyline points, with error probability at most $1/10$, and using $o(nd\log k)$ queries in expectation. By Markov inequality, the probability that the number of queries exceeds 5 times the expectation is at most $1/5$, so truncating the execution at that point adds $1/5$ to the error probability, transforming $\mathcal{A}$ into an algorithm B that recovers the skyline for any input with exactly k skyline points, with error probability at most $1/5+1/10<1/3$, and using $o(5nd\log k)$ queries in the worst case. We claim that this implies that one can solve the (k,ℓ)-Null-Vectors with $o(nd\log k)$ w.p. at least $1/3$ contradicting Lemma 3.

Let S be the input of the (k,ℓ)-Null-Vectors. We cast S as an input $\mathcal{I}_S$ of B as described in Section 5.2. By Lemma 4, the event $\mathcal{E}$ holds w.p. at least $1-1/k$ and thus there are k skyline points.

By assumption, B can thus compute the skyline w.p. at least $1/2-1/k\geq 1/3$, where we used the Union bound. Thus, by Observation 1, one can obtain w.p. at least $1/3$ the solution to (k,ℓ)-Null-Vectors using $o(nd\log k)$ queries, a contradiction.