Skyline in Crowdsourcing with Imprecise Comparisons*

Aris Anagnostopoulos
Sapienza University of Rome
Italy

Adriano Fazzone
Sapienza University of Rome
Italy

Giacomo Vettraino
Sapienza University of Rome
Italy

ABSTRACT

Given an input of a set of objects each one represented as a vector of features in a feature space, the problem of finding the *skyline* is the problem of determining the subset of objects that are not dominated by any other input object. An example of an application is to find the best hotel(s) with respect to some features (location, price, cleanliness, etc.)

The use of the crowd for solving this problem is useful when a score of items according to their features is not available. Yet the crowd can give inconsistent answers. In this paper we study the computation of the skyline when the comparisons between objects are performed by humans. We model the problem using the *threshold model* [1] in which the comparison of two objects may create errors/inconsistencies if the objects are close to each other. We provide algorithms for the problem and we analyze the required number of human comparisons and lower bounds. We also evaluate the effectiveness and efficiency of our algorithms using synthetic and real-world data.

CCS CONCEPTS

• **Information systems → Crowdsourcing**; • **Theory of computation** → *Models of computation.*

KEYWORDS

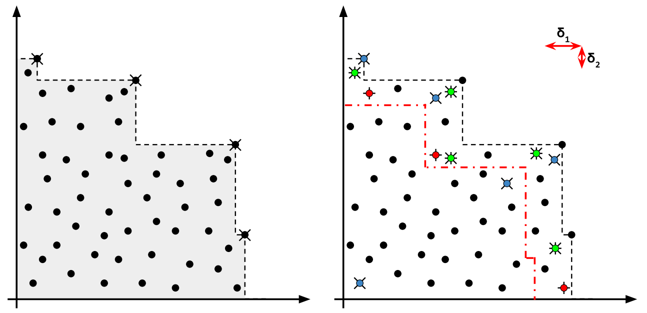
Crowdsourcing; human computation; skyline algorithms; worker models

ACM Reference Format:
Aris Anagnostopoulos, Adriano Fazzone, and Giacomo Vettraino. 2021. Skyline in Crowdsourcing with Imprecise Comparisons. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM ’21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482479

1 INTRODUCTION

In various applications there is the need to select an option among multiple options, each of which is characterized by a set of features. Consider, as an example, selecting a hotel, where competing factors include price, location, luxury, cleanliness, and so on. Other similar examples include the selection of a property to buy or a person to hire. Obviously, when there is an option that is superior in each dimension the choice is clear. Of course, typically in such cases, there are options that are preferable in some dimensions but less preferable in others, and it is up to the end user to examine the various options and select the best one according to her needs or preferences.

An important concept in such settings is that of the *skyline*, also known as the *Pareto frontier*. An object does *not* belong to the skyline if it is dominated by another object, that is, if there exists another object that is at least as good in every dimension and better in at least one dimension. The skyline is the set of options that are not dominated by other options. We can see an example in Figure 1(a). The concept of the skyline is an important one in various disciplines [6, 14] and there has been theoretical and applied work to compute it efficiently [5, 7, 12].



(a) An example of a skyline (points with ×). All the points in the grey area are dominated by at least one point belonging to the skyline.

(b) Three examples of approximate skylines (red–+, blue–× and green–∗). δ_1 and δ_2 are the error thresholds in the two dimensions.

Figure 1: Skyline and skyline in the threshold-error model.

Whereas the definition of the skyline is clear when each option under consideration has a known value, in many real-life scenarios the various options are not as clearly defined, but instead are characterized by the *crowd*: humans may express their personal preference between different options. Of course, humans err, thus it is often not clear how to select the skyline when objects are compared by humans. For this reason, there has been a line of work that studies the problem of finding the maximum elements (the 1-dimensional skyline) as well as the skyline under the presence of errors [1, 10, 11].

Two main models have been proposed to model human errors/inconsistencies when they compare elements: (1) the *probabilistic error model* and (2) the *threshold model*. To explain the

*Supported by the ERC Advanced Grant 788893 AMDROMA “Algorithmic and Mechanism Design Research in Online Markets”, the EC H2020RIA project “SoBigData++” (871042), and the MIUR PRIN project ALGADIMAR “Algorithms, Games, and Digital Markets.”

models, consider the one-dimensional setting. Consider elements x and y and assume that x is objectively better than y: for instance $\mathrm{score}(x) = 8$ and $\mathrm{score}(y) = 7$. Assume that we give pairs of elements to a human *worker* to compare them. In the probabilistic error model, the worker will return the correct answer with probability $1/2 + \epsilon$, for some $\epsilon \in (0, 1/2]$; in the various models proposed, ϵ may be fixed or dependent on the elements x and y [8, 16]. The errors performed by workers are independent, and querying multiple workers and performing majority voting can increase the accuracy exponentially in the number of votes. In the threshold model [1, 3], there exists a threshold δ such that if $\left|\mathrm{score}(x) - \mathrm{score}(y)\right| > \delta$ the workers respond correctly.[1] However, if $\left|\mathrm{score}(x) - \mathrm{score}(y)\right| \leq \delta$ then the responses are *arbitrary*; this means that even if the same query is repeated, the workers may consistently provide erroneous answers.

Of course, often there does not exist an objective winner. Then, depending on the context, the *correct* answer is typically assumed to be the one held by the majority of the workers, or by the workers who are more *expert* in the matter under consideration. Both the probabilistic-error and the threshold models have been studied in the crowdsourcing setting. Anagnostopoulos et al. [3] compared them in a crowdsourcing setting, and showed that for some types of queries the probabilistic-error model is an accurate model, whereas for other ones the threshold model is more appropriate.

Most of the modelling work for crowdsourcing has considered the one-dimensional case. Only in recent years have researchers started considering the multi-dimensional case (the skyline problem). In particular, Asudeh et al. [4] studied the skyline problem in a crowdsourcing scenario in presence of a perfect crowd: the workers make no errors in any comparison. On the other hand, the skyline problem under the probabilistic-error model has been studied in [10, 11] only theoretically. Yet, there has not been any study of the problem under the threshold model. In this paper we do exactly that: we study the skyline problem under the threshold error model both analytically and experimentally.

Contributions. To summarize our contributions: (1) We define the skyline problem under the threshold error model of Ajtai et al. [1] for crowdsourcing. (2) We give a set of algorithms of varying complexity and we provide theoretical guarantees on the number of required human comparisons. (3) We provide lower bounds on the number of comparisons required. (4) We compare the algorithms with various baselines, on synthetic and real-world data.

2 RELATED WORK

As we mentioned in the introduction, the computation of the skyline is a generalization of the problem of finding the maximum within a set of elements. With the involvement of humans in the computation process, there have been multiple works to model the effect of humans in the computation of the maximum element. Venetis et al. [16] consider some models from the psychometrics literature together with tournaments used to compute the maximum in crowdsourcing environments. Anagnostopoulos et al. [3] discuss about this and other approaches for computing the maximum, and we refer the reader to references therein.

[1]It can also be generalized to a worker responding correctly with a probability $p > 1/2$.

The more general problem of computing the skyline in crowdsourcing has attracted more attention lately. Asudeh et al. [4] have been the first to face the problem of crowdsourcing skyline with pairwise comparisons. Others, such as Lofi et al. [13], have focused on a similar scenario where the skyline is computed on incomplete data and the crowd provides estimation for the missing values. However, whereas there a numerical representation for the items' features is assumed, in [4] the authors assume only the existence of a strict partial order for the items according to the different features, because human preferences are very subtle, subjective, and hard to explain and comparing items is shown to be easier, faster and less error-prone for humans [15].

Groz and Milo [11] studied the skyline problem analytically under the probabilistic error model providing algorithms for the problem. Groz et al. [10] improved the theoretical results by providing algorithms that require fewer comparisons. Even though these papers are of high theoretical interest, there is no evaluation of the approaches in practice. To our knowledge, the only evaluation in practice of the skyline problem in a crowdsourcing setting is the aforementioned work of Asudeh et al. [4].

3 MODEL AND PROBLEM DEFINITION

In this section we start by describing the classic skyline problem, and then we introduce the version with the threshold (error) model. We also present some basic results that hold on the threshold model, required for our algorithms.

An assumption made by the model is the presence in each dimension of a (hidden) total order among all elements.

3.1 The Skyline Problem

The input of the *skyline* problem is a set of n d-dimensional points (we also refer to them as *elements*) $\mathcal{U} = \{u^1, u^2, \ldots, u^n\}$, where $u^i \in \mathbb{R}^d$ for $i \in [n] := \{1, 2, \ldots, n\}$.

Given two points $x, y \in \mathcal{U}$, we say that a point x *dominates* point y ($x \triangleright y$) if $x_i > y_i$ for all $i \in [d]$.[2] We say that point y *is dominated by* point x ($y \triangleleft x$) if x *dominates* point y. Finally, we say that a point x *ties with* point y ($x \sim y$) if there are at least two dimensions $i, j \in [d]$, with $i \neq j$, such that $x_i > y_i$ and $y_j > x_j$; that is, if none of the two points dominates the other one.

The set of points $\mathcal{S} \subseteq \mathcal{U}$ is called the *skyline* if it is the subset of $\mathcal{U}$ of maximum size that contains all the points that are not dominated by any point in $\mathcal{U}$ and only them.

Note that, the fact that none of the points in the skyline is dominated by any other point in the skyline, implies that every skyline point is in a *tie-with* relation with every other skyline point. Additionally, our assumption that in each dimension there is a total order between the points implies that the skyline is unique.

3.2 The Threshold-Error Model

In the crowdsourcing setting, we assume that we do not have the objective value of each element property, but we perform queries to the crowd to compare the various elements. When the value

[2]Note that in the introduction we used a slightly different definition. The two definitions are equivalent under our assumption that in each dimension there exists a total order among the elements.

difference is large then the crowd will typically provide an accurate/consistent answer. However, when the difference is small, then the crowd may err. The *threshold (comparison) model* of Ajtai et al. [1] defines a query model capturing this scenario, which we now describe for the d-dimensional setting.

According to the threshold model, a *query* (we also call it a *comparison*) $q_i(x, y)$ to a worker is defined by a pair of elements $(x, y) \in \mathcal{U} \times \mathcal{U}$ and a given dimension $i \in [d]$. The query asks whether $x_i > y_i$ or $x_i < y_i$. Because we assume that for each dimension we have a total order among the elements, ties on values are not allowed. The model also assumes that for each dimension i there exists a fixed threshold δ_i, representing the discernment ability of the worker [1]. If $|x_i - y_i| > \delta_i$, then the worker returns a correct answer. [3] However, if $|x_i - y_i| \leq \delta_i$, then the worker returns an *arbitrary* answer. Given that for $|x_i - y_i| \leq \delta_i$ the answer is arbitrary, this means that even if we repeat the question to multiple workers, we cannot obtain a higher confidence under this model; in other words, the two elements x and y are *indistinguishable* with respect to the ith dimension. This is in contrast to the other well studied theoretical error model for comparisons in crowdsourcing settings, the *probabilistic model* [11]. Anagnostopoulos et al. [3] showed that the threshold model is more accurate than the probabilistic model for some types of queries.

For two elements $x, y \in \mathcal{U}$ for which we perform the query $q_i(x, y)$ to a worker, we write that $x \prec_i y$ if we obtain as a result that x_i is smaller than y_i, and $x \succ_i y$ if we obtain as a result that y_i is smaller than x_i.

Whereas the skyline problem has been studied under the probabilistic-error model, it has not been studied under the threshold model, and this is the topic of this paper.

Two comments are necessary here. First, by rescaling, we can assume that each $\delta_i = \delta$. Second, even though the model assumes that there exist thresholds δ_i, these thresholds are unknown to the algorithms and it is not requested to assess their values for the skyline computation.

3.3 Basic Results on the Threshold Model

In this section we report some results from of Ajtai et al. [1], which we use in our proofs, regarding the problem of sorting and finding the maximum element under the threshold model. All these results refer to the one-dimensional setting.

Let δ be the threshold of the model. First, the authors show that under this model, it is impossible to return an element that is certain to be the maximum:

Theorem 3.1. *No deterministic max-finding algorithm has an error less than* 2δ.

Having an error of 2δ for the max-finding problem means that the element returned as maximum (m) is at most 2δ smaller than the real maximum element (m^*): $m^* - m \leq 2\delta$. The authors also present an algorithm called 2-MaxFind, which returns an element that is at at most 2δ of the maximum.

[3]The model can be generalized, with workers providing an incorrect answer when the difference is above δ, with a fixed probability, which may depend on the difference $|x_i - y_i|$, all the answers being independent, as in the probabilistic error model.

Theorem 3.2. *The 2-MaxFind algorithm is a deterministic algorithm, which performs at most* $2n^{3/2} = O(n^{3/2})$ *worker comparisons and guarantees an error of* 2δ.

They also give a lower bound on the number of comparisons required to find an approximate maximum element. Notice that the number of comparisons required is superlinear.

Theorem 3.3. *Every deterministic algorithm that guarantees to return an element that is at most* 2δ *smaller than the maximum requires* $\Omega(n^{4/3})$ *worker comparisons.*

Ajtai et al. also studied the problem of sorting. We say that the error of the output of a sorting algorithm is some value bounded by τ if for two elements x and y for which we have $x - y > \tau$ x is ranked higher than y in the output. The following corollary, defines the quality of the output of any sorting algorithm under the threshold model [1]:

Corollary 3.4. *According to Theorem 3.1, no deterministic sorting algorithm has an error less than* 2δ.

In addition, Ajtai et al. give a tight lower bound on the number of required comparisons and an optimal deterministic algorithm.

Theorem 3.5. *Every deterministic sorting algorithm with error at most* 2δ *requires* $\Omega(n^{3/2})$ *comparisons.*

Theorem 3.6. *The 2-Sort algorithm is a deterministic algorithm, which performs at most* $4n^{3/2} = \Theta(n^{3/2})$ *worker comparisons and guarantees an error of* 2δ.

A key component for both 2-MaxFind and 2-Sort algorithms is the Round-Robin tournament method, a method that Ajtai et al. defined in the following way: a Round-Robin tournament among the elements of a set $\mathcal{U}$ of size n consists in performing all $\binom{n}{2} = \Theta(n^2)$ pairwise comparisons. It is worth to notice that, under the threshold model, no algorithm benefits by performing more than $\binom{n}{2}$ worker comparisons: repeating the query does not provide a higher confidence.

The 2-Sort algorithm is of paramount importance for this work, for this reason Section 4.5 is dedicated to it.

3.4 Skyline Under the Threshold Model

Having described the threshold model for worker comparisons, we are ready to analyze the skyline problem under the threshold error model. In this model, comparisons are performed by workers, so the responses may be incorrect. In particular, they may not even satisfy the transitivity property. Moreover, given the lower bound of the previous section, we cannot guarantee to obtain the true skyline. Therefore, the goal is to return a set of elements that is as close as possible to the real skyline.

In view of Theorem 3.1, under the threshold model, no algorithm can guarantee to find the true skyline of a set of elements. Thus we need to relax the requirement for the output. We therefore define the concepts of *covering* and of *succinctness* for a solution.

Covering. Let $\mathcal{T}$ be the true skyline of a set of elements $\mathcal{U}$, and consider some solution $\mathcal{S} \subset \mathcal{U}$. Informally, we say that $\mathcal{S}$ is a *covering solution* if for every point in the skyline $\mathcal{T}$ there exists a point in $\mathcal{S}$ that is close to it. Now we define it formally.

We call solution $\mathcal{S}$ *c-covering* (with $c \geq 0$) if, for each element $t \in \mathcal{T}$, there exists an element $s \in \mathcal{S}$ such that for each dimension $i \in [d]$ we have that $s_i \geq t_i - c\delta_i$, namely: $\forall t \in \mathcal{T}, \exists s \in \mathcal{S} : \forall i \in [d], s_i \geq t_i - c\delta_i$.

By extension, we call an algorithm c-covering if it produces c-covering solutions. An example of two 1-covering outputs can be found in Figure 1(b) (blue−× and green−∗).

Succinctness. The covering property guarantees that the solution $\mathcal{S}$ does not miss points that are much better than the ones returned. However, this does not suffice: there may be points in $\mathcal{S}$ that are far from the optimal; in particular, a solution that returns the entire set $\mathcal{U}$ is a 0-covering solution!

Let us observe Figure 1(b). Whenever for two elements x and y the difference in the ith dimension is larger than δ_i, a worker response to a query is correct. Thus, informally, we can define a threshold (the red line in the figure, defined by the points that in each dimension i are δ_i smaller than the "skyline frontier") below and in the left of which we know that points cannot be in the skyline. This is the concept of succinctness. Informally, a solution is succinct if it contains only points that belong in the grey area, above and to the right of the red line. Next we define succinctness formally.

Let $\mathcal{T}$ be the true skyline of a set of elements $\mathcal{U}$, and consider some solution $\mathcal{S} \subset \mathcal{U}$. We call the solution $\mathcal{S}$ *c-succinct* (with $c \geq 0$) if for each pair $(s, t) \in \mathcal{S} \times \mathcal{T}$ there exists a dimension $i \in [d]$ for which we have $s_i \geq t_i - c\delta_i$, namely: $\forall (s, t) \in \mathcal{S} \times \mathcal{T}, \exists i \in [d] : s_i \geq t_i - c\delta_i$.

By extension, we call an algorithm c-succinct if it produces c-succinct solutions. An example of two 1-succinct outputs can be found in Figure 1(b) (red−+ and green−∗).

Notice from the examples in Figure 1(b), a solution can be covering and not succinct, succinct and not covering, or both covering and succinct (or neither). Of course, ideally we want solutions that are both covering and succinct, and our goal is to design algorithms that produce such solutions.

The following corollary follows from Theorem 3.1.[4]

Corollary 3.7. *Under the threshold error model no c-succinct deterministic skyline algorithm exists, with $c < 2$.*

Cost. Given that the expensive resource in the crowdsourcing setting is the set of comparisons performed by workers, we evaluate the efficiency of our algorithms with respect to the number of worker comparisons that they perform. We call the number of comparisons that an algorithm performs the *cost* of the algorithm.

Latency. In a very broad sense, Garcia-Molina at al. [9] define the *latency* of a crowdsourcing method as the time taken by the method to solve the problem. This general definition of *latency* embeds the *cost* of the method, even though, the *cost* of a crowdsourcing method is not the only characteristic associated with the time taken by a crowdsourcing method. Another important characteristic is the level of parallelization that a crowdsourcing method has: algorithms typically send the queries to the workers in *batches*. A *batch* is a set of queries that do not depend on each other and can be executed in parallel. We define the *latency* of an algorithm to be the minimum number of batches in which it can send to the queries to the workers. For instance, if an algorithm performs ℓ queries none of which requires the output of the other queries, the latency is 1; on the other extreme, if deciding what query to perform requires the result of the previous query, then the latency is ℓ.

[4]Proofs are omitted because of lack of space. They will appear in the full version of the paper.

4 ALGORITHMS

In this section we provide four algorithms for computing the skyline under the threshold error model. We present the algorithms in ascending order of their level of *covering* and then *succinctness*. For each algorithm we analyze its cost and latency together with its level of *covering* and *succinctness*. The last paragraph is dedicated entirely to the description of the changes we made to the 2-Sort algorithm (see Section 3.3) to reduce the cost and latency of the last two presented algorithms in the average-case scenario.

4.1 All-Play-All Algorithm

The most straightforward method to address the skyline problem consists in comparing all the n input elements with each other, discovering in this way all the *dominance* and *tie-with* relations among them. Having this complete information, for solving the problem it is enough to provide as output the set of input elements that are not dominated by any other input elements. We refer to this method as the All-Play-All algorithm.

The All-Play-All algorithm can be seen as the extension of the Round-Robin algorithm [1] (described in Section 3.3) to a multidimensional scenario.

The number of worker comparisons required by the All-Play-All algorithm are $d\binom{n}{2}$ in a worst-case scenario and $2\binom{n}{2}$ in a best-case scenario. Because under the threshold error model it is not possible to achieve higher confidence by repeating the question to multiple workers, $d\binom{n}{2}$ worker comparisons is the maximum number of comparisons that an algorithm can perform in our scenario (as already pointed in Section 3.2 and Section 3.3 for the Round-Robin algorithm). Despite its cost inefficiency, the All-Play-All algorithm has an optimal latency: a latency of 1.

The All-Play-All algorithm can provide in output an empty set as skyline even if a real skyline exists for the input set. Because of this unpleasant property, the All-Play-All algorithm has both an unbounded *covering* and *succinctness*.

4.2 Naive Algorithm

Here we present another simple algorithm for the skyline problem: the Naive algorithm. Naive is an iterative algorithm that maintains a set of elements as *dominated* ($\mathcal{D}$), initially equal to the empty set. At each iteration it selects an input element that has not already been discovered as dominated (i.e., not in $\mathcal{D}$), and compares it (in all dimensions using the workers) to all the other input elements not in $\mathcal{D}$. After each worker comparison, the algorithm updates the set of dominated elements $\mathcal{D}$. Once all iterations have been performed, Naive provides in output all the input elements that have been not discovered as dominated during its execution: $\mathcal{S} := \mathcal{U} \setminus \mathcal{D}$. We present the pseudocode in Algorithm 1.

Differently from the All-Play-All algorithm, the Naive algorithm cannot output an empty set and has a *succinctness* of 2.[5] But,

[5]Details for this and the other algorithms will appear in the full version of this paper.

Algorithm 1: Naive

```
Data: $\mathcal{U}$
Result: $\mathcal{S}$
1  $\mathcal{D} \longleftarrow \emptyset$ // set of dominated elements
2  for $x \in \mathcal{U}$ do
3      if $x \in \mathcal{D}$ then continue
4      for $y \in \mathcal{U} \setminus (\mathcal{D} \cup \{x\})$ do
           // performing worker comparisons
5          if $\bigwedge_{\ell \in [d]} x \succ_\ell y$ then $\mathcal{D} \longleftarrow \mathcal{D} \cup \{y\}$
6          if $\bigwedge_{\ell \in [d]} x \prec_\ell y$ then $\mathcal{D} \longleftarrow \mathcal{D} \cup \{x\}$
7  return $\mathcal{U} \setminus \mathcal{D}$
```

similarly to the All-Play-All algorithm, the Naive algorithm has an unbounded *covering* and a cost of $d\binom{n}{2}$ worker comparisons in the worst-case. Moreover, this algorithm has a latency of $n-1$ in the worst case.

4.3 Algorithm SortedDims

The third algorithm that we describe, we call it the SortedDims algorithm and it works as follows. It first sorts all the input elements according to each of the d dimensions independently using the 2-Sort algorithm, obtaining for each dimension a sorted list of all the input elements. By iterating over all input elements, it removes from the candidate set all the elements that are ranked, in all the sorted lists, below the element selected in the current iteration. The candidate set is initialized with all the input elements. Algorithm 2 depicts the pseudocode of the SortedDims algorithm.

Algorithm 2: SortedDims

```
Data: $\mathcal{U}$
Result: $\mathcal{S}$
1  foreach $\ell \in [d]$ do
2      Sort $\mathcal{U}$ along dimension $\ell$ using 2-Sort with workers, storing the result in $\mathcal{L}_\ell$
3  $\mathcal{S} \longleftarrow \mathcal{U}$
4  foreach $e \in \mathcal{U}$ do
5      $\mathcal{S} \longleftarrow \mathcal{S} \setminus \{x \in \mathcal{U} : \bigwedge_{\ell \in [d]} Rank_{\mathcal{L}_\ell}(x) < Rank_{\mathcal{L}_\ell}(e)\}$
6  return $\mathcal{S}$
```

Note that for this algorithm, we used human workers only at the beginning for the sorting phase (step 2) using the 2-Sort algorithm, where every pairwise comparison has been submitted as a crowdsourcing task. The sorting has to be performed for every dimension because the dimensions are completely independent and we do not assume any correlation among them. After having the sorted results in all the dimension, the dominance tests and the subsequent deletion of dominated items (step 5) is carried out without any further crowdsourcing involvement.

SortedDims sorts in each of the d dimensions independently using 2-Sort. By Theorem 3.6, the number of worker comparisons required for each of them is at most $4n^{3/2}$, thus the total number of worker comparisons is $O(dn^{3/2})$, in the worst case. Because all the 2-Sort algorithms can run in parallel, the latency of the SortedDims algorithm is equal to the maximum latency of all the 2-Sort algorithms. Differently from the the Naive algorithm, SortedDims has both a *succinctness* and a *covering* of 2.

4.4 Algorithm SingleDim

This algorithm can be seen as a particular version of the Naive algorithm, in which, instead of iterating without any particular order over the input set of elements, it iterates over them according to the order obtained by sorting the input set of elements $\mathcal{U}$ in relation to an arbitrarily chosen dimension.

The SingleDim algorithm maintains the skyline solution $\mathcal{S}$ and a set C of candidate elements to be part of $\mathcal{S}$. Initially, $\mathcal{S}$ is the empty set and C is the entire set $\mathcal{U}$. During its execution, the algorithm removes from C elements found to be dominated by elements in $\mathcal{S}$. The algorithm starts by creating a list containing all the input elements sorted according to an arbitrarily chosen dimension ℓ, by performing worker comparisons. Then it iterates over the sorted list from the highest to the lowest element. At each iteration, SingleDim adds to $\mathcal{S}$ the element $s \in C$ that is ranked highest in dimension ℓ, and removes it from C. By performing additional worker comparisons, the algorithm removes from C also all the elements that are dominated by element s. We present the pseudocode in Algorithm 3.

Algorithm 3: SingleDim

```
Data: $\mathcal{U}$
Result: $\mathcal{S}$
1  $\mathcal{S} \longleftarrow \emptyset$
2  Select any dimension $\ell \in [d]$.
3  Sort $\mathcal{U}$ along dimension $\ell$ using 2-Sort with workers, storing the result in $\mathcal{L}_\ell$
4  $C \longleftarrow \mathcal{U}$ // set containing candidate elements to be in the skyline
5  $index \longleftarrow |\mathcal{U}|$
6  while $index > 0$ do
7      $s \longleftarrow \mathcal{L}_\ell[index]$
8      if $s \in C$ then
9          $\mathcal{S} \longleftarrow \mathcal{S} \cup \{s\}$
10         $C \longleftarrow C \setminus \{s\}$
11         $C \longleftarrow C \setminus \{c \in C : \bigwedge_{i \in [d]} c \prec_i s\}$
           // we perform new worker comparisons even for $i = \ell$
12     $index \longleftarrow index - 1$
13 return $\mathcal{S}$
```

SingleDim performs worker queries in steps 3 and 11. By Theorem 3.6, step 3 requires $4n^{3/2}$ worker comparisons in the worst case. Regarding step 11, the number of comparisons is upper bounded by knd, where k is the output size. Therefore the total number of worker comparisons is $O(n^{3/2} + knd)$, where k is the size of the output skyline. So we see that in this case the cost is sensitive to the output size. The latency of the SingleDim algorithm is equal to the latency of the 2-Sort algorithm plus the size of the output skyline. Differently from the the previous algorithms, SingleDim has a *succinctness* of 2 and a *covering* of 1.

4.5 2-Sort Algorithm and QuickSort

As anticipated in Section 3.3, the 2-Sort algorithm is a fundamental method for both SingleDim and SortedDims. We modify the algorithm 2-Sort and, as a result, we improve the algorithms SortedDims and SingleDim, by reducing both cost and latency.

When we use this modified version of 2-Sort inside the SortedDims and SingleDim algorithms, we refer to them as SortedDims++ and SingleDim++, respectively.

In Algorithm 4, we report the pseudocode of the 2-Sort algorithm adapted to our scenario. 2-Sort is a recursive algorithm, which takes in input a set of elements—potentially lying on more than one dimension—and a dimension index, and provides in output a list containing all the input elements sorted with respect to the specified input dimension. The algorithm makes use of the workers only in step 3, where it performs the Round-Robin tournament to elect the pivot element, and in step 5, where the input set is partitioned in two subsets: $U_<$ containing all elements deemed smaller than the pivot by the workers according to the dimension ℓ, and $U_>$ containing all elements deemed greater than the pivot by the workers according to the dimension ℓ. Because of the usage of the crowd for two consecutive times in each invocation of the 2-Sort algorithm, the *latency* of this algorithm is equal to two times the height of its recursion tree.

The 2-Sort algorithm can be viewed as a generalization of the well-known QuickSort algorithm: by setting the internal parameter $p = 1$, 2-Sort is equivalent to the QuickSort algorithm. According to [1], the internal parameter p must be equal to $\lfloor\sqrt{2|\mathcal{U}|}\rfloor$ to achieve a number of worker comparisons at most equal to $4n^{\frac{3}{2}}$.

Given all this information, we modify the 2-Sort algorithm to keep a low number of worker comparisons in the average-case together with low latency, and still a number of worker comparisons of $\Theta(n^{\frac{3}{2}})$ in the worst-case scenario.

The first change concerns step 2 of the algorithm. Instead of selecting in an arbitrary way the subset $\mathcal{P}$ of size p, now the selection is done randomly. This allows us to exploit all the benefits that randomness gives to the average case. We also set $p = 1$.

At this point, the modified 2-Sort algorithm is equal to the QuickSort algorithm when the pivot choice is performed randomly. This sorting algorithm has good performance in practice, an $O(n \log n)$ number of comparisons in the best and expected cases, but still a $\binom{n}{2}$ number of comparisons in the worst-case. Under the threshold model, also this algorithm has an optimal error of 2δ.

To keep the number of comparisons in the worst-case equal to $\Theta(n^{\frac{3}{2}})$, we simply keep track of the number of unique pairwise worker comparisons during the execution of the modified 2-Sort with random subset selection of size $p = 1$. During the algorithm execution, in each recursive call, as soon as the number of already performed worker comparisons plus the maximum number of worker comparison than can be performed in the current call exceeds $4n^{\frac{3}{2}}$, we force the next recursive calls to the 2-Sort method to use a $p = \lfloor\sqrt{2|\mathcal{U}|}\rfloor$. Given that, the number of worker comparisons for the modified 2-Sort algorithm is not greater than $4n^{\frac{3}{2}} - 1 + 4n^{\frac{3}{2}} = O(n^{\frac{3}{2}})$.

As already reported in [1], under the threshold model it is mandatory to use 2-Sort-like algorithms, because in most other common sorting algorithm, errors resulting from imprecise comparisons might accumulate, causing very low output quality.

Algorithm 4: 2-Sort

```
  Data: $\mathcal{U}, \ell \in [d]$
  Result: $\mathcal{L}$ (sorted list of all elements of $\mathcal{U}$)
1 $p \longleftarrow \lfloor\sqrt{2|\mathcal{U}|}\rfloor$
2 Pick an arbitrary subset $\mathcal{P} \subset \mathcal{U}$ of size $p$
3 Perform a Round-Robin tournament with workers among the elements of $\mathcal{P}$
4 Let pivot be the element with the median number of wins in the Round-Robin
    tournament
5 Compare pivot with all elements in $\mathcal{U} \setminus \{pivot\}$ with workers
6 $U_< \longleftarrow \{u \in \mathcal{U} \setminus \{pivot\} : u <_\ell pivot\}$
7 $U_> \longleftarrow \{u \in \mathcal{U} \setminus \{pivot\} : u >_\ell pivot\}$
8 return 2-Sort$(U_<, \ell)$ + [pivot] + 2-Sort$(U_>, \ell)$
```

5 LOWER BOUNDS

In this section we provide two lower bounds on the number of required worker comparisons, which show that the room for theoretical improvement of our algorithms is small.

First we prove a lower bound for the skyline problem under the threshold model, which is independent of the skyline size. We prove the following theorem, whose proof is based on the *probabilistic method* [2] and is omitted for lack of space.

Theorem 5.1. *Assume that $d \geq 2\log_2 n$. Under the threshold error model, the number of worker comparisons required in the worst case to find the skyline of n, d-dimensional points is $\Omega(dn^{4/3})$.*

Next we present a second lower bound that does not consider any error model but that depends on the output skyline size. It is incomparable with the previous one; it can be smaller or larger. To derive this lower bound on the number of pairwise comparisons for our problem, we adapted the technique developed by Asudeh et al. [4] (Theorem 2) in the context in which transitivity among preference relations does not hold. We prove the following theorem.

Theorem 5.2. *Consider an instance to the problem of computing the skyline in the threshold model, and assume that the skyline output of an algorithm, which does not contain dominated points, has size k. Then the algorithm must perform at least $(n-k)\left[2(k-1)+d\right] + 2\binom{k}{2} = \Omega((n-k)(k+d)+k^2)$ worker comparisons.*

6 EXPERIMENTS

To evaluate and compare the performance of our algorithms, we performed experiments using different datasets typologies.

We studied the performance of our algorithms on random, semi-synthetic, and real-world datasets. For the first two, we simulated the behavior of a worker in the following way: when a worker is asked to rank a pair of elements according to a particular feature (dimension) whose value difference is below the threshold δ_i, each element is chosen as the answer with probability $1/2$. We chose the values of δ_i manually, such that workers provide inaccurate answers for a reasonable number of nearby elements: we chose δ_i equal to 10% and 5% of the maximum input value in the corresponding dimension, for the random and the semi-synthetic datasets, respectively. Of course, for the real-world dataset, there is no explicit value of δ; it is implied by the workers accuracy. Recall, in any case, that all our algorithms are agnostic to the value δ.

For the real-world dataset, we used the results from experiments performed using data collected by real workers from the well known crowdsourcing marketplace Amazon Mechanical Turk (AMT).[6] We also computed the lower bound on the number of worker comparisons according to Theorem 5.2.

6.1 Datasets

The random dataset allows us to create multiple input points and study the performance of our algorithms as the number of points increases in a controlled experiment. As a random dataset, we generated points with nonnegative coordinates, which are inside the hypersphere of radius 1 centered in the origin. We chose two,

[6]Data and code are available at: https://github.com/fazzadr/Skyline_in_Crowdsourcing_with_Imprecise_Comparisons.

three, and four dimensions, for input sets with a size that spans from 100 to 5000 points.

The semi-synthetic dataset we used in our simulations is the publicly available "California Housing Prices" (we refer to it as *Housing*) dataset.[7] The dataset provides information about 20,000 California city blocks. For conducting our simulations we selected the following, self explanatory, four features: "housing-median-age-within-a-block," "number-of-households-within-a-block," "median-income-for-household-within-a-block," and "median-house-value-for-household-within-a-block." We considered input sets with a size that spans from 100 to 5000 city-blocks.

The real-world dataset we used in our experiments is the publicly available dataset[8] created and used in [4]. This dataset collects 100 photos of the University of Texas at Arlington together with data obtained by real workers from the AMT crowdsourcing marketplace. For each of the 4950 pairs of photos, the dataset contains 5 different worker answers for each of the following three photo features (the dimensions in our setting): *color*, *sharpness*, and *landscape*. The possible answers that a worker was allowed to provide given a pair of photos and a feature (dimension) in input (for example, *sharpness*) where only three: "The first photo is preferred over the second one, for the selected feature," "The second photo is preferred over the first one, for the selected feature," and "The two photos are incomparable, for the selected feature."

6.2 Evaluation Metrics

To assess the quality of the algorithms' output in practice, we define the following two metrics that are coherent with the definition of *covering* and *succinctness* provided in Section 3.4:

$$Covering_{\mathcal{U}}(\mathcal{S}) = \max_{t \in \mathcal{T}} \min_{s \in \mathcal{S}} \min_{i \in [d]} \frac{|t_i - s_i|}{\delta_i}$$

$$Succintness_{\mathcal{U}}(\mathcal{S}) = \max_{s \in \mathcal{S}} \max_{t' \in \{t \in \mathcal{T} : \forall i \in [d], t_i \geq s_i\}} \min_{i \in [d]} \frac{t'_i - s_i}{\delta_i},$$

where $\mathcal{U}$ is the set of elements in input, $\mathcal{S} \subseteq \mathcal{U}$ is the output subset of elements provided by a skyline algorithm and $\mathcal{T} \subseteq \mathcal{U}$ is the real skyline of the input set $\mathcal{U}$.

Because these quantitative versions of *covering* and *succinctness* are applicable only when an explicit representation of the value of each feature is present for every input point, we cannot rely on these metrics for assessing the quality of the output in the experiments with real crowdsourcing workers. For this reason, we adapt some standard metrics developed for assessing the quality of ranking systems to our scenario. In particular, we adapt the standard *normalized-discounted-cumulative-gain@k* evaluation metric [17], used for evaluating the quality of a ranking method, to the evaluation of the quality of the output of a skyline algorithm. The adaptation is that we assign a relevance of 1 to every element in $\mathcal{U}$ and that we consider as the rank of an element the *rank of the skyline* the element belongs to: Given a set of elements $\mathcal{U}$, the skyline of rank 1 ($skyline_1(\mathcal{U})$) is simply defined as the (true) skyline of the set $\mathcal{U}$. The skyline of rank 2 for a set of elements $\mathcal{U}$, is defined as the skyline computed on the set of elements $\mathcal{U} \setminus skyline_1(\mathcal{U})$.

[7]https://www.kaggle.com/camnugent/california-housing-prices
[8]https://github.com/idirlab/crowdpareto

More generally, the skyline of rank $k > 1$ for a set of elements $\mathcal{U}$, is defined as the skyline computed on the set $\mathcal{U} \setminus \bigcup_{r=1}^{k-1} skyline_r(\mathcal{U})$.

Given the definition of $skyline_r(\mathcal{U})$ we define as the *skyline-rank* of an element e the rank of the skyline that contains it (i.e., the value r such that $e \in skyline_r(\mathcal{U})$), and we denote it by $sk\text{-}rank_{\mathcal{U}}(e)$.

For evaluating the *discounted-cumulative-gain@k* of a subset $\mathcal{S} \subset \mathcal{U}$, we consider the elements sorted according to their skyline ranks. Let $sorted_{\mathcal{U}}(\mathcal{S})$ be the list containing the elements of $\mathcal{S} \subset \mathcal{U}$, sorted according to their skyline rank.

Considering all these changes, we define the *adapted-discounted-cumulative-gain@k* of a subset $\mathcal{S}$ of the input set of elements $\mathcal{U}$ in the following way:

$$ADCG_k^{\mathcal{U}}(\mathcal{S}) = \sum_{i=1}^{k} \frac{1}{\log_2(sk\text{-}rank_{\mathcal{U}}(sorted_{\mathcal{U}}(\mathcal{S})[i]) + 1)}$$

Accordingly with its standard definition, we define the *ideal-adapted-discounted-cumulative-gain@k* ($IADCG_k$) as the highest *adapted-discounted-cumulative-gain@k* score that a subset of $\mathcal{U}$ of size k can obtain.

Finally, the *adapted-normalized-discounted-cumulative-gain@k* is defined as follows:

$$AnDCG_k^{\mathcal{U}}(\mathcal{S}) = \frac{ADCG_k^{\mathcal{U}}(\mathcal{S})}{IADCG_k^{\mathcal{U}}}.$$

Finally, for evaluating the *cost* and *latency* of our methods, as defined in Section 3.4, we consider the number of distinct pairwise comparisons performed by workers (*cost*) and the maximum number of times our methods use the workers not in a parallel fashion during their execution (*latency*).

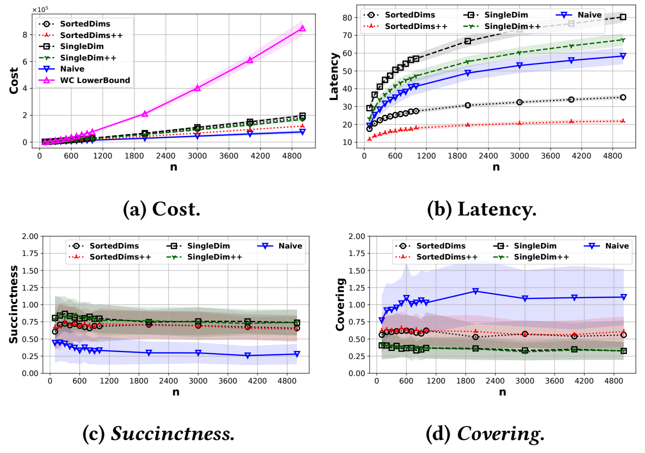
6.3 Experimental Results

6.3.1 Experiments on the Random Dataset. Figure 2 shows the results from our experiments on the random dataset described in Section 6.1 on two dimensions. Figure 2(a) shows that the *worst-case* lower bound on the number of comparisons given by the theory turns out to be much higher than the values that we observe on average in practice. This can be explained by the fact that lower bound is a function of the output size of the true skyline but the algorithms return a smaller number of points: For $n = 5000$, the average size of the real skyline is of 85.61 points against an average size of 46.39 for SingleDim, 47.11 for SingleDim++, 21.63 for SortedDims, 22.15 for SortedDims++, and 7.29 for the Naive method. All the proposed methods seem to have the same performance in terms of cost, with the Naive method being the cheapest one. SingleDim is slightly more expensive than SortedDims, and so is SingleDim++ compared to SortedDims++. We can also observe that the *plus-plus* algorithms are cheaper than their original counterparts, making SortedDims++ and SingleDim++ the cheapest ones with theoretical guarantees, showing the effectiveness of the optimizations performed on their original versions.

Regarding the latency, Figure 2(b) shows that the experimental findings are completely in agreement with the theoretical analysis of the methods: SortedDims (and SortedDims++) has a lower latency than SingleDim (SingleDim++), and the *plus-plus* versions have a lower latency than their original versions. Despite the fact that the Naive method has a latency of $n - 1$ in the worst-case, Figure 2(b) shows that Naive has a little lower latency than SingleDim++ for this dataset.

According to Figures 2(c) and 2(d), both SortedDims and SingleDim have the same value of *succinctness* and *covering* of their *plus-plus* versions, showing that the optimizations performed on their original versions have not affected the provided level of *succinctness* and *covering* not even in practice. All algorithms have almost the same level of *succinctness*, with the exception of the Naive method. Regarding the *covering*, SingleDim and SingleDim++ obtain the best score: this is in agreement with the fact that SingleDim and SingleDim++ have a *covering* of 1, instead SortedDims and SortedDims++ have a *covering* of 2. Despite the fact that all methods have a *succinctness* of 2, Naive obtains the best level of *succinctness* among them: this can be attributed to the small size of its output, relatively to the other methods. The small size of the output skyline, together with an unbounded *covering*, explains the fact that Naive has the worst level of *covering* in the conducted experiments.

Overall, Figure 2 shows that SortedDims++ achieves a good tradeoff among all the considered methods: it always achieves at least the second-best performance for the considered evaluation metrics.



(a) Cost. **(b) Latency.** **(c) *Succinctness.*** **(d) *Covering.***

Figure 2: Average and standard deviation (over 100 repetitions for each input set size) of cost, latency, *succinctness*, and *covering* provided by our algorithms on the *random dataset* for various input sizes (n) in two dimensions.

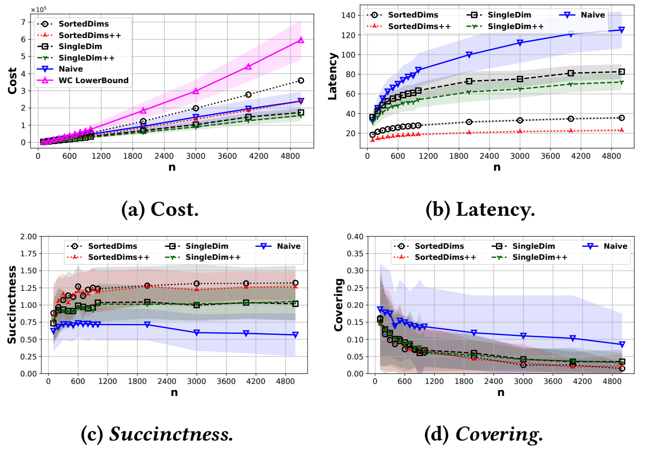
6.3.2 Experiments on the Semi-Synthetic Dataset. Figure 3 show the results from our experiments on the semi-synthetic *Housing* dataset on four dimensions. Figure 3(a) compares the algorithms in terms of cost and shows that also in this dataset the cost given by the worst-case theoretical lower bound is higher than what we observe in practice, again explained by the large dimension of the ground-truth-skyline: for $n = 5000$, the average size of the real skyline is of 58.96 points against an average size of 49.25 for SingleDim, 51.11 for SingleDim++, 71.14 for SortedDims, 70.1 for SortedDims++, and 27.42 for the Naive method.

Differently from the *random* dataset, Figure 3(a) shows that the cheapest method is SingleDim++: this fact is in concordance with the theoretical cost analysis of the method and emphasized by the presence of four dimensions instead of two. The fact that SingleDim++ and SingleDim are cheaper than SortedDims++ and SortedDims, respectively, can explained by the fact that the points are in four dimension and the cost of the first two methods is less dependent to the number of dimensions than the second ones. Moreover, coherently with the experiments performed on the *random* dataset, the *plus-plus* algorithms are again cheaper than their original counterparts. In contrast with the experiments performed on the *random* dataset, Naive is no more the cheapest one, reporting a cost similar to the one for SortedDims++.

Regarding the latency, Figure 3(b) shows that the experimental findings are completely in agreement with the theoretical analysis of the methods: SortedDims (SortedDims++) has a lower latency than SingleDim (SingleDim++), the *plus-plus* versions have a lower latency than their original versions, and the Naive method gave the highest level of latency.

Similarly to the experiments performed on the *random* dataset, Figures 3(c) and 3(d) show that both SortedDims and SingleDim obtain the same value of *succinctness* and *covering* with their *plus-plus* versions. Regarding the *covering*, Figure 3(d) shows that all methods achieve the same good level of *covering*, with Naive performing slightly worse than the other methods. Figure 3(c) shows that also for the *Housing* dataset, Naive achieves the best level of *succinctness*, followed by SingleDim and SingleDim++, and then by SortedDims and SortedDims++.

Overall, Figure 3 shows that SingleDim++ is the preferable algorithm among all the considered methods: it essentially achieves the best performance in terms of *covering* and cost, while keeping a latency that is second only to the methods based entirely on sorting (SortedDims and SortedDims++) and the second-best level of *succinctness*.



(a) Cost. **(b) Latency.** **(c) *Succinctness.*** **(d) *Covering.***

Figure 3: Average and standard deviation (over 100 repetitions for each input set size) of cost, latency, *succinctness*, and *covering* provided by our algorithms on *semi-synthetic “Housing” datasets* for various input sizes (n) in four dimensions.

6.3.3 Experiments Using a Real Crowdsourcing Marketplace. The last set of experiments regards the real-world crowdsourcing setting and the findings are summarized in Figure 4. For each subfigure, the box plots represent the distribution of values of the specified metric obtained from 1000 different algorithms' executions. Each time that one of our algorithms asks for a comparison to the workers, a random answer among all five collected answers from the crowd is selected as the worker's answer. We also never provide to the crowd the same comparison twice. Following Asudeh at al. [4], we assumed as ground-truth the skyline obtained considering as correct answer the one obtained applying majority voting to the five answers provided by the dataset. For this dataset, we do not report *covering* and *succinctness* evaluations because we do not have an explicit representation of the elements in the space.

Recalling Section 6.1, for this experiment the input set of elements has size 100 and consists of photos of the University of Texas at Arlington. The number of dimensions is 3, corresponding to three photo features: *color*, *sharpness*, and *landscape*. Given the relative small size of the input set of elements, here we also tested the performance of the All-Play-All method.

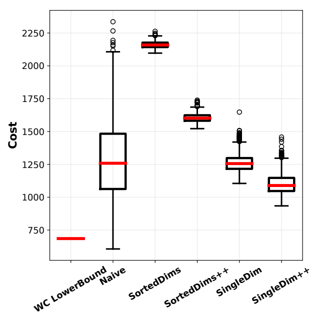
According to Figure 4(a), differently from the previous two experiments, the number of worker comparisons provided by the output-sensitive worst-case lower bound (Theorem 5.2) is smaller than the one that our algorithms performed (with the exception of some executions of the Naive method). Regarding the cost (Figure 4(a)) and latency (Figure 4(b)), the SortedDims++ and SingleDim++ methods have better performance than the original versions (SortedDims and SingleDim), while keeping the same level of quality for the output skyline (Figure 4(c) and Figure 4(d)). This result shows the effectiveness of the optimizations performed on their original versions, for the real-world dataset as well.

Despite the fact that the All-Play-All method has an optimal latency of 1 (Figure 4(b)), the method performed a median of 12, 770 worker comparisons on this dataset: this number is completely out of scale and this is the reason behind the absence of this method in Figure 4(a). Moreover, because of the presence of empty skylines in output, the All-Play-All algorithm has zero score for both metrics shown in Figures 4(c) and 4(d).

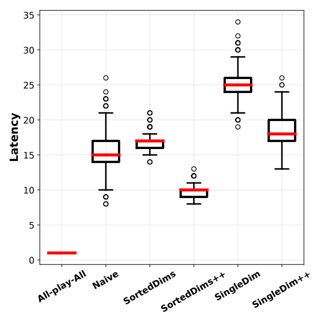
The Naive method has a median cost essentially equal to the median cost of the SingleDim method, but with a much wider distribution. Regarding the median latency value, the Naive method is only worse than SortedDims++ and, of course, than All-Play-All (which has the optimal latency of one). According to Figures 4(c) and 4(d), the quality of the skylines in output by the Naive method have a level of quality not higher than the ones provided by the methods with guaranteed *succinctness* and *covering*, despite the fact that it never provided an empty skyline in output: this confirms the observations of Section 4.2.

For a ground truth skyline size of 3, the median of the distributions of the output skyline sizes provided by all the tested methods are the following: 4 for All-Play-All, 5 for Naive, 9 for both SortedDims and SortedDims++, and 10 for both SingleDim and SingleDim++.

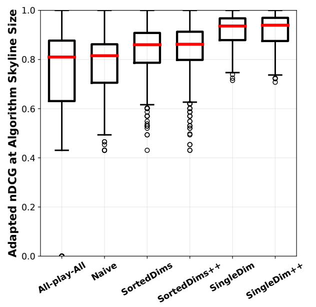
Overall, according to the experimental results depicted in Figure 4, the SingleDim++ method provided in output the skylines with the highest level of quality at the cheapest cost, but at a higher latency than the other methods (with the exception of the SingleDim method).



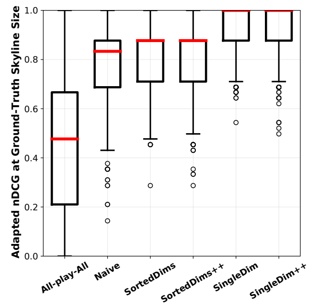
(a) Cost.



(b) Latency.



(c) Adapted nDCG computed at output size.



(d) Adapted nDCG computed at Ground-Truth size.

Figure 4: Distributions of cost, latency, and adapted normalized discounted cumulative gain provided by our algorithms on the *real-world dataset*; results calculated over 1000 independent executions on 100 elements lying on 3 dimensions.

7 CONCLUSION AND FUTURE WORK

In this paper we continued the study of the problem of skyline computation in the context of crowdsourcing. Following prior work on crowdsourcing, we considered the threshold error model for modeling the human comparisons. We provided algorithms and close lower bounds. We then evaluated the performance or our approaches on a variety of synthetic and real data.

There are some problems that are open. First, there is a small gap between the lower bounds and our algorithms, which we would like to be closed. Note that for the output-independent bound, this would require some significant theoretical improvement of the work in [1], which is a component of our algorithms.

On a broader scope, in this work we followed a large tradition in the analysis of algorithms in crowdsourcing, and we used the comparison of two elements as a primitive. This allows for a clean formulation of the problem and a theoretical analysis. Often, however, humans use different primitives. For instance, they may assign some subjective score to some of the elements. How such an approach can be used to model finding the skyline is an interesting modeling and algorithmic problem.

REFERENCES

[1] Ajtai, M., Feldman, V., Hassidim, A., and Nelson, J. Sorting and selection with imprecise comparisons. *ACM Trans. Algorithms 12*, 2 (Nov. 2015).

[2] Alon, N., and Spencer, J. H. *The Probabilistic Method*, 4th ed. Wiley Publishing, 2016.

[3] Anagnostopoulos, A., Becchetti, L., Fazzone, A., Mele, I., and Riondato, M. The importance of being expert: Efficient max-finding in crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD '15, Association for Computing Machinery, p. 983–998.

[4] Asudeh, A., Zhang, G., Hassan, N., Li, C., and 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* (New York, NY, USA, 2015), CIKM '15, ACM, pp. 753–762.

[5] Borzsony, S., Kossmann, D., and Stocker, K. The skyline operator. In *Proceedings 17th International Conference on Data Engineering* (April 2001), IEEE Comput. Soc, pp. 421–430.

[6] Chomicki, J. Preference formulas in relational queries. *ACM Trans. Database Syst. 28*, 4 (Dec. 2003), 427–466.

[7] Das Sarma, A., Lall, A., Nanongkai, D., and Xu, J. Randomized multi-pass streaming skyline algorithms. *Proc. VLDB Endow. 2*, 1 (Aug. 2009), 85–96.

[8] Feige, U., Raghavan, P., Peleg, D., and Upfal, E. Computing with noisy information. *SIAM Journal on Computing 23*, 5 (1994), 1001–1018.

[9] Garcia-Molina, H., Joglekar, M., Marcus, A., Parameswaran, A., and Verroios, V. Challenges in data crowdsourcing. *IEEE Transactions on Knowledge & Data Engineering 28*, 04 (apr 2016), 901–911.

[10] Groz, B., Mallmann-Trenn, F., Mathieu, C., and Verdugo, V. Skyline computation with noisy comparisons. In *Combinatorial Algorithms - 31st International Workshop, IWOCA 2020, Bordeaux, France, June 8-10, 2020, Proceedings* (2020), L. Gasieniec, R. Klasing, and T. Radzik, Eds., vol. 12126 of *Lecture Notes in Computer Science*, Springer, pp. 289–303.

[11] Groz, B., and Milo, T. Skyline queries with noisy comparisons. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (New York, NY, USA, 2015), PODS '15, ACM, pp. 185–198.

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

[13] Lofi, C., El Maarry, K., and Balke, W.-T. Skyline queries in crowd-enabled databases. In *Proceedings of the 16th International Conference on Extending Database Technology* (New York, NY, USA, 2013), EDBT '13, Association for Computing Machinery, p. 465–476.

[14] Sacharidis, D., Papadopoulos, S., and Papadias, D. Topologically sorted skylines for partially ordered domains. In *2009 IEEE 25th International Conference on Data Engineering* (2009), pp. 1072–1083.

[15] Thurstone, L. L. A law of comparative judgment. *Psychological Review 34*, 4 (1927), 273–286.

[16] Venetis, P., Garcia-Molina, H., Huang, K., and Polyzotis, N. Max algorithms in crowdsourcing environments. In *Proceedings of the 21st International Conference on World Wide Web* (New York, NY, USA, 2012), WWW '12, Association for Computing Machinery, p. 989–998.

[17] Wang, Y., Wang, L., Li, Y., He, D., and Liu, T.-Y. A theoretical analysis of ndcg type ranking measures. In *Proceedings of the 26th Annual Conference on Learning Theory* (Princeton, NJ, USA, 12–14 Jun 2013), S. Shalev-Shwartz and I. Steinwart, Eds., vol. 30 of *Proceedings of Machine Learning Research*, PMLR, pp. 25–54.