Skyline Queries with Noisy Comparisons

Benoit Groz[*†]
benoit.groz@lri.fr

Tova Milo[*]
milo@cs.tau.ac.il

[*]Tel Aviv University, School of Computer Science
Tel Aviv
Israel

[†]Univ Paris-Sud, LRI, UMR 8623, and INRIA
Orsay, F-91405
France

ABSTRACT

We study in this paper the computation of skyline queries - a popular tool for multicriteria data analysis - in the presence of noisy input. Motivated by crowdsourcing applications, we present the first algorithms for skyline evaluation in a computation model where the input data items can only be compared through noisy comparisons. In this model comparisons may return wrong answers with some probability, and confidence can be increased through independent repetitions of a comparison. Our goal is to minimize the number of comparisons required for computing or verifying a candidate skyline, while returning the correct answer with high probability. We design output-sensitive algorithms, namely algorithms that take advantage of the potentially small size of the skyline, and analyze the number of comparison rounds of our solutions. We also consider the problem of predicting the most likely skyline given some partial information in the form of noisy comparisons, and show that optimal prediction is computationally intractable.

1. INTRODUCTION

The rapid expansion of data generated by web users, sensor networks, and other noisy/uncertain data sources, raises new challenges for decision support systems. We focus in this paper on the computation of skyline queries - a popular tool for multicriteria data analysis - in the presence of noisy input. Given a set of data items, the skyline is the subset of items (a.k.a. Pareto optima) that are not "dominated", where an item is dominated if there is another item that is superior for every criterion. For instance, consider a scenario in which we wish to identify which cities offer the highest salaries together with high quality education. The skyline and dominated items are as illustrated in Figure 1.

Skyline queries are traditionally viewed as a problem of computing maximal vectors in a multidimensional space $\mathbb{R}^d$; data items correspond to points and each criterion corresponds to one dimension. Much research has been devoted to efficient skyline computation in the presence of exact data.

©ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *ACM PODS*, http://dx.doi.org/10.1145/2745754.2745775.

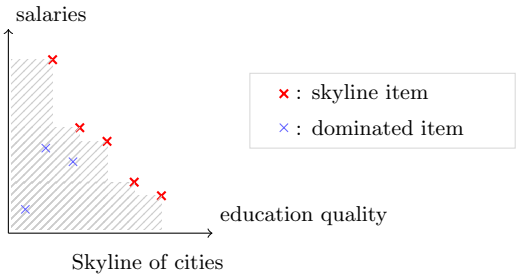


Figure 1: Skyline example

For the noisy case, the main approaches so far deal with the computational complexity of skyline queries when the input is a set of points with uncertain location. In contrast, we consider here a different setting where no information is given a priori about point location and points can only be compared, through noisy comparisons, along each dimension. We study the complexity of skyline computation here in terms of the required number of such comparisons.

This setting is motivated by the processing of skyline queries in crowdsourcing scenarios. In such settings, uncertainty is inherent, numerical estimates are not always relevant, and the focus lies in general on the cost of interaction with the crowd rather than computational complexity. To illustrate, let us consider a crowdsourcing scenario based on the example from Figure 1. Information about the average salary or schooling in different cities may be missing and people may not be able to return numerical estimates. Instead, comparing different cities may be more natural (Marcus et al. [34], for instance, show that comparisons provide more accurate rankings than ratings in certain Crowdsourcing experiments). Therefore 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.

We refer to our computation model as the *noisy comparison model*. We assume that items are fully ordered along each dimension. The order is unknown but items can be compared through oracles $(<_i)_{i\leq d}$, where $<_i$ compares a pair of items on dimension i. Each call to the comparison oracle would intuitively be implemented in our crowd scenario

by asking a new person to compare two items on a particular dimension. In order to take into account noisy answers, we model queries to the comparison oracle as i.i.d. random boolean variables that may return an erroneous answer with probability bounded away from 1/2, e.g., $p < 1/3$. The assumption here is that there is an underlying ground truth, but the oracle may make mistakes. We thus design algorithms to mitigate those mistakes. Our cost model reports the number of oracle calls required for skyline computation, rather than computational complexity. Our assumption that error is bounded away from 1/2 makes sense for real-life scenarios as it is hard to distinguish error probabilities close to 1/2 from statistical noise.

Of course the model is a simplification of actual crowd behavior and evaluating skyline queries on a real crowd raises many further issues which we leave for future work, such as estimating the error rate of workers and dealing with varying error rates. Nevertheless the formal results in this paper may serve as a yardstick on what could be expected from the performance of algorithms in refined crowdsourcing models.

Contributions. In this paper, we provide the first algorithms for skyline queries in the noisy comparison model, and evaluate their performance with respect to the following parameters:

- the number of input items n
- the dimension (number of criteria) d
- the error probability tolerated for the result δ
- the (unknown) skyline cardinality k

Specifically, we show that if we want our algorithms to return the correct answer with probability at least $1 - \delta$

- we can check if a candidate set of k items is the skyline with $O(dnk \log \frac{1}{\delta})$ or $O(dn \log \frac{dk}{\delta})$ comparisons,
- the skyline can be computed with $O(dkn \log(dk/\delta))$, $O(dk^2 n \log(k/\delta))$ or $O(dn \log \frac{dn}{\delta})$ comparisons.
- $\Omega(n \log \frac{k}{\delta})$ comparisons are necessary to check a candidate skyline in the worst case (hence also to compute the skyline).

These algorithms rely on sorting, binary search, and maxima procedures from the literature. The complexity of the corresponding problems in presence of noisy comparisons has indeed been established in [16] as $\Theta(n \log(n/\delta))$, $\Theta(\log(n/\delta))$, and $\Theta(n \log(1/\delta))$. Our results thus show that naïvely sorting the data along all dimensions and computing the skyline based on the corresponding orders is optimal for constant d when $k = \Omega(n)$, and we provide more efficient solutions when this is not the case.

We also analyze the number of rounds #rounds required by our algorithms when all comparisons whose execution has been decided at some point of the algorithm are processed in a single round in parallel. This measure is in particular relevant for crowdsourcing scenarios where questions are issued in batches, hence the number of successive batches provides some measure on the time required to complete the scenario. Obtaining low #rounds for sorting-based algorithms proved challenging. The first (and so far unique) efficient parallel sorting algorithm in presence of noise derives from the notoriously complex AKS comparator circuit network, which achieves an optimal #rounds of $O(\log n)$ rounds. We also design a simpler algorithm that does not rely on the AKS network, runs in (optimal) $O(n \log \frac{n}{\delta})$ noisy comparison, and uses $O(n^\alpha)$ rounds for any (arbitrarily small) constant $\alpha > 0$.

To achieve our results, this paper slightly extends several results from the skyline [28] and fault-tolerant sorting [32] literature to fit our arbitrary dimension and arbitrary precision setting. Finally, to complete the picture, we also consider an incremental scenario where some noisy comparisons were already performed, and the task is to process or complement the collected information in order to compute the most likely skyline. We thus prove that results from [19] about maxima computation can be extended to show that in presence of arbitrary sorting information (a multiset of noisy comparison results), it is Θ_2^p-hard for every dimension d to:

- compute the most likely skyline
- decide which additional comparison will most increase our confidence on the skyline.

This setting may in particular be relevant for our Crowdsourcing scenario if one endeavours to make the best of the comparison data available instead of computing this information from scratch.

Organization. Section 2 introduces formally our model and the problems we investigate. The results from the literature that we exploit in this paper (e.g., sorting and searching with noisy comparisons) are introduced in Section 2, whereas Section 7 provides a broader overview of related work. Section 3 investigates the complexity of verifying a candidate skyline, and those techniques are exploited in Section 4 to devise algorithms for computing skylines. The latency of our algorithms is analyzed in Section 5. Finally, Section 6 is devoted to our hardness results for the optimal exploitation of available information in skyline computation.

2. TECHNICAL PRELIMINARIES

Comparison-based results for computing skylines in the noiseless case typically rely on sorting and max algorithms and bounds [31]. In this section we first present our computation model. Then we survey skyline problems in the noiseless case. Finally we discuss sorting and max algorithms and bounds with noisy comparisons.

2.1 Model and Notations

Let S denote a set of n items. We assume these items admit a full (but not necessarily strict) order $\leq_i$ along d dimensions $i \in \{1, \ldots, d\}$. These implicit orders are not known and can only be discovered through queries of the form "is item v' superior to item v along dimensions i; i.e., $v \leq_i v'$?". We also write $v \preceq v'$ to denote that $v \leq_i v'$ for each $i \leq d$. When $v \preceq v'$ and there is some $i \leq d$ such that $v <_i v'$, we say that v' *dominates* v, which we denote by $v \prec v'$. The notation extends to any set of items C: $v \prec C$ iff there exists some $v' \in C$ such that $v \prec v'$. Finally, we denote the lexicographic order among items with $<_{\text{lex}}$: $v <_{\text{lex}} v'$ iff there is $j \leq d$ satisfying both (1) for all $i < j$, $v \leq_i v'$, and (2) $v <_j v'$.

DEFINITION 1. *Given a set of d-dimensional items S, the* skyline *of S is the set of items that are not dominated (we assume that two items can not coincide):*

$$Sky(S) = \{v \in S \mid \forall v' \in S \setminus \{v\}, \exists i \leq d.\ v >_i v'\}.$$

REMARK 1. *Skylines are a generalization to multiple dimensions of the maximum problem: for $d = 1$, the skyline $Sky(S)$ is the maximal item of S. We therefore only consider the case $d \geq 2$ in the proofs.*

Noisy comparison model. The *noisy* comparison model assumes we are given access to an oracle that takes as input a pair of items v, v' together with a dimension $i \leq d$, and answers with probability at least[1] $1 - p$ whether $v \leq_i v'$, where p is a fixed constant $p < 1/2$ (say, $p \leq 1/3$). We assume that oracle queries are independent, so that repeating a query decreases the probability of error. The *noiseless* comparison model corresponds to the particular case where $p = 0$.

We shall also consider the more limited *noisy boolean variable* model from [16] where the algorithm takes as input a set of boolean variables and an oracle providing with error probability $p \leq 1/3$ the correct value of the variables. The latter can be considered as a noisy comparison model where each item can only be compared to a specific item representing variable "0".

The problems we consider in these models take as input a parameter δ called the *tolerance*, and the algorithms must return the correct answer with error probability at most δ. Henceforth we shall abbreviate error probability as err. pr. and omit to mention that error probability is obviously allowed to be smaller than δ.

Complexity measure. Our focus is on the worst case *oracle complexity*; the number of calls to the comparison oracle. But at each step of their execution, our algorithms may require extensive computation, based on previous comparison answers, to decide which comparisons should be asked next or which answer should be returned. Unless specified otherwise, our upper bounds will therefore deal with *computational complexity*. The latter of course bounds from above oracle complexity.

Problems of interest. The problem that we wish to investigate is the following:

Skyline computation problem:
Input: S: set of n items, δ: tolerance
Objective: compute $\mathrm{Sky}(S)$ with error probability δ.

Before we tackle this Skyline computation problem, we shall address the simpler problem of checking a candidate skyline:

Skyline verification problem:
Input: S: set of n items, C: candidate set, δ: tolerance
Objective: Check $C = \mathrm{Sky}(S)$ with error probability δ.

We shall in particular investigate *output sensitive* algorithms, namely algorithms whose complexity depends on the number k of items in the skyline. Of course, we do not assume prior knowledge of this number, so the algorithm has to guess the value of k.

2.2 Complexity of noiseless skylines

Before going into the noisy model we first recall results for the noiseless case. Most results in the literature analyze the computational complexity of skyline queries rather than oracle complexity. But the lower bounds generally count the comparisons required to compute the skyline, and conversely a few algorithms guarantee a low oracle complexity under some restrictions.

Of course the oracle complexity is at most $O(dn \log n)$ since sorting the input along all dimensions solves any problem w.r.t. oracle complexity. For $d = 2$ a tight lower bound of $f_{\mathrm{sort}}(n) + n - 1$ on the oracle complexity was proved by Yao [45], where $f_{\mathrm{sort}}(n)$ denotes the number of comparisons required to sort n items. For $d = 3$, an upper bound of $2n \log_2 n + O(n)$ comparisons follows from Kung et al's algorithm [31]; a lower constant factor than the naïve sorting approach, but one checks easily that Kung et al's algorithm does not guarantee better constant factors for oracle complexity than naïve sorting beyond $d = 3$. A bound of $n \log_2 k + O(n\sqrt{\log k})$ was recently established [11], matching the information-theoretic lower bound of $n \log_2 k$ comparisons [28].

We are not aware of results on the oracle complexity of skyline for higher dimension. The question of computing an asymptotic equivalent for the oracle complexity of skylines beyond $d = 2$ is actually left open in [11]. For arbitrary large d, a few algorithms nevertheless outperform the naïve sorting approach in terms of computational and (thereby also) oracle complexity when k is small enough.

A standard skyline algorithm allows to compute the skyline in $O(dnk)$. For this we can for instance maintain a partial skyline S_i for $i = 0, \ldots, k$ containing the i greatest skyline points for lexicographic order, together with the set R_i of points that are not dominated by S_i ($S_0 = \emptyset$, R_0 is the whole input). At step i we compute S_{i+1} in $O(dn)$ from S_i by adding the largest item of R_i for lexicographic order, then compute R_{i+1} from R_i, also in $O(dn)$ by removing all items that are dominated by this largest item. This algorithm is essentially the one we shall adopt in presence of noisy comparison, except that we will not maintain R_i due to the higher cost of this screening operation in presence of noise. When R_i is not available, the computation of S_{i+1} from S_i and the set of all input items has a higher cost, though.

A shrewder algorithm with low computational complexity for small values of k and d has been proposed by Kirkpatrick and Seidel [28], based on a Divide and Conquer paradigm. The authors only investigate the complexity for constant d, but we outline in the Appendix an analysis for arbitrary dimensions:

THEOREM 1 (ADAPTED FROM [28]). *The skyline can be computed in $O(d^2 n \log^{d-2} k)$ (computational complexity)[2].*

[1] The algorithms are robust to an adversary oracle that could return a correct answer instead of an incorrect one.

[2] Throughout the paper, we abuse notations and write $d-2$ for $\max(1, d-2)$, $d-3$ for $\max(1, d-3)$, etc.

2.3 Computing in the noisy comparison model

While skyline computation has not been previously studied (to the best of our knowledge) in the noisy comparison model, several other operators were investigated: the OR problem decides if one of n boolean input variables is `true`, whereas MAX returns the maximum of the n items, SORTING returns the input items in sorted order, and TOP-k returns the k largest items (in arbitrary order). Finally, BINARY SEARCH takes as input (1) an ordered list S of n items and (2) another item v, and returns the successor of v in S.

LEMMA 1 ([16]). *The problems above can be computed within the following bounds for computational complexity, which are tight even for oracle complexity:*

OR	$\Theta(n \log \frac{1}{\delta})$
MAX	$\Theta(n \log \frac{1}{\delta})$
SORTING	$\Theta(n \log \frac{n}{\delta})$
Binary search	$\Theta(\log \frac{n}{\delta})$
TOP-k	$\Theta(n \log \frac{\min(k,n-k)}{\delta})$

Actually, it is obvious that any algorithm for the noiseless case with complexity $f(n)$ can be turned into an algorithm with tolerance δ in presence of noise by repeating each comparison requested by the algorithm $\log(f(n)/\delta)$ times and taking majority vote: each noiseless comparison will thus be simulated by a comparison with tolerance $\delta/f(n)$, hence an overall error of δ by union bound [16]. The bounds of Lemma 1 show that for the problems considered we can do better.

The algorithms above assume the input oracle has constant error probability. While our access to data is limited to noisy comparisons, we shall consider any procedure computing some boolean condition (with the help of such comparisons) as an additional oracle that can be exploited by other queries. Our algorithms thus compute compositions of boolean queries, etc. In order to optimize the cost of such compositions, we investigate the cost of *trust-preserving* algorithms, whose tolerance is determined by that of the input oracle(s): for all $\delta < 1/3$ the output must be correct with err. pr. at most δ if the input oracle(s) has err. pr. δ. Trust-preserving algorithms can be pipelined; as observed in [36] for the similar ϵ-fault-tolerant model: a trust-preserving algorithm for the composition of functions (e.g. boolean functions) can be obtained through the composition of trust-preserving algorithms for these functions. In the case of OR, Newman provided a simple trust-preserving algorithm in linear time.

LEMMA 2 ([36]). *OR can be computed in $O(n)$ with a trust-preserving algorithm that returns the variable of minimal index among the true ones (if any).*

We can derive from this simple algorithm a *trust-preserving* algorithm for MAX. Assume w.l.o.g. the input oracle has err. pr. $\delta < 1/6$. We observe that we can simulate a comparison with err. pr. $\delta/2$ by majority of 3 comparisons having err. pr. δ. Furthermore the maximum of 4 items can be computed with err. pr. $\delta/2$ in $c = O(1)$.

LEMMA 3. *MAX can be computed in $O(n)$ with a trust-preserving algorithm, as illustrated in Algorithm 1.*

Algorithm 1: Algorithm $T(n, \delta)$

1 Partition input items into groups of 4 (last group may be smaller), AND compute with err. pr. $\delta/2$ the max within each group.
2 Apply recursively $T(n/4, \delta/2)$ to these $n/4$ candidate maxima (if $n > 4$).

PROOF. The cost of $T(n, \delta)$ satisfies the equation below: $C(n, \delta) = c \cdot \lceil n/4 \rceil + C(n/4, \delta/2) \leq c \cdot n + 3 \cdot C(n/4, \delta) = O(n)$. We show by induction that $T(n, \delta)$ errs with probability at most δ: the probability that the maximum has been unduly eliminated in step 1 is $\delta/2$, and the probability that it is eliminated in step 2 is also $\delta/2$ by induction hypothesis, hence an overall tolerance δ. □

Alternatively a slightly stronger result can be obtained as follows: [18] shows that MAX can be computed with a deterministic noisy tournament tree in $O(\log \log n)$ rounds, with $O(n)$ comparisons. A simple analysis of their proof shows the algorithm to be trust-preserving (we only need to maintain the dependency on ϵ throughout their proof).

In our skyline algorithm we repeatedly compute the maximal item (for $<_1$) that is not dominated by larger items. For this, we will use the following lemma:

LEMMA 4. *Let S denote a set of n items, P a boolean property on S, and $<$ a total order on S. Given an oracle that decides P with tolerance δ in time $\alpha(\delta)$, and a similar oracle with tolerance δ for $<$ in $\beta(\delta)$, one can compute $\max\{v \in S \mid v \text{ satisfies } P\}$ with tolerance δ in $O((\alpha(\delta) + \beta(\delta))n)$.*

PROOF. One can simply view the problem as the search of the maximum for a modified total order; $<^P$, defined from P by:

- $v <^P v'$ if $v < v' \wedge P(v')$
- $v <^P v'$ if $\neg P(v) \wedge P(v')$
- when $\neg P(v) \wedge \neg P(v')$, say v and v' are ties (any arbitrary choice would do)

We can clearly simulate in $O(\alpha(\delta)+\beta(\delta))$ an oracle with tolerance δ for $<^P$, using the oracles for $<$ and P. Furthermore, the maximum for P is the item we are looking for, therefore we can solve our problem by executing the max-algorithm of Lemma 3 on order $<^P$. □

We show in the next two sections how our skyline problems can be solved using the results about sorting and computing maxima with noisy comparisons.

3. SKYLINE VERIFICATION PROBLEM.

Dominance tests are a cornerstone of our skyline algorithms. We therefore begin our exposition of skyline algorithms with two procedures for dominance testing.

LEMMA 5. *Let $C \subseteq S, v \in S$.*

1. *We can check $v \preceq C$ with err. pr. δ in $O(d|C| \log \frac{1}{\delta})$*
2. *When the order of C is known along each dimension, we can check whether $v \preceq C$ in $O(d \log \frac{d|C|}{\delta})$ oracle complexity, with err. pr. δ .*

PROOF. (1) The first procedure views dominance testing as the composition of OR queries: $v \preceq C$ is equivalent to $\bigvee_{w \in C} \bigwedge_{i \leq d} v \leq_i w$. Each dominance test $v \preceq w = \bigwedge_{i=1}^{d} v \leq_i w$ can be checked in $O(d)$ with err. pr. 1/3 using the OR algorithm from Lemma 1. Using these tests as the basic oracle of the OR algorithm, we can check $\bigvee_{w \in C} v \preceq w$ in $O(d|C| \log \frac{1}{\delta})$.

(2) Alternatively, assume we know the ordering $<_i$ for C in every dimension $i \leq d$. We can then compute with err. pr. δ/d for each $i \in \{1, \ldots, d\}$ the successor for $<_i$ of v in C. Using the binary search algorithm of Lemma 1, each successor can be computed with err. pr. δ/d in $O(\log \frac{d|C|}{\delta})$. We then deduce whether $v \preceq C$ without further oracle comparisons (though with possibly large computational cost). □

We are now ready to address the problem of checking a candidate skyline. We develop two algorithms that mostly differ on which procedure from Lemma 5 they adopt for dominance queries. The algorithms simply check the two properties (1) $C = \text{Sky}(C)$ and (2) $\text{Sky}(S) \subseteq C$ with err. pr. $\delta/2$. Both properties can be viewed as boolean combinations of dominance tests:

- $C = \text{Sky}(C)$ iff $\bigwedge_{v \neq v' \in C} \neg(v \preceq v')$
- $C \supseteq \text{Sky}(S)$ iff $\bigwedge_{v \in S} v \preceq C$.

Our first algorithm uses the first procedure of Lemma 5 for the dominance tests, whereas our second algorithm, presented below as Algorithm 2, first sorts C along all dimensions and then relies on binary search to process dominance queries

THEOREM 2. *Let $C \subseteq S$, we can check if $C = Sky(S)$*

1. *in $O(dn|C| \log \frac{1}{\delta})$ (computational complexity)*
2. *or with $O(dn \log \frac{d|C|}{\delta})$ oracle complexity.*

PROOF. We first observe that the two conditions above are necessary and sufficient to guarantee $\text{Sky}(S) = C$.
(1) Each dominance test $v \preceq C$ can be checked in $O(d|C|)$ with err. pr. 1/3 using the first procedure of Lemma 5. Using these tests as the basic oracle for the OR algorithm, we check $C \supseteq \text{Sky}(S)$ in $O(n|C| \log \frac{1}{\delta})$ calls to this dominance test, which yields $O(dn|C| \log \frac{1}{\delta})$. We use similarly the OR algorithm to check $C = \text{Sky}(Cs)$ with the same complexity. Computational and Oracle complexity are the same for this algorithm.

(2) Line 1 of Algorithm 2 runs in $O(d|C| \log(d|C|/\delta))$ according to Lemma 1 (which in turn summarizes results from [16]). In line 2 we then check $C = \text{Sky}(C)$ without any further call to the comparison oracle. When the orders computed in line 1 are correct, we can simulate in $O(d \log(d|C|/\delta))$ (oracle complexity) an oracle that for each item $v \in S$ checks with err. pr. $\delta/2$ if v is dominated by C, according to Lemma 5. Using this oracle, we can then check $\bigvee_{v \in S} v \preceq C$ with err. pr. $\delta/2$ in $O(n \cdot d \log(d|C|/\delta))$, by Newman's trust-preserving OR algorithm (see Lemma 2). This yields an overall oracle complexity of $O(dn \log(d|C|/\delta))$ for the algorithm. The computational complexity, however, may be higher due to the dominance tests and skyline computation on C (see discussion in Section 7). □

We do not have tight bounds for checking skylines in the general case, but we next prove that the second bound in Theorem 2 is optimal for constant d, whereas the first one is optimal for constant $|C|$.

Algorithm 2: Skyline verification(S, C, δ)

```
1 Sort C along each dimension with err. pr. δ/(2d)
2 if C ≠ Sky(C) according to these orderings
3     return false
4 else Check ⋀_{v∈S} v ⪯ C with err. pr. δ/2
```

PROPOSITION 1. *Let $C \subseteq S$. Checking if $C = Sky(S)$ has oracle complexity $\Omega(n \log |C| + dn \log(1/\delta))$.*

PROOF. The $\Omega(n \log_2 |C|)$ lower bound is actually a particular case of stronger bounds from the literature [4, 28]. We can also prove it directly from the information-theoretic bound in the noiseless case with $d = 2$, adapting the argument of Yao: even when items can be dominated by at most one item from C, there are at least $|C|^{n-|C|} \cdot |C|!$ possible ways to order items of C and assign each remaining items to its dominating point in C. Any algorithm checking the skyline must gather information sufficient to distinguish two such configurations, and therefore performs $\Omega(n \log_2 |C|)$ oracle comparisons (details are left for the Appendix).

The $\Omega(dn \log \frac{1}{\delta})$ lower bound derives from an immediate reduction of OR: assume that we wish to compute the disjunction of $d \times n$ noisy variables $x_{i,j}$ $(i \leq n, j \leq d)$. Let $C = \{v_0\}$ denote the unique tuple with value 1 on dimension $d+1$ and 0 on the others. For each $i \in \{1, \ldots, n\}$, let v_i denote the tuple $(x_{i,1}, \ldots, x_{i,d}, 0)$. The disjunction is true if and only if $C \neq \text{Sky}(S)$, which concludes our reduction. □

4. COMPUTING SKYLINE.

After the last section discussing skyline candidate verification, we now investigate the complexity of skyline computation. The oracle complexity of skyline computation (or actually any problem) is bounded from above by the complexity of sorting.

Algorithm 3: Full sort skyline algorithm(S, δ)

```
1 Sort C along each dimension with err. pr. δ/d
2 Deduce the skyline, assuming all orders are correct.
```

THEOREM 3. *Algorithm 3 computes $Sky(S)$ with oracle complexity $O(dn \log(dn/\delta))$.*

PROOF. Each dimension can be sorted with err. pr. δ/d in $O(n \log(dn/\delta))$ according to Lemma 1. By union bound, all orders are then correct with err. pr. δ, so that any standard algorithm for noiseless skylines can compute the skyline based on these orders without further oracle calls. □

By reduction from skyline verification (Proposition 1), this is again optimal for constant d when the skyline contains $k = \Omega(n^c)$ $(c > 0)$ items. We next turn our attention to output-sensitive algorithms, namely algorithms that perform better when k is small. Recall that k denotes the number of items belonging to the skyline. Our output-sensitive algorithms for computing the skyline rely on the following auxiliary procedure:

Algorithm 4: Skysample$(S, \hat{k}, \delta)$

```
1  S_0 ← ∅
2  for i in 0, ..., k̂ − 1
3      Compute z ← max_{<lex}{v | v ⋠ S_i} with
   err. pr. δ/k̂
4      if z = ∅ return S_i
5      else S_{i+1} ← S_i ∪ z
6  return S_k̂

   Oracle for <lex(v, v', δ):
7  Find l ← min{i | v <_i v'} with err. pr. δ/2
8  Find L ← min{i | v >_i v'} with err. pr. δ/2
9  if (L = Null or l ≼ L)
10     return true
11 else
12     return false
```

PROPOSITION 2. *Depending on the procedure adopted for dominance testing in line 3, Algorithm 4 computes the first* $\min(|Sky(S)|, \hat{k})$ *points of the skyline in decreasing lexicographic order*

1. *in* $O(dk^2 n \log(\hat{k}/\delta))$ *(computational complexity)*
2. *or with* $O(dkn \log(d\hat{k}/\delta))$ *oracle complexity.*

PROOF. Let S_i denote the set comprising the i items of $\mathrm{Sky}(S)$ having the highest rank for $<_{\mathrm{lex}}$. Lines 7 to 12 show how an oracle for lexicographic comparison can be simulated in $O(d)$ with err. pr. 1/3, using Newman's OR algorithm (see Lemma 2). Using this oracle for lexicographic order and the oracle for dominance testing of Lemma 5 as basic oracles for the max-algorithm of Lemma 4, lines 3 to 5 compute iteratively S_{i+1} from S_i with the requested complexity:

(1) Using the first procedure for dominance testing of Lemma 5 we get the new skyline item with err. pr. $\delta/\hat{k}$ in $O(d \cdot i \cdot n \log(\hat{k}/\delta))$. Overall, we thus get k skyline items with err. pr. δ in $O(\sum_{i<k} d \cdot i \cdot n \log(\hat{k}/\delta)) = O(dk^2 n \log(\hat{k}/\delta))$.

(2) Using the second procedure of Lemma 5 yields the new skyline item with err. pr. $\delta/\hat{k}$ in $O(dn \log(d\hat{k}/\delta))$. Overall, we can thus get with err. pr. δ a set of k skyline items in $O(\sum_{i<k} dn \log(d\hat{k}/\delta)) = O(dkn \log(d\hat{k}/\delta))$. □

Remark: For $d = 2$, dominance tests are essentially trivial, so the problem can be solved in a simpler way in the sense that the algorithm needs only use MAX/OR algorithms and not binary insertion. The complexity of the first algorithm is lowered to $O(kn \log(\hat{k}/\delta))$.

We next present our main algorithm for computing skylines with noisy comparisons. The idea is to exploit the SkySample algorithm from Proposition 2, but since the value of k is not known in advance, we must be careful to set a small enough value of $\hat{k} \geq k$ in order to guarantee low complexity. We thus use Chan's trick [9] to "guess" k by binary search with increasing candidate values: the i^{th} call to SkySample uses $k_i = 2^{2^i}$ instead of k, and $\delta/2^i$ instead of δ.

Algorithm 5: Skyline computation(S, δ)

```
1  i ← 1;   k_i ← 4;   compl←false
2  while compl= false
3      R ← SkySample(S, k_i, δ/2^i)
4      if |R| < k_i
5          compl←true
6      else
7          i ← i + 1
8          k_i ← 2^{2^i}
9  return R
```

THEOREM 4. *Algorithm 5 computes Sky(S)*

1. *in* $O(dk^2 n \log(k/\delta))$ *(computational complexity)*
2. *or with* $O(dkn \log(dk/\delta))$ *oracle complexity.*

PROOF. The probability of returning a wrong answer at round i is $\delta/2^i$. By union bound, the error probability sums up to at most $\sum_{i \leq \lfloor \log \log k \rfloor} \delta/2^i \leq \delta$ overall.

(1) Using the first procedure for dominance testing in Proposition 2, the computational (hence oracle) complexity of the algorithm is at most $O(dn \sum_{i=1}^{\lfloor \log \log k \rfloor} k_i^2 \log(k_i \cdot 2^i/\delta)) + O(dk^2 n \log(k/\delta)) = O(dk^2 n \log(k/\delta))$.

(2) Alternatively, if we use the second procedure for dominance testing instead, the oracle complexity of Algorithm 5 is in $O(dn \sum_{i=1}^{\lfloor \log \log k \rfloor} k_i \log(dk_i \cdot 2^i/\delta)) + O(dkn \log(dk/\delta))$, which amounts to $O(dkn \log(dk/\delta))$. □

5. DELAY IN TERMS OF ROUNDS

In the algorithms above, some oracle calls depend on the result of previous calls. This may become an issue in, e.g., crowdsourcing scenarios where tasks involve substantial delays. We therefore analyze the number of rounds required by each algorithm when all comparisons are processed simultaneously in a same round, unless their execution is determined by the outcome of some comparison that has not been processed yet, in which case it is left for future rounds. A formal definition of this number of rounds #rounds can be found in [18].

Before presenting our results we survey some previous work on #rounds for binary search and sorting, and improve some of these results, then use the improved bounds to analyze our skyline algorithms.

5.1 Parallel algorithms with noise: sorting

Feige et al. [16] already investigate the question of parallelism for MAX and Sorting with noisy comparisons when using a maximum of n processors. In our setting we instead focus on the model of Newman [36] and Goyal and Saks [18]. This model, which they apply to the MAX and OR problems, does not restrict the number of simultaneous comparisons. This model seems more relevant for our crowd scenario as we can recruit additional workers when needed. In particular, Newman [36] shows a trust-preserving OR algorithm in $O(n \log(1/\delta))$ with $O(\log^* n)$ rounds. Goyal and Saks [18] prove a corresponding $\Omega(\log^* n)$ lower bound and also propose a trust-preserving MAX algorithm in $O(n)$ with $O(\log \log n)$ rounds (the algorithm is presented for constant tolerance, but a careful analysis of their proof shows it is error preserving). The $O(n \log n)$ sorting algorithm of Feige et al. [16] mentioned in Lemma 1 relies on binary search to sort items incrementally by insertion, and therefore requires $\Omega(n \log(n/\delta))$ rounds as their binary search algorithm requires $O(\log(n/\delta))$. We first show that #rounds of binary search can be lowered to $O(\log n)$.

THEOREM 5. *Binary Search can be solved with err. pr. δ in time $O(\log(n/\delta))$ and **with** $O(\log n)$ **rounds**.*

LEMMA 6 (CHERNOFF BOUND). *Let $X_1, \ldots, X_k$ denote i.i.d. 0-1 variables with $\Pr(X_i = 1) = \rho$, where $\rho < 1/2$. Then we have*

$$\Pr(\sum_{i=1}^{k} X_i \geq k/2) \leq (2\rho e)^{k/2} e^{-\rho k}$$

PROOF. This is the standard Chernoff bound with $\mu = \rho k$, $1 + \delta = 1/(2\rho)$.

$$\begin{aligned}\Pr(\sum_{i=1}^{k} X_i \geq (1+\delta)\mu) &\leq \left(e^{\delta}/(1+\delta)^{1+\delta}\right)^{\mu} \\ &\leq e^{\delta\mu}(2\rho)^{k/2} \\ &\leq e^{k/2-\rho k}(2\rho)^{k/2}\end{aligned}$$

□

We next prove the theorem.

PROOF (ADAPTED FROM [16]). When $\delta \geq 1/n$ the result is obvious since the $O(\log(n/\delta))$ rounds of [16] satisfies our bound. We therefore restrict our attention to the case $\delta < 1/n$. Given a comparison oracle with error probability $p < 1/3$, we can use majority vote to simulate a comparison oracle with error probability at most $2^{-\log(1/\delta)/\log n}/(2e)$, in $O(\log(1/\delta)/\log n)$ and with constant number of rounds. We then process all comparisons of the binary search algorithm described in [16] with this oracle.

This algorithm interprets a binary search as a random walk down the binary search tree. In short, let $x_1 \leq x_2 \leq \cdots \leq x_n$ denote the n input items in sorted order. We fix $x_0 = -\infty$ and $x_{n+1} = \infty$. For each $i \leq n$, the i^{th} leaf of the tree is associated to the interval $[x_i, x_i + 1)$ and the interval of an internal node is the union of both children intervals. The random walk starts at the root of the tree. Each step of the algorithm compares the value of the item searched with the bounds of the children intervals and proceeds to the corresponding child. In case of a contradiction (the node appears to be outside both children intervals) the walk backtracks to the parent of the current node. Finally, when a leaf is reached, since there are no children to move to, a counter is incremented instead when the comparisons confirm the interval, and decremented if the comparisons contradict the interval.

Using our comparison oracle, we make sure the probability that each step of the random walk moves toward the correct destination with err. pr. at most $\rho = 2^{-\log(1/\delta)/\log n}/(2e)$. By the Chernoff bound therefore, after $k = 2\log n$ steps the probability that the walk does not terminate at the correct leaf is at most $(2\rho e)^{k/2} = (2^{-\log(1/\delta)/\log n})^{\log n} = \delta$. The cost of the whole algorithm is clearly $O(\log(1/\delta))$ and #rounds is $O(\log n)$. □

The next results show that very efficient parallel algorithms can be obtained while maintaining the number of comparisons within the optimal $O(n\log(n/\delta))$ bound.

THEOREM 6. *SORTING can be solved **with err. pr.** δ in time $O(n\log(n/\delta))$ and with $O(\log n)$ rounds.*

PROOF (ADAPTED FROM [32]). The proof almost literally follows the proof of [32], replacing error $1/n^2$ by δ (we assume $\delta < 1/n^2$). Their sorting algorithm relies on a recursive application of some AKS partial-sorting network whose properties are described in the following lemma.

LEMMA 7 (COROLLARY 2.1 IN [32]). *Let X be a set of size m and let $\beta = 1 - 1/(8\log 6)$. Given a comparison oracle with err. pr. ρ for any (not necessarily constant) $\rho < 1/3$, we can compute with err. pr. at most $\rho^{\Theta(\log m)}$, in time $O(m\log m)$ and with $O(\log m)$ rounds, a partition of X into disjoint sets $\{S, X_1, \ldots, X_{\overline{m}}\}$ where $\overline{m} = \Theta(m^{1-\beta})$, $|S| = O(m^{3/4})$, and $|X_1| = \cdots = |X_{\overline{m}}| = \Theta(m^{\beta})$ such that the items in X_i are smaller than items in X_j for all $i < j$.*

Step 1- Using Lemma 7, we compute with err. pr $\delta/10$ a partition $X = S \cup X_1 \cup X_2 \cup \cdots \cup X_{\overline{n}}$ such that all the items in X_i are smaller than all the items in X_j for $i < j$, where $\overline{n} = \Theta(n^{1-\beta})$. To obtain such accuracy we need for some constant c to make sure that $\rho^{c\log n} \leq \delta/10$. This can be guaranteed via majority vote by repeating each comparison $O(\log\delta/\log n)$ times, thus lowering ρ to $2^{\log(\delta/10)/(c\log n)}$. The cost of applying this step is at most $O(n\log(1/\delta))$, and #rounds remains $O(\log n)$.

Step 2 and 3- Using Lemma 3 we compute in parallel for all $i \leq \overline{n}$ the maximum item M_i of X_i with tolerance $\delta/(10\overline{n})$. We then sort $P = S \cup \{M_1, \ldots, M_{\overline{n}}\}$ using parallel MergeSort [13], repeating each comparison via majority vote so that P is sorted with err. pr $\delta/10$. Based on the sorted order of P, we can derive the approximately correct position for each item in S. In particular, we partition X as

$$X = \bigcup_{1 \leq i \leq \overline{n}} Y_i$$

where $Y_i = X_i \cup \{s \in S \mid M_{i-1} < s \leq M_i\}$ for $i < \overline{n}$ and $Y_{\overline{n}} = X_{\overline{n}} \cup \{s \in S \mid s > M_{\overline{n}-1}\}$ (and M_0 is assumed to be $-\infty$). One checks easily $|Y_i| = \Theta(n^{\beta})$. Since $|P| = O(n^{3/4})$ the cost of computing maxima and sorting P, thus building the partition, is $O(n\log(n/\delta))$ and #rounds is $O(\log n)$.

Step 4- To sort the items within Y_i recursively, we use an adaptive approach. In parallel, we recursively sort Y_i with tolerance δ^{β}. Let p_0 denote the probability that there are 2 or more unsorted groups.

$$\begin{aligned}p_0 &\leq \binom{\overline{n}}{2}\delta^{2\beta} \\ &\leq 0.5n^{2(1-\beta)} \cdot \delta^{2\beta} \\ &\leq \delta/2\end{aligned}$$

The last derivation is justified by $2\beta - 1 > 2(1-\beta)$ together with $\delta < 1/n$. Observe that δ decreases at the same rate as n so $\delta < 1/n$ is maintained through the recursive calls.

Step 5 and 6- We now detect which groups of the form Y_i remain unsorted. We check the correctness of the order reported by the recursive sorting algorithm for each i in parallel. This is achieved with err. pr. $\delta/10$ by comparing each pair of adjacent items $O(\log(n/\delta))$ times. With probability at least $1 - \delta/10$, we will detect the (probably) unique unsorted group, with constant number of rounds. We then sort this group using MergeSort, repeating each comparison via majority vote so that the group is sorted with err. pr $\delta/10$. The number of items contained in the unsorted group is at most $O(n^{\beta})$, so the cost of this sorting step is $O(n\log(n/\delta))$ and #rounds is $O(\log n)$.

Summing over all steps, we check immediately that the algorithm has error probability at most δ. The cost $C(n,\delta)$ and number of rounds $T(n,\delta)$ of the algorithm satisfy:

$$C(n,\delta) = n^{1-\beta}C(n^{\beta},\delta^{\beta}) + O(n\log(n/\delta))$$

$$T(n,\delta) = T(n^\beta, \delta^\beta) + O(\log n)$$

This gives $C(n,\delta) \in O(n\log(n/\delta))$ and $T(n) \in O(\log n)$. □

As the AKS network [5] on which this algorithm is based is notoriously complex [42], we propose another sorting algorithm in $O(n\log(n/\delta))$ that requires more rounds asymptotically but is based on a much simpler binary insertion merging procedure and therefore may be more practical. We decompose our algorithm into β stages, as illustrated in Figure 2. At stage $k \in \{0,\dots,\beta-1\}$, the algorithm receives a partition of the input into sorted groups of size $n^{\frac{k}{\beta}}$ (input sets at stage 0 are singletons, and the last group may be smaller). During this stage, the algorithm computes a partition into sorted groups of size $n^{\frac{k+1}{\beta}}$ by merging $j = n^{\frac{1}{\beta}}$ groups one by one. Let $S_1,\dots,S_j$ denote the groups to be merged. In order to merge S_i into $\bigcup_{h<i} S_h$, we execute in parallel $n^{\frac{k}{\beta}}$ binary searches with err. pr. $\delta/(\beta n)$; one for each item in S_i.

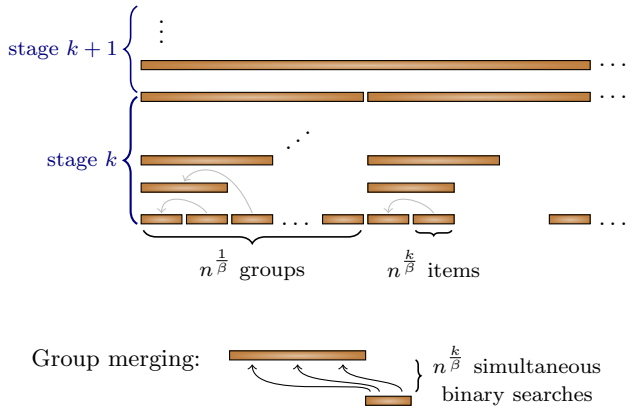


Figure 2: Parallel sort with noisy comparisons

THEOREM 7. *For every $\alpha > 0$, SORTING can be solved in $O(n\log\frac{n}{\delta})$ with $O(n^\alpha)$ rounds with a simple merging approach based on parallel insertion.*

PROOF. The binary searches within each stage can be processed in parallel because the order of items in S_h is already known, and will not be contradicted as long as the binary searches return correct answers. Over the whole execution of the algorithm, at most βn binary searches are executed (β per item), hence (1) by union bound all insertions are correct with err. pr. at most δ, and (2) the complexity of the algorithm is in $O(\beta n\log(n/\delta))$. The number of rounds during each stage is $O(n^{\frac{1}{\beta}}\log n)$ according to Theorem 5. Taking $\beta = 1 + \lfloor 1/\alpha \rfloor$ thus concludes the proof. □

5.2 Parallel algorithms with noise: skylines

We next analyze the number of rounds required by each of our skyline algorithms, when multiple comparisons can be processed simultaneously. The maximal number of comparisons in a round is roughly equal to the average number of comparisons per round (number of comparisons divided by number of rounds), as detailed in the proof (deferred to the Appendix).

THEOREM 8. *Analyzing #rounds for our algorithms yields the following results:*

- *One can check a candidate skyline C in:*

Oracle complexity	*Rounds*
$dn\lvert C\rvert\log(1/\delta)$	$\log^*(d)\cdot\log^*(n)\cdot\log^*(\lvert C\rvert)$
$dn\log(d\lvert C\rvert/\delta)$	$\log^* n\cdot\log\lvert C\rvert$

- *One can compute the skyline in:*

Oracle complexity	*Rounds*
$dn\log(dn/\delta)$	$\log n$
$dk^2 n\log(k/\delta)$	$k\cdot\log\log n\cdot\log^*(d)\cdot\log^*(k)$
$dkn\log(dk/\delta)$	$k\cdot\log\log n\cdot\log k$

where every entry is implicitly an asymptotic $O()$, and complexity refers to oracle *complexity.*

We can similarly analyze the number of simultaneous comparisons per round. We omit exact expressions.

6. EXPLOITING AVAILABLE DATA

Up until now we studied algorithms that query a comparison oracle to find the correct skyline (with a low probability of error *a priori*). We next turn our attention to the setting where some comparison information is already available and the objective is to exploit this information in an optimal way with respect to computational complexity. In this new approach, we now assume that on each dimension all orders have equal probability a priori, but the (noisy) comparison information adds to our knowledge on the hidden order, which raises the question of computing the most likely skylines given the information available. This fits for instance a scenario where the computation may be interrupted after it runs out of time or resources. It is also justified in [19] as a model of CrowdSourcing that is simpler to implement when human workers do not behave as predicted and fail to answer some of the questions due to unacceptable response time or lack of knowledge.

We henceforth call the result of a comparison a *vote*; for instance: "$v <_i v'$". Two questions of interest in our model are (1) how can we compute the most likely skyline given a multiset of votes (without executing any further comparison), and (2) given a multiset of votes, which pairs of objects should we compare next with the oracle (and on which dimension) to maximize our information on $\mathrm{Sky}(S)$. Formally, we first define problem MLsky and show it is computationally intractable in general. We then do the same for NVsky. Those two problems, standing respectively for Maximum Likelihood and Next Vote, generalize to multiple dimensions problems from [19] about maxima computation. Those problems were proved hard when $d = 1$ [19] but the case $d = 1$ is somewhat particular and it does not seem easy to cast it as a particular case of higher dimension skylines: the nature of the problem changes a bit from $d \geq 2$, with the number of possible skylines raising from n to 2^n... So we next explain how those hardness results extend to arbitrary dimension.

Maximum Likelihood Skyline (MLsky):
Input: p: err. pr., $M \in (\mathbb{N}^{S\times S})^d$: input votes
Objective: compute the most likely skyline; $P \subseteq S$ that maximizes $\Pr(\mathrm{Sky}(S) = P \mid M)$.

We first detail our setting and conventions. To simplify the proofs we assume that objects admit a full and *strict*

ordering along each dimension, and we also assume that all $(n!)^d$ d-tuples of orders are equally likely a priori (before any votes are known). The proofs generalize however to the case where ties would be allowed.

The comparison oracle again errs with probability at most p on each execution independently, for some $p \leq 1/3$. Given any d-tuple v and $k \leq d$, $v.k$ shall denote the k^{th} entry of v. The entries of $M.k$ count the answers of the oracle for the corresponding comparisons on dimension k: $(M.k)_{i,j}$ is the number of times the oracle answered "$i <_k j$". Using Bayes rule we can determine the probability a posteriori of a possible world, given the list M of vote matrices. The *possible worlds* are all d-tuples specifying an order for each dimension. To each possible world π corresponds a skyline, denoted $\text{Sky}(\pi)$, which can be computed in polynomial time using skyline algorithms on noiseless comparisons. The probability $\Pr(\text{Sky}(S) = P \mid M)$ can thus be derived by summing possible worlds in which P is the skyline:

$$\begin{aligned}\Pr(\text{Sky}(S) = P \mid M) &= \sum_{\pi:\text{Sky}(\pi)=P} \Pr(\pi \mid M) \\ &= \sum_{\pi:\text{Sky}(\pi)=P} \Pr(M \mid \pi) \cdot \Pr(\pi)/\Pr(M)\end{aligned}$$

$\Pr(\pi)/\Pr(M)$ does not depend on π, so the most likely skyline P is obtained by maximizing

$$\sum_{\pi:\text{Sky}(\pi)=P} \Pr(M \mid \pi) = \sum_{\pi:\text{Sky}(\pi)=P} \prod_{i=1}^{d} \Pr(M.i \mid \pi.i).$$

The probabilities $\Pr(M.i \mid \pi.i)$ can easily be computed [19]. The number of possible worlds, however, is exponential, even for $d = 1$, so this only provides an exponential algorithm (though in polynomial space) for computing $\Pr(\text{Sky}(S) = P)$. We next show that a polynomial algorithm is unlikely.

When $d = 1$, the problem MLsky is actually the problem of computing the most likely maximum in elections. When p is small enough, the most likely maximum coincides with the winner elected by the Kemeny voting rule [19]. The determination of the Kemeny winner is Θ_2^p-complete [23], where Θ_2^p is the class of problems solvable by a polynomial Turing machine with a logarithmic number of calls to an NP oracle. The hardness result even holds when the input votes are guaranteed to elect a single Kemeny winner. We exploit this to show that MLsky is Θ_2^p hard for every dimension d.

PROPOSITION 3. *MLsky is Θ_2^p-hard **for every** d.*

PROOF. The proof works by reduction from the unique Kemeny winner problem. We assume that votes are the same along all dimensions and guarantee a unique Kemeny winner x (obviously the same in all dimensions). We then show in the Appendix that when p is small enough, the most likely skyline consists of a single item, corresponding to the Kemeny winner x. □

We next turn our attention to the Next vote problem which decides "which comparison should be asked next" as specified in [19]. The Next vote problem computes which comparison should be executed next to maximize the expected probability that the maximum-likelihood skyline is correct. More formally, the probability that we can compute correctly the skyline after asking the additional comparison between x and y on dimension i is given by:

$$\begin{aligned}\max_{P_0} \sum_{\pi:\text{Sky}(\pi)=P_0} &\Pr(\pi \mid M \wedge x >_i y) \Pr(x >_i y \mid M) \\ + \max_{P_0} \sum_{\pi:\text{Sky}(\pi)=P_0} &\Pr(\pi \mid M \wedge x <_i y) \Pr(x <_i y \mid M)\end{aligned}$$

Applying Baye's Rule and removing irrelevant factors, one checks easily that to maximize this probability we need to maximize

$$\begin{aligned}F_{x,y,i} = \max_{P_0} \sum_{\pi:\text{Sky}(\pi)=P_0} &\Pr(M \wedge x >_i y \mid \pi) \\ + \max_{P_0} \sum_{\pi:\text{Sky}(\pi)=P_0} &\Pr(M \wedge x <_i y \mid \pi)\end{aligned}$$

We can now formally define the Next Vote problem and claim it is computationally intractable (the proof is deferred to the Appendix).

Next Vote for Skyline (NVsky):
Input: p: err. pr., $M \in (\mathbb{N}^{S\times S})^d$: input votes
Objective: compute the next comparison to ask; $(x, y, i) \in S^2 \times \{1, \ldots, d\}$ that maximizes $F_{x,y,i}$.

PROPOSITION 4. *NVsky is Θ_2^p-hard **for every** d.*

The proofs generalize to the case when ties are allowed because both hardness results rely on a reduction from the (unique) Kemeny winner problem, the complexity of which is not affected by ties [23, 22]. The fact that the two problems studied above are computationally hard motivates future study of possible approximation schemes that would be computable in polynomial time and yet provide reasonable quality guarantees.

Similarly, the other hardness results from [19] about computing the maximum imply hardness of the corresponding problems for skylines. For instance, they show that computing the associated probabilities for those two problems (computing the probability that the most likely skyline is indeed the skyline - resp., will be the skyline after the next vote is processed) is $\#P$-hard.

7. RELATED WORK

Skyline algorithms. As mentioned in the introduction, skyline queries (a.k.a. maximal vectors or Pareto maxima) have been the subject of extensive study in databases, computational geometry and multicriteria optimization. Almost all results focus on the computational complexity in the absence of noise, although lower bounds are proved through oracle complexity.

The history of worst-case-oriented comparison-based algorithms is interwoven with the history of convex hull algorithms [31, 28, 4, 11]. Kung et al. [31] proposed a Divide and Conquer algorithm with worst case complexity $O(d^2 n \log^{d-2} n)$. Kirkpatrick and Seidel then adapted the algorithm into an output-sensitive algorithm with complexity $O(d^2 n \log^{d-2} k)$, and with even better complexity when $k \ll \sqrt{n}$. These results are optimal when $d \in \{2, 3\}$ in the comparison model [31, 45], and even in the algebraic decision tree model [28]. For $d = 2$ and $d = 3$ the results were recently strengthened with instance-optimal algorithms [4].

Chan and Lee [11] raised the question of determining the oracle complexity of skyline queries, and investigate the constant factors in the $O(n \log k)$ complexity for $d = 2$ and $d = 3$. For arbitrary values of d, Sheng and Tao recently showed that skylines can be computed in $O(n \log^{d-2} n)$ computational complexity with a minor adaptation of Kung et al's algorithm [31], thus removing a d^2 factor.

Skylines can be computed faster in the RAM model [10, 2] or when the dimension is very large [47]. In particular, the skyline can be computed in $O(n \log^{d-3} n)$ expected time [10] on RAM. For $d = n$, an $O(n^{2.684})$ algorithm can be obtained through matrix multiplication techniques [35, 47]. These approaches do not seem to lower oracle-complexity because the first step of the matrix-multiplication approach is actually to sort all items along each dimension, and the RAM algorithms similarly assume rank-space reduction of the input.

After the skyline was suggested as a new operator to express preferences [7], the database community investigated the integration of this operator in database systems [12], and the performance of skyline algorithms in theory and practice in various settings [7, 30, 17]. Further from our purpose, other related problems (sampling, datastructures for subspace skyline or dominance reporting, layers of skyline), computational models and evaluation criteria have been considered, such as I/O efficient algorithms in external memory [43, 24], average-case analysis [6, 17] or progressive computation [37].

Parallel sort and parallel skyline. The number of rounds necessary to compute OR and MAX queries with optimal (linear) complexity was recently proven to be respectively $\Theta(\log^* n)$ and $\Theta(\log \log n)$ [18, 36]. Minimizing latency to compute with noisy comparisons more complex queries such as selection [18], top-k [15] or sorting [16] was formulated as an open question. The only parallel sorting algorithms in $O(n \log(n/\delta))$ with noisy comparisons we are aware of are a randomized algorithm [16] and an EREW-PRAM algorithm [32] with tolerance $\delta = 1/n^c$ for constant c. The deterministic PRAM algorithm is based on a fault-tolerant adaptation of the AKS circuits [5]. This is in sharp contrast with the noiseless case where several sorting algorithms with optimal $O(\log n)$ latency have been proposed [39, 13, 14]. Several algorithms have been suggested to leverage parallelism in (noiseless) skyline computation, see [1] and references therein, though again they generally partition the data according to the rank of items, hence seem to be of little help in our setting.

Maximum Likelihood for maxima. The problem of computing winners in elections has been investigated under a maximum likelihood perspective since -at least- the works of Condorcet [46]. When the probability of error is low, the maximum corresponds to the Kemeny winner and is therefore hard to compute [19].

Skyline algorithms with noisy information. The problem of computing skylines under noisy information was only investigated recently, though related results appeared earlier in the literature about multicriteria optimization, where the number of points needs not be finite. For instance, Papadimitriou and Yannakakis [38] show that any –possibly infinite– set of Pareto-optimal points in $\mathbb{Q}^d$ can be approximated up to any multiplicative factor $\epsilon > 0$ with a "small" set of solution points, and subsequent work [44] investigates algorithms and bounds when the objective is to minimize the size of the approximated solution. Similar bounds and results were established for the approximation of skylines, where the instance is a finite set of n points [29]: a smallest ϵ-approximation (cover) of the skyline can be computed greedily in two dimensions, but this problem becomes NP-hard from $d = 3$. However one can always find an ϵ-approximation of the skyline that is larger than the minimal one by a factor at most $\log n$, or even by a constant factor when $d = 3$.

Pei et al. [40] introduce the ρ-skyline as the set of points that have probability at least ρ to belong to the skyline, and propose heuristics for their computation. The ρ-skyline has mostly been investigated in the "locational" model where the location of each of the n uncertain points is independently defined by a discrete probability distribution over k possible points: each uncertain point P is thus defined as (1) a k-tuple of probabilities summing up to one, together with (2) a k-tuple of vectors in $\mathbb{R}^d$ representing the corresponding locations. The ρ-skyline can then be computed in $(nk)^{2-1/d}$, or alternatively in $O(\min(n, k)nk \log(nk))$ [3]. Afshani et al. [3] also devise some approximation algorithms for ρ-skylines. More heuristic approaches have also been suggested for skyline computation on incomplete or imprecise data [27, 33].

Alternative noisy comparison model. Multiple models have been developed that aim to guarantee some form of fault-tolerance in computation. Variations of the noisy comparison model have been investigated in particular on decision trees [16, 18, 26] and broadcast networks [36] for sorting problems [16], maxima computation [16, 18] and boolean function evaluation [16, 26, 36, 18].

Sorting and selection problems were also considered in other noise models; some assume for instance an upper bound k on the number of errors instead of random errors [41]. Another extends the noisy comparison model in the context of clustering and top-k query processing through Crowdsourcing [15], so that the probability of error depends on some distance function between the items compared: the motivation for this extension being that crowd workers are more likely to make mistakes when two items are similar. Many results also deal with the particular case where the algorithm must be implemented as a network of comparators. The questions and techniques considered in Section 6 are connected to optimization problems such as minimum feedback-arc set on graphs, and related voting theory questions [8]. Hardness results and heuristics have recently been suggested for those problems in the framework of crowdsourcing [19].

8. CONCLUSION AND FUTURE WORK

We introduced several algorithms and bounds for skyline queries with noisy comparisons. We showed that sorting the full dataset is asymptotically optimal when the dimension is fixed and the skyline contains at least a constant fraction of the input. But efficient algorithms can be obtained when the skyline cardinality is small.

The quadratic or cubic number of comparisons required by our algorithms may surprise in view of the traditionally lower complexity claimed for noiseless skylines. To explain this gap, we point out that (1) traditional skyline bounds

generally hide constants depending on d as d is generally fixed (and often limited to $d \leq 3$), and (2) most skyline algorithms rely on Divide and Conquer schemes that do not work well in presence of noisy comparisons. Specifically, two bottlenecks here are the accurate computation of the *screening* (identifying all dominated points) and sorting procedures: identifying all dominated items or sorting accurately a large number of non-trivial subsets both have the same complexity as sorting the whole input set.

Our algorithms and bounds on the number of comparisons mostly build upon the literature of sorting and searching with noisy comparisons. The gap between those algorithms and lower bounds highlight numerous open questions, in both the output-sensitive and general cases. First of all, can skylines be computed with cost $o(nk)$ in the planar case, or even for any constant d? How does the number of comparison vary with d? Can we establish some tradeoff between rounds and questions similar to the one observed for maxima queries?

Getting the right value of error parameters is a typical issue, e.g., in probabilistic databases. In crowdsourcing scenarios for example, the simplest approach adopted by practitioners is to sample users on a set of questions for which the answers are known. Of course there are more sophisticated schemes and this is an interesting question but outside the scope of this paper. Other possible directions for future works include the analysis of average-case and other models of computations, for instance restricting the comparisons authorized to a subset of all item pairs and dimensions [20, 21, 25], hybrid models allowing both comparisons and numerical values, etc. The model should also be extended in several directions to address issues of real-world crowdsourcing scenarios: taking into account varying error rates among users, dependencies between dimensions and correlation between errors, etc.

Acknowledgements

This work has been partially funded by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071, and by the Israel Ministry of Science.

9. REFERENCES

[1] F. N. Afrati, P. Koutris, D. Suciu, and J. D. Ullman. Parallel skyline queries. In *ICDT*, pages 274–284, 2012.

[2] P. Afshani. Fast computation of output-sensitive maxima in a word ram. In *SODA*, pages 1414–1423, 2014.

[3] P. Afshani, P. K. Agarwal, L. Arge, K. G. Larsen, and J. M. Phillips. (approximate) uncertain skylines. *Theory Comput. Syst.*, 52(3):342–366, 2013.

[4] P. Afshani, J. Barbay, and T. M. Chan. Instance-optimal geometric algorithms. In *FOCS*, pages 129–138, 2009.

[5] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in c log n parallel sets. *Combinatorica*, 3(1):1–19, 1983.

[6] J. L. Bentley, K. L. Clarkson, and D. B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. *Algorithmica*, 9(2):168–183, 1993.

[7] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[8] M. Braverman and E. Mossel. Noisy sorting without resampling. In *SODA*, pages 268–276, 2008.

[9] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996.

[10] T. M. Chan, K. G. Larsen, and M. Patrascu. Orthogonal range searching on the ram, revisited. In *SOCG*, pages 1–10, 2011.

[11] T. M. Chan and P. Lee. On constant factors in comparison-based geometric algorithms and data structures. In *30th Annual Symposium on Computational Geometry, SOCG'14, Kyoto, Japan, June 08 - 11, 2014*, page 40, 2014.

[12] J. Chomicki, P. Ciaccia, and N. Meneghetti. Skyline queries, front and back. *SIGMOD Record*, 42(3):6–18, 2013.

[13] R. Cole. Parallel merge sort. *SIAM J. Comput.*, 17(4):770–785, 1988.

[14] R. Cole. Correction: Parallel merge sort. *SIAM J. Comput.*, 22(6):1349, 1993.

[15] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, pages 225–236, 2013.

[16] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994.

[17] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDB J.*, 16(1):5–28, 2007.

[18] N. Goyal and M. Saks. Rounds vs. queries tradeoff in noisy computation. *Theory of Computing*, 6(1):113–134, 2010.

[19] S. Guo, A. G. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD Conference*, pages 385–396, 2012.

[20] A. Gupta and A. Kumar. Sorting and selection with structured costs. In *FOCS*, pages 416–425, 2001.

[21] A. Gupta and A. Kumar. Where's the winner? max-finding and sorting with metric costs. In *APPROX-RANDOM*, pages 74–85, 2005.

[22] E. Hemaspaandra, L. A. Hemaspaandra, and J. Rothe. Hybrid elections broaden complexity-theoretic resistance to control. *CoRR*, abs/cs/0608057, 2006.

[23] E. Hemaspaandra, H. Spakowski, and J. Vogel. The complexity of kemeny elections. *Theor. Comput. Sci.*, 349(3):382–391, 2005.

[24] X. Hu, C. Sheng, Y. Tao, Y. Yang, and S. Zhou. Output-sensitive skyline algorithms in external memory. In *SODA*, pages 887–900, 2013.

[25] Z. Huang, S. Kannan, and S. Khanna. Algorithms for the generalized sorting problem. In *FOCS*, pages 738–747, 2011.

[26] C. Kenyon and V. King. On boolean decision trees with faulty nodes. *Random Struct. Algorithms*, 5(3):453–464, 1994.

[27] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski. Skyline query processing for incomplete data. In *ICDE*, pages 556–565, 2008.

[28] D. G. Kirkpatrick and R. Seidel. Output-size sensitive algorithms for finding maximal vectors. In *SOCG*, pages 89–96, 1985.

[29] V. Koltun and C. H. Papadimitriou. Approximately dominating representatives. *Theor. Comput. Sci.*, 371(3):148–154, 2007.

[30] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.

[31] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.

[32] F. T. Leighton, Y. Ma, and C. G. Plaxton. Breaking the theta (n $\log^2$ n) barrier for sorting with faults. *J. Comput. Syst. Sci.*, 54(2):265–304, 1997.

[33] C. Lofi, K. E. Maarry, and W.-T. Balke. Skyline queries in crowd-enabled databases. In *EDBT*, pages 465–476, 2013.

[34] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[35] J. Matousek. Computing dominances in e^n. *Inf. Process. Lett.*, 38(5):277–278, 1991.

[36] I. Newman. Computing in fault tolerant broadcast networks and noisy decision trees. *Random Struct. Algorithms*, 34(4):478–501, 2009.

[37] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[38] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *FOCS*, pages 86–92, 2000.

[39] M. Paterson. Improved sorting networks with o(log n) depth. *Algorithmica*, 5(1):65–92, 1990.

[40] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.

[41] A. Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002.

[42] J. I. Seiferas. Sorting networks of logarithmic depth, further simplified. *Algorithmica*, 53(3):374–384, 2009.

[43] C. Sheng and Y. Tao. Worst-case i/o-efficient skyline algorithms. *ACM Trans. Database Syst.*, 37(4):26, 2012.

[44] S. Vassilvitskii and M. Yannakakis. Efficiently computing succinct trade-off curves. *Theor. Comput. Sci.*, 348(2-3):334–356, 2005.

[45] F. F. Yao. On finding the maximal elements in a set of plane vectors. Technical Report UIUCDCS-R-74-667, University of Illinois, 1974.

[46] H. P. Young. Condorcet's Theory of Voting. *American Political Science Review*, 82:1231–1244, 1988.

[47] R. Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted apsp. In *SODA*, pages 950–957, 2009.

Appendix

Proof of Theorem 1

We next evaluate the dependence on d of the output-sensitive algorithm of Kirpatrick and Seidel. We recall that most results in [28] only hold for constant d. In particular, the proofs of their Lemma 3.1 and Theorem 3.1 are not valid for non-constant d in spite of the deceptive d factors maintained along these proofs.

We follow closely the proof of [28] and only highlight the results on which we depart from the original proof. Adopting the notations of [28], let $C_d(k,n)$ denote the complexity of computing the skyline of n items when the (unknown) number of skyline items[3] is k, and let $F_d(r,s)$ denote the complexity of determining which items of S are dominated by some item(s) of R when $S, R \subseteq \mathbb{R}^d$, $|S| = s$, $|R| = r$. Let also $G_d(r,s,t)$ denote the complexity of Algorithm SCREEN as defined in [28].

The algorithm Max1 in [28] proceeds as follows to compute the skyline of a d-dimensional n-items set T:

- partition T around the median for lexicographic order into S and R such that $x <_{\text{lex}} y$ for all $(x,y) \in S \times R$
- compute recursively $M(R) = \text{Max1}(R)$
- remove from S the items dominated by $\text{Max1}(R)$
- compute $M(S) = \text{Max1}(S)$
- return $M(R) \cup M(S)$.

The partition can be computed in $O(nd)$ using a linear selection algorithm, with lexicographic comparisons of cost $O(d)$ each. We thus deduce, by setting $r = |M(R)|$:

LEMMA 8 (FROM LEMMA 2.2 IN [28]). *When parameter r ranges over $\{1, \ldots, \min(k-1, n/2)\}$,*

$$C_d(k,n) \leq \max_r (C_d(r,n/2) + F_{d-1}(r,n/2) + C_d(k-r,n/2) + O(dn))$$

Following the proof in [28] and taking care of the cost $O(d)$ for lexicographic comparisons, we also show easily that:

LEMMA 9 (FROM LEMMA 3.1 IN [28]). *For all values of d we have:*

(i) $G_d(1,s,t) \in O(ds)$

(ii) For $d \geq 3$ and $2 \leq t \leq r$:

$$G_d(r,s,t) \in O(d^2(s + r(t-1)^{d-3})(\log r)(\log_t r)^{d-3}).$$

We imediately deduce:

LEMMA 10 (FROM COROLLARY 3.1 IN [28]). *For all values of d we have:*

$$F_d(r,s) \in O(d^2(r+s)(\log r)^{d-2}).$$

We next turn to the proof of the theorem, which is where we differ most from the original proof. For $d = 2, 3$ the complexity of skyline computation is $C_d(k,n) = O(n \log k)$ [28], and we next prove the theorem for $d \geq 4$ by induction. By Lemmas 8 and 9, when r ranges over $\{1, \ldots, \min(k-1, n/2)\}$,

$$C_d(k,n) \leq \max_r (C_d(r,n/2) + C_d(k-r,n/2) + O(d^2 n \log^{d-3} r))$$

We distinguish two cases:

[3] for consistency with our paper we denote this variable by k instead of v.

Case 1: $d-3 \geq \log r$. Then by induction there are constants $c, c' > c$ such that

$$\begin{aligned} C_d(k,n) &\leq \max_r (c' d^2 n(\log^{d-2} r + \log^{d-2}(k-r))/2 \\ &\qquad + c(d^2 n \log^{d-3} r)) \\ &\leq c' d^2 n(1 + \log^{d-2}(k-1))/2 \\ &\qquad + c(d^2 n \log^{d-3} k) \\ &\leq c' d^2 n(\log^{d-2} k) \end{aligned}$$

The middle step is justified because $r \mapsto \log^{d-2} r$ is convex and increasing, hence $r \mapsto \log^{d-2} r + \log^{d-2}(k-r)$ reaches its maximum in our range for $r = 1$.

Case 2: $d-3 \leq \log r$. Then by induction there are constants $c, c' > c$ such that

$$\begin{aligned} C_d(k,n) &\leq \max_r (c' d^2 n(\log^{d-2} r + \log^{d-2}(k-r))/2 \\ &\qquad + c(d^2 n \log^{d-3} r)) \\ &\leq c' d^2 n(\log^{d-2}(k/2)) \\ &\qquad + c(d^2 n \log^{d-3} k) \\ &\leq c' d^2 n(\log^{d-2} k) \end{aligned}$$

The middle step is justified because $r \mapsto \log^{d-2} r$ is concave.

We have thus shown that for some constant c', $C_d(k,n) \leq c' d^2 n(\log^{d-2} k)$ which concludes our proof. □

Proof of Proposition 1

The $\Omega(n \log_2 |C|)$ lower bound is actually a particular case of stronger bounds from the literature: Kirkpatrick and Seidel [28] show a $\Omega(n \log_2 |C|)$ lower bound for the stronger *algebraic decision tree model* by reduction from the multiset size verification problem. Afshani et al [4] prove a refined $n\mathcal{H}(S)$ bound in the comparison tree model for $d = 2$, where $\mathcal{H}(S)$ is the entropy-like expression defined as follows. $\mathcal{H}(S)$ is the minimal value of $\sum_k (|S_k|/n) \log(n/|S_k|)$ for a partition $(S_k)_k$ of S such that each subset S_k is either a singleton or an axis-aligned box. But the weaker bound that we need can be proved directly through an information-theoretic argument adapting the proof of Yao [45]: There are at least $|C|^{n-|C|} \cdot |C|!$ possible ways to order items of C and assign each remaining items to its dominating point in C such that each item is dominated by at most one item from C. Any deterministic algorithm that given an arbitrary input S, C checks if C is the skyline of S must gather information sufficient to distinguish two such configurations, and therefore performs $\Omega(n \log_2 |C|)$ oracle comparisons. To see why two configurations as above must be distinguished: let $<_1, <_2$ be some orderings of S such that $C = \mathrm{Sky}(S)$ and each item is dominated by at most one item from C. We next show that

1. for every $v' \notin \mathrm{Sky}(S)$ the comparisons of any skyline verification algorithm must allow to determine some $v \in C$ such that $v' \prec v$
2. the comparisons must order all items of $\mathrm{Sky}(S)$ on both $<_1$ and $<_2$.

For each $v \in C$ let D_v be the set of items $v' \prec v$ such that the comparisons performed by the algorithm do not allow to determine $v' <_1 v$. If $D_v \neq \emptyset$ then we can shift all items v' of D_v so that $v' >_1$ without falsifying the results of any comparison performed, to the effect that D_v now contributes to $\mathrm{Sky}(S)$. This provides an instance which the algorithm cannot distinguish from S, whose skyline differs from C. A contradiction. The same is true on $<_2$ which concludes the proof of the first property. The second property is established similarly. □

Proof of Theorem 8

The only part of our algorithms that we actually parallelize are essentially the dominance checking and sorting procedures. We therefore revisit the procedures for dominance checking and then deduce #rounds for our algorithms. The first procedure for dominance checking in Lemma 5 can clearly be executed in $O(\log^*(d) \cdot \log^*(|C|))$ rounds each involving at most $O(d|C| \log \frac{1}{\delta})$ simultaneous comparisons. The second procedure requires $\log \frac{d|C|}{\delta}$ rounds, each involving at most $O(d \log \frac{d}{\delta} / \log |C|)$ simultaneous comparisons.

Skyline candidate verification:

1. The composition of the 3 applications of the OR algorithm presented in Theorem 2 can clearly be executed in $O(\log^*(d) \cdot \log^*(n) \cdot \log^*(|C|))$ rounds.
2. We can sort C along all dimensions with $O(\log |C|)$ rounds, according to Theorem 6. Each oracle call can then be processed in $O(\log^* n \cdot \log |C|)$ rounds.

Skyline computation: The first item below discusses the trivial approach that sorts the whole dataset whereas the second and third items analyze #rounds for the procedures in Proposition 2 and thereby Theorem 4. The oracle for lexicographic order can be implemented in $O(\log^* d)$ rounds, and therefore has no impact on cost since both procedures for domination oracles require more.

1. We can sort the whole dataset along all dimensions with $O(\log n)$ rounds, according to Theorem 6.
2. The procedure for dominance testing through composition of OR queries requires $O(\log^*(d) \cdot \log^*(k))$ rounds. Thus, each new point can be computed in $O(\log \log n \cdot \log^*(d) \cdot \log^*(k))$ rounds. This yields a total of $O(k \cdot \log \log n \cdot \log^*(d) \cdot \log^*(k))$ rounds for SkySample, hence for the computation of skylines too.
3. To minimize the number of rounds we compute the ordering of S_{i+1} from that of S_i, inserting the new point through binary search, hence #rounds $= k \log(dk/\delta)$ for sorting partial skylines. Then each oracle call for dominance test can be processed in $O(\log(\hat{k}))$ rounds. This yields a total of $O(k \cdot \log \log n \cdot \log \hat{k})$ rounds for SkySample, hence the result. □

Proof of Proposition 3

Before we discuss the proof, we first present the *Kemeny Winner problem*. The input consists of an $n \times n$ vote matrix M with $M_{i,j}$ counting the number of votes for $i < j$. As in our noisy comparison problems, votes can contradict each others, but in contrast to our problems, there are no probabilities involved. The rankings that contradict the fewest votes are called *Kemeny permutations*, and the *Kemeny Winner problem* is the problem of deciding whether there is some Kemeny permutation in which the given object is ranked first. The *unique Kemeny Winner problem* is the same problem restricted to instances for which the Kemeny Winner is unique. Both the Kemeny Winner [23] and the unique Kemeny Winner [22, Appendix] problems are Θ_2^p-

hard (and actually Θ_2^p-complete when votes are given as a *list* of preference rankings [23, 22]).

PROOF (ADAPTED FROM [19]). We adapt to arbitrary d the proof from [19]. As already mentioned, for $d = 1$ the result is immediate from [19] which proves that by setting $p < 1 - 1/(1 + \frac{1}{n!})$, any most likely maximum is a Kemeny winner. To the best of our knowledge, our extension of this hardness result to skylines in arbitrary dimension is new.

To extend the proof to arbitrary values of d, we use instead a reduction from the unique Kemeny Winner problem and set $p < 1 - 1/(1 + \frac{1}{n!(2d+1)})$. From the vote matrix M_{Kem} that guarantees a unique Kemeny winner, we build a tuple of d identical matrices $M.1 = \cdots = M.d = M_{\text{Kem}}$. From the argument in [19, Theorem 2], one deduces easily that the most likely skyline consists of a single item; the Kemeny winner. We say that a possible world π is *Kemeny* if for all $i \leq d$ $\pi.i$ is a Kemeny permutation. We wish to prove that the probability of Kemeny possible worlds exceeds that of non-Kemeny worlds when p is small enough.

On each dimension i, for any Kemeny permutation π' and non-Kemeny π'',

$$\Pr(\pi.i = \pi' \mid M) \cdot \frac{p}{1-p} \geq \Pr(\pi.i = \pi'' \mid M)$$

Summing over non-Kemeny permutations, we deduce:

$$\Pr(\pi.i = \pi' \mid M) \cdot n! \cdot \frac{p}{1-p} \geq \sum_{\pi'' \text{not Kemeny}} \Pr(\pi.i = \pi'' \mid M)$$

Consequently, when $p < 1 - 1/(1 + \frac{1}{n!(2d)})$, the probability that $\pi.i$ is not Kemeny is strictly less than $1/(2d)$. Hence a probability greater than $1/2$ that π is Kemeny. By construction of M, when π is Kemeny, the skyline $\text{Sky}(\pi)$ consists of a single item; the Kemeny winner of M_{Kem}. Therefore, the most likely skyline is the Kemeny winner of M_{Kem}. □

Proof of Proposition 4 (adapted from [19])

The proof for arbitrary d exploits exactly the same ideas as the one for $d = 1$ in [19], but considering several dimensions opens more possibilities for the choice of the next comparison, and our proof also fixes some imprecisions[4]. In order to generalize the result to higher dimensions, we again replace the reduction from the Kemeny winner problem with a reduction from the *unique* Kemeny winner problem, and replicate the vote matrix along all dimensions.

We are given an $n \times n$ vote matrix W for which we are guaranteed there exists a unique Kemeny winner. We first replicate along all dimensions this matrix to create a new matrix W', then add an additional item v to W', such that for all dimensions $i \leq d-1$, $W'[i]$ contains a vote $v >_i w$ for every item w. Finally, we replicate 3 times each vote in W' so that adding a new vote will never turn a non-Kemeny permutation into a Kemeny one. Note that this replication step preserves Kemeny permutations, and thereby the Kemeny winner. By definition, v is the Kemeny winner in $W'[j]$ for all $j < d$, and one of the two Kemeny winners in $W'[d]$. Recall, however, that our goal is to return a Kemeny winner in W, not in $W'[d]$.

Let us show that the Next Vote problem on W' returns the comparison on dimension d of v and v' where v' is the Kemeny winner in W. We first observe that $\Pr(W' \wedge x >_i y \mid \pi) = \Pr(W' \mid \pi) \Pr(x >_i y \mid \pi)$ for all x, y and i. Let (x, y, i) be the comparison returned by the next vote problem on W', comparing x and y on dimension i. We want to show that $\{x, y\} = \{v, v'\}$ and $i = d$. We assume that the probability of error p is small enough that the probabilities associated to non-Kemeny possible worlds are negligible. More formally, let $P_1 = \{\pi \mid \exists j \leq d.\ \pi.i \text{ is not Kemeny for } W'[i]\}$ and let $\pi_{Kem} \notin P_1$. Then we assume p is small enough that $\Pr(W' \mid \pi_{Kem}) > \sum_{\pi \in P_1} \Pr(W' \mid \pi)$. We also assume that $\Pr(W' \mid \pi_{Kem})(1-p) > \sum_{\pi}(W' \mid \pi_{Kem})np$. Let $i = d, x = v, y = v'$, let c be the number of Kemeny permutations in W, and let π_0 be an arbitrary Kemeny permutation of W. Let π_1 denote the possible world such that $\pi_1.j$ is obtained by placing v on top of π_0 for all $j \leq d$. Let also π_j $(2 \leq j \leq n+1)$ be the possible world that only differs from π_1 on dimension d, such that $\pi_j.d$ is obtained from π_0 by inserting v on the j^{th} position.

For the values of i, x, y specified above we thus obtain

$$\begin{aligned}
F_{v,v',d} &\geq \sum_{\pi:\text{Sky}(\pi)=\{v\}} \Pr(W' \wedge v >_d v' \mid \pi) \\
&\qquad + \sum_{\pi:\text{Sky}(\pi)=\{v,v'\}} \Pr(W' \wedge v <_d v' \mid \pi) \\
&\geq \sum_{\pi:\text{Sky}(\pi)=\{v\}} \Pr(W' \mid \pi)(1-p) \\
&\qquad + \sum_{\pi:\text{Sky}(\pi)=\{v,v'\}} \Pr(W' \mid \pi)(1-p) \\
&\geq (c + cn)c^{d-1} \Pr(W' \mid \pi_1)(1-p)
\end{aligned}$$

Let us justify the derivation. By construction, the most likely skyline in W can either be $\{v, v'\}$ or $\{v\}$, depending on whether $v <_d v'$. In every Kemeny possible world π for $W' \wedge v >_d v'$, v is the maximal item in every dimension so $\text{Sky}(\pi) = \{v\}$; the most likely skyline is $P_0 = \text{Sky}(\pi)$. In every Kemeny possible world π for $W' \wedge v <_d v'$, v is the maximal and v' the second item in every dimension except in dimension d where v' is first and the position of v is arbitrary. In this case the most likely skyline is $P_0 = \{v, v'\}$. Consider the left term first: there are c^d Kemeny possible worlds π such that $\text{Sky}(\pi) = \{v\}$. The probability of getting votes W' and vote $x >_i y$ given any such possible world equals $\Pr(W' \mid \pi_1)(1-p)$. Now, we observe that $\Pr(W' \mid \pi_j) = \Pr(W' \mid \pi_1)$ for all $j \geq 2$ so the second term, $\sum_{\pi:\text{Sky}(\pi)=\{v,v'\}} \Pr(W' \mid \pi)(1-p)$, is at least $nc^d \Pr(W' \mid \pi_1)(1-p)$. This justifies the derivation above.

On the other hand, we next show that $F_{x,y,i}$ is smaller for every other value of $\{x, y\}$ and i. We first recall that by construction any additional vote can only reduce or preserve the set of Kemeny permutations and observe that $P_0 = \{v, v'\}$ becomes a most likely skyline for both $W' \wedge x <_i y$ and $W' \wedge x >_i y$. This is because for both comparison outcomes there are as many Kemeny possible worlds consistent with the outcome that satisfy $v' >_d v$ as there are that satisfy $v' <_d v$. Then it is clear that every Kemeny permutation for $W[i]$ (hence every Kemeny possible world) is contradicted by one of $x <_i y$ or $x >_i y$. As a consequence $F_{x,y,i}$ is equal to $nc^d \Pr(W' \mid \pi_1)((1-p)+p)$ plus negligible contributions from non-Kemeny possible worlds. This is smaller than $F_{v,v',d}$ since $np \ll 1 - p$. □

[4] in particular, we insist that $W \wedge a$ is the conjunction of events W and a, not a vote matrix obtained by combining the vote multisets W and a.