data

MDPI

Article

Computing Non-Dominated Flexible Skylines in Vertically Distributed Datasets with No Random Access

Davide Martinenghi

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo 32, 20133 Milan, Italy; davide.martinenghi@polimi.it

Abstract: In today's data-driven world, algorithms operating with vertically distributed datasets are crucial due to the increasing prevalence of large-scale, decentralized data storage. These algorithms process data locally, thereby reducing data transfer and exposure to breaches, while at the same time improving scalability thanks to data distribution across multiple sources. Top-k queries are a key tool in vertically distributed scenarios and are widely applied in critical applications involving sensitive data. Classical top-k algorithms typically resort to *sorted access* to sequentially scan the dataset and to *random access* to retrieve a tuple by its id. However, the latter kind of access is sometimes too costly to be feasible, and algorithms need to be designed for the so-called "no random access" (NRA) scenario. The latest efforts in this direction do not cover the recent advances in ranking queries, which propose hybridizations of top-k queries (which are preference-aware and control the output size) and skyline queries (which are preference-agnostic and have uncontrolled output size). The *non-dominated flexible skyline* (ND) is one such proposal, which tries to obtain the best of top-k and skyline queries. We introduce an algorithm for computing ND in the NRA scenario, prove its correctness and optimality within its class, and provide an experimental evaluation covering a wide range of cases, with both synthetic and real datasets.

Keywords: skyline; flexible skyline; top-k query; random access

Academic Editor: Han Woo Park

Received: 17 March 2025
Revised: 1 May 2025
Accepted: 14 May 2025
Published: 15 May 2025

Citation: Martinenghi, D. Computing Non-Dominated Flexible Skylines in Vertically Distributed Datasets with No Random Access. *Data* **2025**, *10*, 76. https://doi.org/10.3390/data10050076

1. Introduction

In today's data-centric world, algorithms designed for vertically distributed datasets are vital due to the growing trend of large-scale, decentralized data storage. These algorithms enhance data privacy by processing information locally, thereby reducing the need for data transfers and minimizing the risk of breaches. They also improve scalability by efficiently managing large volumes of data distributed across various locations without relying on centralized access. Top-k queries (the fundamental tool to tackle multi-objective optimization by transforming the problem into a single-objective problem through a *scoring function*) have been studied extensively under this lens, and they are particularly suitable in applications involving healthcare, finance, and the IoT, where data are often sensitive and distributed across various sources. Classical top-k algorithms, such as [1], are based on the availability of two kinds of access to sources: *sorted access*, i.e., a sequential scan in the internal sort order, one tuple at a time, of the dataset; *random access*, which provides all the information available at a data source for a tuple whose ID is known. However, in scenarios where data retrieval costs are high, data are streamed in real-time, or data are from external sources that only offer sorted access, random access may become impractical or impossible due to latency issues or data access constraints.

The "no random access" (NRA) scenario is practically relevant in all those scenarios in which efficient random access to tuples is not supported. Notable examples of real-world applications for these scenarios include NoSQL Databases with Scan-Based Querying [2,3], such as key–value stores and document stores; Web Scraping and APIs with Paginated Access [4], which do not allow direct access to an arbitrary record without sequentially retrieving previous pages, e.g., in offset and limit queries; Streaming Databases and Data Pipelines [5], in which data flow in a sequential manner, and there is no direct way to jump to an arbitrary tuple; and Federated Databases [6], where some sources may support only limited querying capabilities.

Fortunately, a long tradition of algorithms designed for the NRA scenario exists for classical top-k queries [7]. Yet, these do not cover the recent advances that have proposed hybridizations of top-k queries and skyline queries [8] (the other common tool for multi-criteria analysis, based on the notion of *dominance*), which try to get the best of both worlds. Although both share the overall goal to identify the best objects in a dataset, the way they work is, indeed, totally different: top-k queries are preference-aware and control the output size, while skylines are preference-agnostic and have uncontrolled output size. *Flexible skylines* [9,10] are a popular attempt at reconciling top-k queries and skylines under a unifying perspective, and, in particular, the non-dominated (ND) flexible skyline is a preference-aware generalization of skylines, which admits efficient implementations. This paper refers to a further generalization, which introduces the k parameter into skylines (called k-skybands [11] and consisting of all tuples dominated by fewer than k tuples), leading to the notion of non-k-dominated flexible skyline (ND_k).

Example 1 (Running example)**.** *To exemplify the notion of a flexible skyline, consider the case of a job seeker wanting to enhance their CV with certifications to be shown during an interview. However, a job seeker may have a limited budget that certification costs should not exceed, and they may be willing to spend only a limited amount of time on the preparation, so that the difficulty of a certification test is also a factor to be taken into account. The dataset in Figure 1a shows two ranked lists of certification tests (named* a, b, ..., g*) sorted by their cost (in list r_1) and by their difficulty (in list r_2), both expressed on a 10-point scale (lower values are preferable). The only options that are not dominated by other options in this dataset are* a, b, c, *and* e*, because no alternative is better than any of them on both criteria; this is the skyline of the dataset. In particular, each of these options is top for at least one monotone scoring function; for instance,* c *is the top option if we take the average of the two criteria (4), i.e., if we use the scoring function $s_c(x, y) = 0.5x + 0.5y$, where x and y are, respectively, the numeric attributes from r_1 and r_2. Note that the scoring function can be thought of as a strategy for exploring the dataset. In particular, with a linear scoring function like s_c, its weights determine the slope of a straight line that passes through the origin and moves away from it in the first quadrant while keeping the same slope: the first tuple that is encountered in this way is the best ranked tuple. With s_c, the slope is $-45°$ and* c *is the first tuple that such a straight line would meet[1]. A different scoring function, such as $s_e(x, y) = y$, would have a horizontal slope, and* e*, which indeed has the lowest difficulty value, would be met first.*

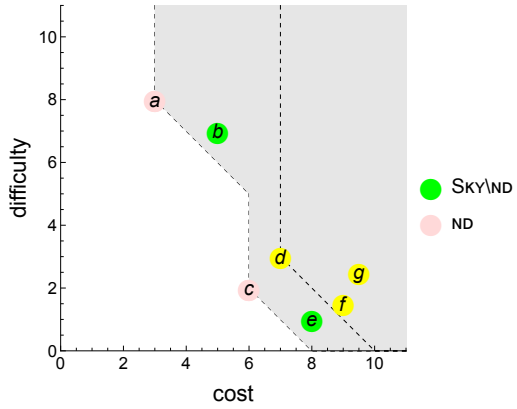
Suppose, now, we also consider a preference that states the following: while we still want to linearly combine the two criteria, the cost is more important, i.e., we refer to the set $\mathcal{F}$ of scoring functions of the form $s(x, y) = w_1x + w_2y$, where $w_1 > w_2$ (in terms of straight lines, this means disallowing slopes between $-45°$ and $0°$). Under this assumption, b *and* e *are no longer ideal, as no scoring function of the stated form will ever rank them as the top solution; the remaining options,* a *and* c*, are the non-dominated flexible skyline* ND *with respect to the set $\mathcal{F}$. The gray area shown in Figure 1b represents the region of points that can never become top-1 in this dataset for any of the functions in $\mathcal{F}$; in particular,* b *can never beat* a *and* e *can never beat* c *with these functions.*

The 2-skyband of the dataset includes all the tuples that are dominated by fewer than 2 tuples (i.e., 0 or 1) and consists of `d` *and* `f`*, in addition to all the tuples in the skyline (*`a`*,* `b`*,* `c`*,* `e`*). This set contains all tuples that may be in the result set of any top-2 query using a monotone scoring function. For instance, with* $s_c(x, y) = 0.5x + 0.5y$*, the top-2 results are* `c` *and* `e` *(indeed,* `e` *is the second tuple met by traversing the dataset with a slope of* $-45°$*). However, if we consider the set* $\mathcal{F}$*,* `f` *is now, in a sense, also dominated by* `d`*, because no scoring function in* $\mathcal{F}$ *may rank it better than* `d`*, as shown with the dashed line at* $-45°$ *traversing* `d`*. The tuples in the 2-skyband, except for* `f`*, are the so-called non-k-dominated flexible skyline* ND_k *with respect to* $\mathcal{F}$*, with* $k = 2$*.*

	r_1
`a`	3.0
`b`	5.0
`c`	6.0
`d`	7.0
`e`	8.0
`f`	9.0
`g`	9.5

	r_2
`e`	1.0
`f`	1.5
`c`	2.0
`g`	2.5
`d`	3.0
`b`	7.0
`a`	8.0

(**a**) Dataset in tabular form.



(**b**) 2D depiction of the certification tests.

Figure 1. A set of certification tests ranked by difficulty and cost.

In this paper, after reconsidering the standard `NRA` approaches, we introduce a novel algorithm for computing the ND flexible skyline in the `NRA` scenario, prove its correctness and instance optimality (a very strong form of optimality in an I/O sense), and provide an experimental evaluation covering a wide range of datasets, both synthetic and real. The gist of the algorithm is to exploit the main idea behind classical `NRA` algorithms in a conservative manner so as to allow the application of the notion of flexible dominance, which requires considering a (possibly infinite) set of scoring functions at the same time instead of just one. This is done by moving from the space of scores and bounds, used in the classical approaches, to that of representative points, which capture the best and worst possible circumstances when scores are unknown and there are multiple scoring functions. Suitable bookkeeping implemented with proper counters allows for separating a *growing phase*, during which sorted accesses are made to buffer the tuples potentially in the result, from a *shrinking phase* that consists of further sorted accesses to acquiring sufficient information for discarding irrelevant tuples and finalizing the result.

Summary of contributions. The main contributions of this paper are as follows:

- We consider for the first time the problem of computing the flexible skyline on a vertically distributed dataset when random access is not available (`NRA` scenario).
- We address the problem by proposing an algorithm that is proved correct and provides strong optimality guarantees on the execution cost.
- We run an extensive set of experiments validating our solution, covering several real and synthetic datasets, and propose two baselines obtained through non-trivial adaptations of the algorithms existing in the literature. Our solution largely outperforms the baselines and proves to be effective, especially in those scenarios (like uniformly distributed datasets) in which early termination can be achieved even in the absence of random access.

The paper is organized as follows. Section 2 introduces the main notation and basic notions used for building our results. Section 3 describes the algorithmic pattern for

computing ND_k and proves its correctness and optimality. Section 4 tests our solution through several experiments and compares it to baselines. Section 5 offers a comprehensive discussion of related work. Finally, Section 6 proposes concluding remarks and outlines possible future work.

2. Preliminaries

We refer to datasets with numeric attributes (namely, and without loss of generality, the non-negative real numbers $\mathbb{R}^+$). A schema S is a set of attributes $\{A_1, \ldots, A_d\}$, and a tuple $t = \langle v_1, \ldots, v_d \rangle$ over S is a function associating each attribute $A_i \in S$ with a value v_i, also denoted $t[A_i]$, in $\mathbb{R}^+$; a relation over S is a set of tuples over S.

A *scoring function* f over $S = \{A_1, \ldots, A_d\}$ applies to the attribute values of a tuple t over S and associates t with a numeric score $f(t[A_1], \ldots, t[A_d]) \in \mathbb{R}^+$, also indicated $f(t)$; f is *monotone* if, for all tuples t and s over S, we have $(\forall A \in S \,.\, t[A] \leq s[A]) \rightarrow f(t) \leq f(s)$. The function $s_c(x, y) = 0.5x + 0.5y$ used in Example 1 is a monotone scoring function.

We shall consider attributes such as "cost", and thus prefer smaller values over higher ones; this is, of course, an arbitrary convention that could be changed and does not affect generality. Similarly, we also set our preference for lower scores (again, without loss of generality).

The *rank* $\mathsf{rank}(t; r, f)$ of a tuple $t \in r$ according to the scoring function f is defined as $\mathsf{rank}(t; r, f) = 1 + |\{s \in r \mid f(s) < f(t)\}|$. In Example 1, $\mathsf{rank}(\mathtt{c}; r, s_c) = 1$, where the relation $r = r_1 \bowtie r_2$ refers to the natural join of relations r_1 and r_2; similarly, $\mathsf{rank}(\mathtt{e}; r, s_c) = 2$.

Definition 1. *A* top-k query *takes* (i) *a relation r over a schema S,* (ii) *a scoring function f over S that totally orders r,* (iii) *a positive integer k, and returns the set* $\textsc{Top}_k(r; f)$ *of top-k ranked tuples in r according to f, i.e.,* $\textsc{Top}_k(r; f) = \{t \mid t \in r \wedge \mathsf{rank}(t; r, f) \leq k\}$.

With the tuples of Example 1, we have $\textsc{Top}_2(r; s_c) = \{\mathtt{c}, \mathtt{e}\}$.

We assumed that f orders r totally so that no ties occur and the set returned by a top-k query is univocally defined. When $k = 1$, the definition can be expanded as follows:

$$\textsc{Top}_1(r; f) = \{t \mid t \in r \wedge \forall s \in r \,.\, s \neq t \implies f(s) > f(t)\}. \tag{1}$$

A *skyline* query [8] takes a relation r as input and returns the set of tuples in r that are dominated by no other tuples in r. Dominance is defined as follows.

Definition 2. *Let t and s be tuples over a schema S; t* dominates *s, denoted $t \prec s$, if, for every attribute $A \in S$, $t[A] \leq s[A]$ holds and there exists an attribute $A' \in S$ such that $t[A'] < s[A']$ holds. The* skyline $\textsc{Sky}(r)$ *of a relation r over S is the set $\{t \in r \mid \nexists s \in r \,.\, s \prec t\}$.*

As mentioned in the previous section, in our running example, $\textsc{Sky}(r) = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{e}\}$.

Equivalently, the skyline can be identified as the set of tuples that are top-1 results for at least one monotone scoring function:

$$\textsc{Sky}(r) = \{t \mid t \in r \wedge \exists f \in \mathtt{MF} \,.\, \forall s \in r \,.\, s \neq t \implies f(s) > f(t)\}, \tag{2}$$

where $\mathtt{MF}$ indicates the set of all monotone functions. For instance, in the running example, tuple $\mathtt{c}$ is the top-1 result for scoring function s_c, while $\mathtt{e}$ is top-1 for function s_e. Finding a function that makes $\mathtt{b}$ top-1 is less evident, but one is guaranteed to exist (e.g., $s_b(x, y) = x^2 + 0.1y^2$).

Inspired by the similarity between the expanded definition of top-1 set given in Equation (1) and the alternative definition of skyline in Equation (2), Ref. [9] introduced the

notion of *flexible skyline*, observing that we have the set of all monotone scoring functions in the latter and just one function in the former. The generalization of both equations to the case where we have any set of monotone scoring functions gives rise to the so-called *non-dominated flexible skyline*:

Definition 3. *Let t and s be tuples over a schema S and $\mathcal{F}$ a set of monotone scoring functions over S; t $\mathcal{F}$-*dominates *s, denoted $t \prec_{\mathcal{F}} s$, iff, $\forall f \in \mathcal{F}\,.\, f(t) \leq f(s)$ and $\exists f \in \mathcal{F}\,.\, f(t) < f(s)$. The* non-dominated flexible skyline $\text{ND}(r;\mathcal{F})$ *of a relation r with respect to $\mathcal{F}$ is the set $\{t \in r \mid \nexists s \in r\,.\, s \prec_{\mathcal{F}} t\}$.*

Generally, $\text{ND}(r;\mathcal{F}) \subseteq \text{SKY}(r)$, since $\mathcal{F} \subseteq \texttt{MF}$. As mentioned in the previous section, in our running example, for $\mathcal{F} = \{w_1x + w_2y \mid w_1 > w_2\}$, we have $\text{ND}(r;\mathcal{F}) = \{\texttt{a}, \texttt{c}\}$, while $\text{SKY}(r) = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{e}\}$, so that the inclusion holds. Moreover, ND generalizes both skylines and top-1 queries, since $\text{SKY}(r) = \text{ND}(r;\texttt{MF})$ and $\text{TOP}_1(r;f) = \text{ND}(r;f)$ (provided, again, that f causes no ties, which are not discarded by ND). With this, [12] introduced a further generalization of the ND operator to the general case $k \geq 1$.

Definition 4. *Given a relation r over S and a set of monotone scoring functions $\mathcal{F}$ over S, the $\text{ND}_k(r;\mathcal{F})$ operator returns the set of tuples in r that are $\mathcal{F}$-dominated by less than k tuples:*

$$\text{ND}_k(r;\mathcal{F}) = \{t \in r \mid \nexists t_1, \ldots, t_k \in r\,.\, t_1 \prec_{\mathcal{F}} t \wedge \ldots \wedge t_k \prec_{\mathcal{F}} t \wedge \texttt{diff}(t_1, \ldots, t_k)\}, \quad (3)$$

where $\texttt{diff}(t_1, \ldots, t_k)$ stands for $\forall i, j\,.\, (1 \leq i, j \leq k \wedge i \neq j) \rightarrow t_i \neq t_j$.

In the running example, we have $\text{ND}_2(r;\mathcal{F}) = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}\}$, since each of these tuples is $\mathcal{F}$-dominated by at most one tuple, while `f` and `g` are $\mathcal{F}$-dominated by (at least) `d` and `e`.

Another common notion that extends skylines to the set of tuples dominated by less than k other tuples is the so-called *k-skyband* [11], which we indicate here as $\text{SKY}_k(r)$ and define as $\text{SKY}_k(r) = \{t \in r \mid \nexists t_1, \ldots, t_k \in r\,.\, \texttt{diff}(t_1, \ldots, t_k) \wedge t_1 \prec t \wedge \ldots \wedge t_k \prec t\}$. Clearly, $\text{ND}_k(r, \texttt{MF}) = \text{SKY}_k(r)$ and, in general, $\text{ND}_k(r, \mathcal{F}) \subseteq \text{SKY}_k(r)$. In Example 1, we have $\text{SKY}_2(r) = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}, \texttt{f}\}$: `f` is included since it is dominated only by `e`, while `g` is discarded, being dominated by three tuples (`c`, `e`, `f`). Therefore, $\text{ND}_2(r;\mathcal{F}) \subseteq \text{SKY}_2(r)$, as expected.

Solutions for computing ND_k in a vertically distributed setting where both sorted and random access are available were proposed in [12] through an algorithm called `FSA` that generalizes two classical algorithms for computing top-k queries: `FA` (Fagin's Algorithm [1]) and `TA` (the Threshold Algorithm [7]). All these algorithms were defined for the so-called "middleware scenario", in which relation r over schema $S = \{Id, A_1, \ldots, A_d\}$ is vertically distributed across relations $r_1, \ldots, r_d$ such that, for all $i \in \{1, \ldots, d\}$, r_i has a schema $\{Id, A_i\}$ and is sorted on A_i in ascending order (i.e., from the best to the worst value). In other words, Id is meant to be a tuple identifier that can be used for reconstructing tuples through random access (and joins), and each r_i is a ranked list. We adopted the same setting in Example 1, where relations r_1 and r_2 represent a vertical distribution of data, and tuples carry an implicit Id (the name `a`, `b`, etc.), which we use for reconstructing the full relation r as $r_1 \bowtie r_2$.

We are also going to consider this kind of distribution, but we are now focusing on settings in which random access is unavailable or too expensive to be viable. For the "no random access" scenario, the relevant literature has described a general pattern, called `NRA`, working for the classical top-k scenario [7]. The main idea is to proceed, in parallel on all ranked lists, from top to bottom through sorted access until we have seen enough tuples to be sure that proceeding with more accesses will not change the solution. In particular, since

a tuple may have been seen only on some but not all ranked lists, for each tuple we keep track of the worst (highest) and the best (lowest) possible scores: when at least k tuples have a worst score that is better than a tuple t's best score, then t can be dismissed. More so, when the seen tuples with the k best worst scores have better worst scores than the best scores of all other tuples (seen or unseen), then `NRA` stops. To determine that no unseen tuple can beat the current top-k tuples, `NRA` watches the best possible score of the so-called *threshold point* τ, i.e., the (virtual) tuple with attribute values corresponding to the last seen values in every ranked list: no unseen tuple can have better values than those in τ, since the lists are ranked; so, if k tuples already beat the threshold point, then no unseen tuple can ever enter the final result.

This pattern, however, was defined for the classical top-k scenario, with just one scoring function in mind. To the best of our knowledge, no algorithm exists that has addressed the "no random access" scenario for the case of the non-dominated flexible skyline and its extension ND_k.

3. Flexible `NRA`

The classical `NRA` scheme consists of the following main steps:

- Access sorted items, one at a time, on all ranked lists. This will unveil the value of a tuple on some list but possibly not in all lists, so we may not be able to compute the overall score of a tuple. We can, anyhow, compute bounds expressing the best and worst possible values for such a score.
- Keep a buffer for all the seen tuples, and for all tuples, compute the worst and the best bound on their overall score[2].
- Also, compute a threshold value for the overall score that may be attained by unseen tuples.
- Repeat until there are k tuples in the buffer whose worst bound is no worse than the best bound of all the other seen tuples and the threshold.
- Return such k objects.

Figure 2 shows all execution steps made by `NRA` on the dataset of Example 1 to find the top-2 tuples according to the scoring function s_c. The execution requires six rounds of sorted accesses on both ranked lists to guarantee that `c` and `e` are the required solutions. The tables in the figure report the accessed tuples sorted in descending order of their worst bound; the current result set is shown with a green background. Observe that the algorithm does not require exploring the lists in their entirety, and, indeed, some of the scores of the tuples are not known when `NRA` stops.

Matters are much more complex in the case of ND_k because obtaining the bounds is more challenging and also because the output size is not limited to k tuples as in a top-k query.

Algorithm 1 shows the pseudocode that we developed for the computation of ND_k. As recognized in the pertinent literature [13], an `NRA`-like algorithm goes through a "growing phase", during which all tuples that may contribute to the final result are collected, followed by a "shrinking phrase", which eliminates all tuples that are not part of the result.

Essentially, during the growing phase (Lines 2–9), tuples are accessed in parallel through sorted access (and inserted in a buffer B) for as long as needed. In particular, we can stop this phase when we have seen at least k objects which, in the worst case, $\mathcal{F}$-dominate the threshold point τ. Since the threshold is the best bound for all unseen tuples, meeting this condition means that no unseen tuple can be part of the result. This roughly corresponds to the so-called sorted access phase of Fagin's Algorithm (`FA`); yet, unlike `FA`, here we cannot proceed with random access to complete the missing parts of the extracted tuples.

This leads us to the shrinking phase (Lines 10–20), which aims to remove all tuples that are not part of the final result. To do this, for each tuple t in the buffer B, we keep track of how many tuples $\mathcal{F}$-dominate t and also of how many cannot $\mathcal{F}$-dominate t. With this, if at least k tuples $\mathcal{F}$-dominate t, we remove t from B. Instead, if a tuple s is not removed and there are still at least k tuples that might $\mathcal{F}$-dominate it, then we need to continue doing sorted access, so as to discover new missing pieces of the tuples in B and update the bounds. The condition on Line 17 expresses precisely this: the number of surviving tuples excluding s, i.e., $B - 1$, minus the number c of those tuples that do not $\mathcal{F}$-dominate s is greater than or equal to k. Clearly, tuple t cannot $\mathcal{F}$-dominate tuple s if the best possible completion of t does not $\mathcal{F}$-dominate the worst possible completion of s (by completion of a tuple t, we mean here the tuple whose attribute values are the same as t's, when available, and then the best or worst still possible for that attribute, if not available in t).

Algorithm 1: Algorithmic pattern for computing ND_k.

```
Input:    Ranked lists r_1, ..., r_d, scoring functions F
Output:   ND_k(r; F)
1.  B := ∅   // buffer of tuples
2.  while lists not exhausted
3.    c := 0   // a counter of tuples F-dominating the threshold point
4.    make a sorted access on r_1, ..., r_d and insert/update extracted tuples in B
5.    τ = ⟨ℓ_1, ..., ℓ_d⟩   // threshold point: last scores on every list
6.    for t in seen tuples
7.      if wb(t) ≺_F τ   // worst bound of t F-dominates τ
8.        if ++c = k   // threshold F-dominated by k tuples
9.          break to Line 10
10. while true   // keep digging if at least one tuple can be F-dominated by k tuples
11.   remove from B tuples F-dominated by other k tuples
12.   for s in B   // candidate non-F-dominated tuples
13.     c := 0   // a counter of non-F-dominance relationships
14.     for t in B \ {s}   // candidate non-F-dominating tuples
15.       if bb(t) ⊀_F wb(s)   // best bound of t does not F-dominate worst bound of s
16.         c++
17.     if k ≤ |B| − 1 − c // if k tuples may F-dominate it, we keep deepening
18.       make a sorted access on r_1, ..., r_d and insert/update extracted tuples in B
19.       continue to Line 10
20.   break
21. return B
```

At the end of the shrinking phase, we are left with tuples that cannot be $\mathcal{F}$-dominated by k or more tuples, so we have our final result.

Theorem 1. *Algorithm 1 correctly computes* ND_k.

Proof. The condition on Line 17 guarantees that B does not contain any tuple that is $\mathcal{F}$-dominated by at least k others. Therefore $B \subseteq \mathrm{ND}_k(r; \mathcal{F})$. In order to show that B coincides with $\mathrm{ND}_k(r; \mathcal{F})$, we need to prove that no object in $r \setminus B$ belongs to $\mathrm{ND}_k(r; \mathcal{F})$. Indeed, when exiting the first **while** loop (growing phase), there are k different tuples $t_1, \ldots, t_k$ $\mathcal{F}$-dominating the threshold τ. If t' is a tuple in $r \setminus B$, t' cannot exceed τ's scores on any of the ranked lists, since t' has not yet been met by sorted access. Therefore t' is necessarily also $\mathcal{F}$-dominated by $t_1, \ldots, t_k$, thus it cannot belong to $\mathrm{ND}_k(r; \mathcal{F})$. □

Round 1

	r_1	r_2	worst bound	best bound
e	?	1.0	11.0	4.0
a	3.0	?	13.0	4.0
τ	3.0	1.0	threshold: 4.0	

Round 2

	r_1	r_2	worst bound	best bound
e	?	1.0	11.0	6.0
f	?	1.5	11.5	6.5
a	3.0	?	13.0	4.5
b	5.0	?	15.0	6.5
τ	5.0	1.5	threshold: 6.5	

Round 3

	r_1	r_2	worst bound	best bound
c	6.0	2.0	8.0	8.0
e	?	1.0	11.0	7.0
f	?	1.5	11.5	7.5
a	3.0	?	13.0	5.0
b	5.0	?	15.0	7.0
τ	6.0	2.0	threshold: 8.0	

Round 4

	r_1	r_2	worst bound	best bound
c	6.0	2.0	8.0	8.0
e	?	1.0	11.0	8.0
f	?	1.5	11.5	8.5
g	?	2.5	12.5	9.5
a	3.0	?	13.0	5.5
b	5.0	?	15.0	7.5
d	7.0	?	17.0	9.5
τ	7.0	2.5	threshold: 9.5	

Round 5

	r_1	r_2	worst bound	best bound
c	6.0	2.0	8.0	8.0
e	8.0	1.0	9.0	9.0
d	7.0	3.0	10.0	10.0
f	?	1.5	11.5	9.5
g	?	2.5	12.5	10.5
a	3.0	?	13.0	6.0
b	5.0	?	15.0	8.0
τ	8.0	3.0	threshold: 11.0	

Round 6

	r_1	r_2	worst bound	best bound
c	6.0	2.0	8.0	8.0
e	8.0	1.0	9.0	9.0
d	7.0	3.0	10.0	10.0
f	9.0	1.5	10.5	10.5
b	5.0	7.0	12.0	12.0
g	?	2.5	12.5	11.5
a	3.0	?	13.0	10.0
τ	9.0	7.0	threshold: 16.0	

Figure 2. Application of NRA on the dataset of Example 1 to find the top-2 tuples according to scoring function $s_c(x, y) = 0.5x + 0.5y$. At each round, NRA makes one sorted access on each ranked list. It stops when e's worst bound is no worse than the best bound of all the other tuples and the threshold. The current top-2 tuples are shown with a green background, while all other accessed tuples are shown with a dark yellow background. Question marks ('?') indicate that the corresponding score is not yet known. The threshold point is shown with a pink background.

Figure 3 shows the application of Algorithm 1 on the dataset of Example 1 to find $\mathrm{ND}_2(r; \mathcal{F})$, with $\mathcal{F} = \{w_1 x + w_2 y \mid w_1 > w_2\}$. When some of the scores are missing, we keep track of its best and worst possible completion for each tuple, which we indicate as "best point" and "worst point".

Round 1

	r_1	r_2	worst point	best point
e	?	1.0	(10.0,1.0)	(3.0,1.0)
a	3.0	?	(3.0,10.0)	(3.0,1.0)
τ	3.0	1.0		

Round 2

	r_1	r_2	worst point	best point
e	?	1.0	(10.0,1.0)	(5.0,1.0)
f	?	1.5	(10.0,1.5)	(5.0,1.5)
a	3.0	?	(3.0,10.0)	(3.0,1.5)
b	5.0	?	(5.0,10.0)	(5.0,1.5)
τ	5.0	1.5		

Round 3

	r_1	r_2	worst point	best point
c	6.0	2.0	(6.0,2.0)	(6.0,2.0)
e	?	1.0	(10.0,1.0)	(6.0,1.0)
f	?	1.5	(10.0,1.5)	(6.0,1.5)
a	3.0	?	(3.0,10.0)	(3.0,2.0)
b	5.0	?	(5.0,10.0)	(5.0,2.0)
τ	6.0	2.0		

Round 4

	r_1	r_2	worst point	best point
c	6.0	2.0	(6.0,2.0)	(6.0,2.0)
e	?	1.0	(10.0,1.0)	(7.0,1.0)
f	?	1.5	(10.0,1.5)	(7.0,1.5)
g	?	2.5	(10.0,2.5)	(7.0,2.5)
a	3.0	?	(3.0,10.0)	(3.0,2.5)
b	5.0	?	(5.0,10.0)	(5.0,2.5)
d	7.0	?	(7.0,10.0)	(7.0,2.5)
τ	7.0	2.5		

Round 5

	r_1	r_2	worst point	best point
c	6.0	2.0	(6.0,2.0)	(6.0,2.0)
e	8.0	1.0	(8.0,1.0)	(8.0,1.0)
d	7.0	3.0	(7.0,3.0)	(7.0,3.0)
f	?	1.5	(10.0,1.5)	(8.0,1.5)
g	?	2.5	(10.0,2.5)	(8.0,2.5)
a	3.0	?	(3.0,10.0)	(3.0,3.0)
b	5.0	?	(5.0,10.0)	(5.0,3.0)
τ	8.0	3.0		

Round 6

	r_1	r_2	worst point	best point
c	6.0	2.0	(6.0,2.0)	(6.0,2.0)
e	8.0	1.0	(8.0,1.0)	(8.0,1.0)
d	7.0	3.0	(7.0,3.0)	(7.0,3.0)
f	9.0	1.5	(9.0,1.5)	(9.0,1.5)
b	5.0	7.0	(5.0,7.0)	(5.0,7.0)
g	?	2.5	(10.0,2.5)	(8.0,2.5)
a	3.0	?	(3.0,10.0)	(3.0,3.0)
τ	9.0	7.0		

Round 7

	r_1	r_2	worst point	best point
c	6.0	2.0	(6.0,2.0)	(6.0,2.0)
e	8.0	1.0	(8.0,1.0)	(8.0,1.0)
d	7.0	3.0	(7.0,3.0)	(7.0,3.0)
b	5.0	7.0	(5.0,7.0)	(5.0,7.0)
a	3.0	8.0	(3.0,8.0)	(3.0,8.0)
τ	9.5	8.0		

Figure 3. Application of Algorithm 1 on the dataset of Example 1 to find $\mathrm{ND}_2(r; \mathcal{F})$, with $\mathcal{F} = \{w_1 x + w_2 y \mid w_1 > w_2\}$. After Round 5, the growing phase stops, since there are two tuples (c and e) whose worst point $\mathcal{F}$-dominates the threshold point (shown with a pink background). All accessed tuples are shown with a dark yellow background, with unknown scores indicated with a question mark ('?'). The tuples removed after Round 6 are shown with a purple background. The final result (Round 7) is shown with a green background.

The growing phase stops after five rounds, when (the worst points of) two tuples (c and e) $\mathcal{F}$-dominate the threshold point. Indeed, at this point, all the tuples that can be part of the result have been seen. However, we cannot yet determine the results because of the missing scores, so we need to keep making sorted access in order to eliminate redundant tuples. Indeed, there is at least one tuple (for instance d) that could be $\mathcal{F}$-dominated by at least two tuples (for instance, c certainly $\mathcal{F}$-dominates d and the best point of f also $\mathcal{F}$-dominates d). The shrinking phase starts at Round 6 and allows discarding f and g, which are both $\mathcal{F}$-dominated by c and e. However, d may still be $\mathcal{F}$-dominated by two tuples (c and a), so one more round is needed to complete a's scores and see that it does not $\mathcal{F}$-dominate d. Round 7 produces the final result.

In top-k scenarios, performance is usually measured in terms of the "depth" of the execution, i.e., the number of sorted accesses made on each ranked list as an indication of the cost incurred by an algorithm. Let $\texttt{depth}(A, I, i)$ indicate the depth reached on r_i by algorithm A before returning a solution to problem I. We define $\texttt{sumDepths}(A, I)$ as $\sum_{i=1}^{d} \texttt{depth}(A, I, i)$. It is also common to assume that only tuples seen via sorted access can be returned as part of the result (no "wild guesses"). The notion of instance optimality characterizes those algorithms that cannot be beaten by an arbitrarily large amount by other algorithms solving the same problem. To this end, let **A** be the class of correct algorithms for computing ND_k with no wild guesses and no random access; let **I** be the set of all instances of ND_k problems. We say that A is instance optimal over **A** and **I** for the sumDepths cost metric if there exist constants c_1 (called *optimality ratio*) and c_2 such that $\forall A' \in \mathbf{A} \,.\, \forall I \in \mathbf{I} \,.\, \texttt{sumDepths}(A, I) \leq c_1 \cdot \texttt{sumDepths}(A', I) + c_2$. We can now show that Algorithm 1 is instance optimal for computing ND_k with no random access.

Theorem 2. *Let* **A** *be the class of correct algorithms for computing* ND_k *with no wild guesses and no random access; let* **I** *be the set of all* $\mathrm{ND}_k(r, \mathcal{F})$ *problems, where* $\mathcal{F}$ *is a set of monotone scoring functions. Then Algorithm 1 is instance optimal over* **A** *and* **I**.

Proof. Consider any instance $I = \langle r, \mathcal{F}, k \rangle$ and any algorithm $A \in \mathbf{A}$. Let δ_i, $1 \leq i \leq d$, be the depth reached by A on list r_i when A halts on I. Since A does not make wild guesses, all tuples seen by A have been extracted by sorted access. The best possible tuple t not seen by A coincides with the threshold point τ_A when A halts. Since A is correct, t is $\mathcal{F}$-dominated by at least k (seen) tuples in the result. Now, when Algorithm 1 reaches depth $\delta = max\{\delta_1, \ldots, \delta_d\}$, it will have seen all tuples seen by A and the corresponding threshold point τ^* will be either coinciding with or $\mathcal{F}$-dominated by τ_A. Therefore τ^* is also necessarily $\mathcal{F}$-dominated by k seen tuples. This means that Algorithm 1's stopping condition is met and, therefore, that $\texttt{sumDepths}(\text{Algorithm 1}, I) \leq \delta \cdot d$, while $\texttt{sumDepths}(A, I) = \sum_{i=1}^{d} \delta_i \geq \delta$, i.e., Algorithm 1 is instance optimal with an optimality ratio of d. □

We observe that our focus here is not on efficiently checking $\mathcal{F}$-dominance, which is used in Algorithm 1 as a black box. The problem of testing $\mathcal{F}$-dominance has been studied extensively in [9,10,12] for specific classes of scoring functions and sets thereof defined by means of linear constraints on weights. We will use these results and these constraints in our experiments.

4. Experiments

In this section, we test our implementation of Algorithm 1 on a number of scenarios, including both synthetic and real datasets. For synthetic datasets, we produce d-dimensional datasets of varying sizes and distributions according to the configurations displayed in

Table 1. In particular, the class ANT comprises datasets with values anti-correlated across different dimensions, while UNI has uniformly distributed values.

In addition to synthetic datasets, we also ran our experiments against a real dataset, NBA, consisting of a 2D selection of stats for 190,862 NBA games from nba.com.

In our experiments, we mainly focus on the number of $\mathcal{F}$-dominance tests required to compute the result and the depth incurred by the algorithm. In addition to these objective measures, we also measure execution times on a machine sporting an 8-Core Intel Core i9 with 32 GB of RAM.

Table 1. Operating parameters for testing efficiency (defaults in bold).

Name	Tested Value
Distribution	synthetic: UNI, ANT; real: NBA
Synthetic dataset size (N)	10 K, 50 K, **100 K**, 500 K, 1 M
# of dimensions (d)	**2**, 3, 4
k	1, 2, 5, **10**, 20, 50, 100
Spread (ε)	none, **1%**, 2%, 5%, 10%, 20%, 50%, full
Batch size (μ)	1, 10, **100**, 1000

In addition to the dataset N, its dimensionality d, and its distribution, we also study the effect of k in the ND_k operator, of the constraints used for defining the set of functions $\mathcal{F}$, and the granularity of the sorted accesses (which may function in batches of μ accesses).

For the constraints, we adopt the so-called *ratio bounds constraints*, i.e., constraints of the form:

$$\bar{w}_i(1-\varepsilon) \leq w_i \leq \bar{w}_i(1+\varepsilon), \tag{4}$$

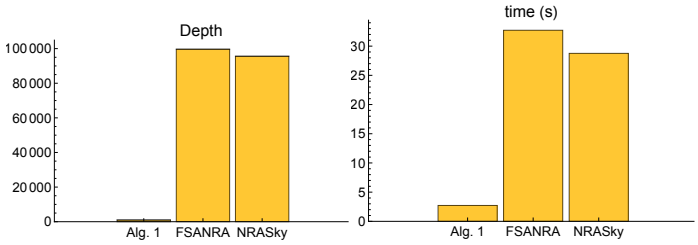
where we set $\bar{w}_i = 1/d$, applied to linear scoring functions with weights $w_1, \ldots, w_d$. The values of ε used in our tests are reported in Table 1; in particular, when $\varepsilon = 0$ (none) the weights are univocally defined and ND_k coincides with a top-k query, whereas when ε can vary freely (full), ND_k coincides with the k-skyband.

In order to better appreciate the effectiveness of Algorithm 1, we also compare it with two baselines. The first baseline, indicated here as FSANRA, consists of adopting the FSA solution proposed in [12], which requires random access, and simulating random access by executing as many sorted accesses as needed so as to find the requested tuple. The second baseline, NRASky, runs a suitably modified version of the NRA algorithm so as to obtain the k-skyband $\mathrm{SKY}_k(r)$ of the dataset, with all the scores, and computes the final results as $\mathrm{ND}_k(\mathrm{SKY}_k(r); \mathcal{F})$ in memory, without performing any further access. The strategy adopted by NRASky to compute the k-skyband requires generalizing NRA's bounds as follows: the worst bound is the maximum possible coordinate for a tuple (i.e., $+\infty$ for incomplete tuples); the best bound equals the minimum possible coordinate of a tuple. When all the tuples in the k-skyband are determined, further sorted accesses are needed to complete all of their scores.

Both baselines perform poorly with respect to Algorithm 1 in terms of both I/O cost (measured as the depth reached by sorted access on the ranked lists during the execution) and execution time. Figure 4 shows how these approaches compare with default parameter values on the UNI dataset. Simulating random access through sorted accesses is extremely impractical, and FSANRA needs to scan the ranked lists almost entirely (over 99,600 tuples out of 100,000 on each list) to find the scores of the tuples that FSA would access via random access. The computation of the k-skyband attempted by NRASky is also very expensive and requires extracting many candidate tuples (1990), only a fraction of which (101) are in the result; discarding the irrelevant tuples requires reaching a depth of 77,621, while completing all the missing scores for such tuples requires a depth of 95,601. Figure 4a

compares the depths incurred by the baselines with the depth reached by Algorithm 1 (3600) in the same conditions. These extreme differences in I/O costs have repercussions also on the execution times, showing that Algorithm 1, with a time of around 2.5 s, clearly outperforms the baselines by at least one order of magnitude (Figure 4b).

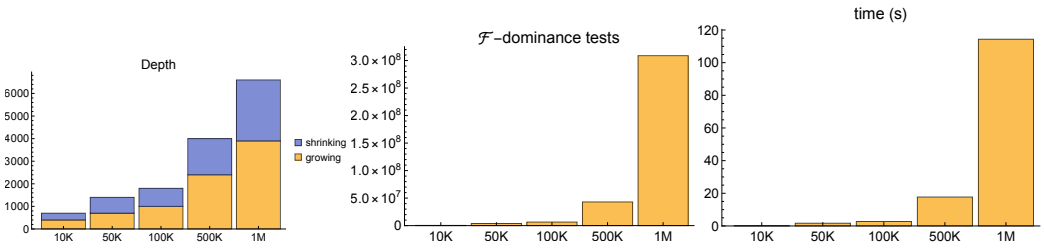
Having ascertained that the baselines do not offer competitive performances when compared to Algorithm 1, we shall disregard them in the next experiments, where we measure the effect of the various operating parameters on Algorithm 1.



(**a**) `depth` (**b**) Time

Figure 4. Depth (**a**) and time (**b**) with default parameter values on `UNI`.

Varying the dataset size N**.** The effect of the dataset size on the computation of ND_k through Algorithm 1 is illustrated in Figure 5. We varied the dataset size on `UNI` using the values indicated in Table 1 and default values (indicated in bold in the table) for all other parameters. Figure 5a reports stacked bars for each of the tested dataset sizes, in which the lower part refers to the depth reached at the end of the growing phase, while the top part indicates the additional depth incurred during the shrinking phase. We observe that, for datasets with uniformly distributed values, such as `UNI`, the depth grows less than linearly with the dataset size, varying from around 7% with $N = 10$ K to around 0.7% with $N = 1$ M. Figure 5b shows the number of $\mathcal{F}$-dominance tests that were executed in order to find the result. In this case, we can see the effect of the quadratic nature of skyline-based operators such as ND_k, with the number of tests varying from 387,810 with $N = 10$ K to 308,877,008 with $N = 1$ M. Execution times are essentially related to the number of $\mathcal{F}$-dominance tests, which are the most expensive operation in the process. Figure 5c shows that such times, in seconds, vary from 0.2 s with $N = 10$ K to 114.3 with $N = 1$ M, i.e., there is a growth of nearly 3 orders of magnitude, as also observed for the number of $\mathcal{F}$-dominance tests.

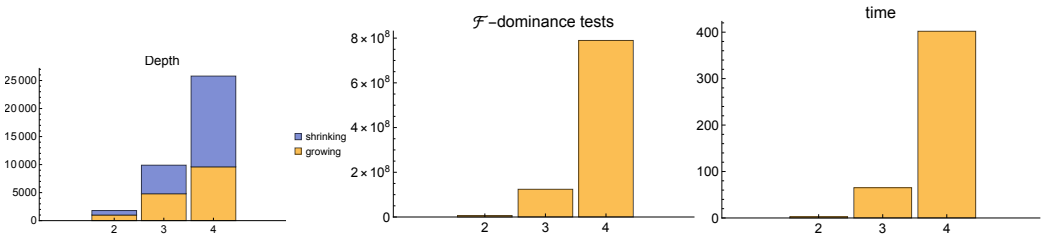


(**a**) `depth` (**b**) $\mathcal{F}$-dominance tests (**c**) Time

Figure 5. Depth (**a**), $\mathcal{F}$-dominance tests (**b**), and time (**c**) as size N varies on `UNI`.

Varying the number of dimensions d**.** Figure 6 shows the effect of the number of dimensions d on our measurements. Algorithms based on a "no random access" (`NRA`) policy heavily suffer from the so-called curse of dimensionality, since an increased number of dimensions entails less likely dominance (and $\mathcal{F}$-dominance) relationships, with result sets growing larger and larger. In such cases, an `NRA` policy essentially mandates a full scan of the dataset, since stopping criteria are met no earlier than that, thereby defeating the very purpose of "early exit" top-k algorithms exploiting the ranking inherent in the vertically distributed sources. For these reasons, we limited our analysis to low values of d (2, 3, and

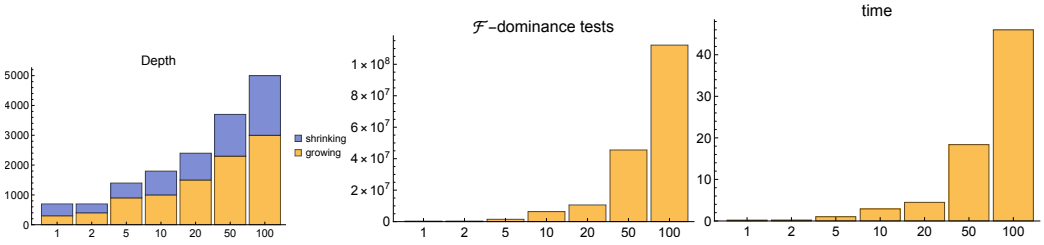
4). While the charts in Figure 6 are analogous to those in Figure 5, here we see that the effect of augmenting d is heavier on the depth, which reaches 25% with 4 dimensions, while it was just 1.8% with $d = 2$. The larger number of involved tuples, with higher values of d, consequently entails larger numbers of $\mathcal{F}$-dominance tests and longer execution times, as can be seen in Figure 6b,c.



(**a**) `depth` (**b**) $\mathcal{F}$-dominance tests (**c**) Time

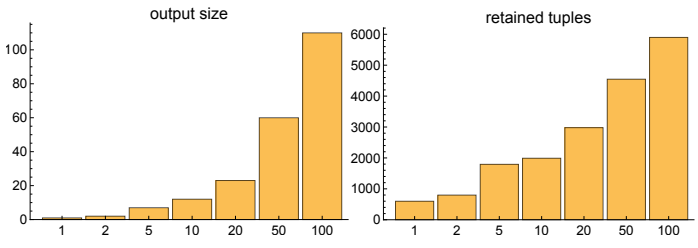
Figure 6. Depth (**a**), $\mathcal{F}$-dominance tests (**b**), and time (**c**) as d varies on `UNI`.

Varying k. Figure 7 shows the effect of k on `UNI`. While k is not an exact output size in the case of ND_k, it can be considered as the initial output size, which applies when $\mathcal{F}$ contains just one function. We observe here that the depth grows from 0.7% when $k = 1$ to 5% when $k = 100$, i.e., less than linearly as k grows. The number of $\mathcal{F}$-dominance tests and the execution times are, again, tightly connected and mainly depend on the number of tuples that are retained in the growing phase and that, consequently, might need to be removed in the shrinking phase. To this end, Figure 8 shows how the number of retained tuples varies from right after the growing phase, i.e., when the buffer has its largest size, shown in Figure 8b, to the end of the execution, when the buffer contains the final result, whose size $|\mathrm{ND}_k|$ is shown in Figure 8a. With our default spread value $\varepsilon = 0.01$, the output size does not grow too much larger than k, topping $k + 10$ for $k \geq 50$. Instead, the number of tuples retained at the end of the growing phase goes from just 599 for $k = 1$ to 5898 for $k = 100$, thus causing the steep increase in the number of $\mathcal{F}$-dominance tests shown in Figure 7b.



(**a**) `depth` (**b**) $\mathcal{F}$-dominance tests (**c**) Time

Figure 7. Depth (**a**), $\mathcal{F}$-dominance tests (**b**), and time (**c**) as k varies on `UNI`.

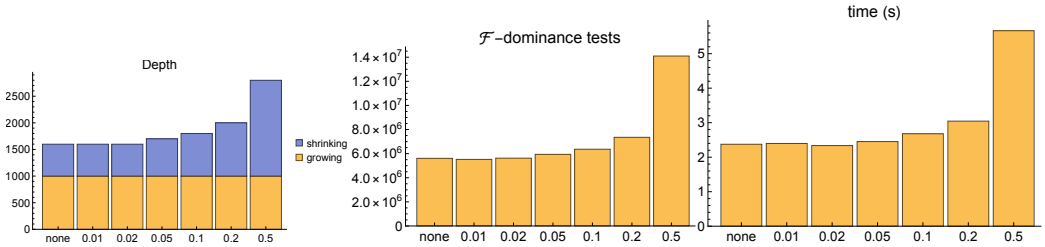


(**a**) Output size (**b**) Tuples after growing phase

Figure 8. Output size (**a**) and number of tuples retained after the growing phase (**b**) as k varies on `UNI`.

Varying the spread ε. The effect of $\mathcal{F}$, and, more precisely, of the constraints used on the weights to determine $\mathcal{F}$ is shown in Figure 9. In particular, we vary the spread ε of

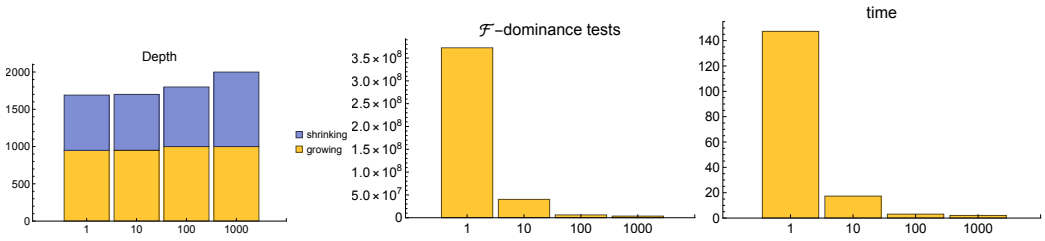
the constraints shown in (4) so that the ND_k operator ranges from a pure top-k query (with just one linear scoring function) to a pure k-skyband query (with all possible linear scoring functions, which, as is well known [10], result-wise have the same power as all the monotone scoring functions). Figure 9a shows that, on the `UNI` dataset with default parameter values, small values of ε make ND_k deviate very little from the behavior of a top-k query—and this is also confirmed in terms of $\mathcal{F}$-dominance tests (Figure 9b) and execution time (Figure 9c). Some growth is visible starting at $\varepsilon = 0.05$, and it is definitely evident for $\varepsilon = 0.5$, where the depth nearly doubles with respect to the case $\varepsilon = 0$ (none). However, we also observe that the computational toll is entirely ascribable to the shrinking phase, which requires more deepening to satisfy looser constraints. We intentionally left out of the charts the case where ε can vary freely (full), because there we experience an explosion in the depth (reaching 77.6% vs. just 2.8% with $\varepsilon = 0.5$) as well as in the $\mathcal{F}$-dominance tests (nearly 53 M vs. 14 M) and execution times (24.9 s vs. 5.7 s), which would make the charts difficult to read.



(**a**) `depth` (**b**) $\mathcal{F}$-dominance tests (**c**) Time

Figure 9. Depth (**a**), $\mathcal{F}$-dominance tests (**b**), and time (**c**) as spread ε varies on `UNI`.

Varying the batch size μ. In order to reduce the number of times the stopping criterion is checked (which requires a high number of $\mathcal{F}$-dominance tests), one could try to increase the number of rows read by sorted access at once. Normally, d sorted accesses (one per list) are made, and then the threshold-based stopping condition is checked (this happens during both the growing phase and the shrinking phase). Reducing the frequency of the checks to once per batch of accesses may entail a significant speed-up. Additionally, this behavior mimics the case of online services returning results in pages of a given size μ. Figure 10 shows the effect of varying μ from the no-batch scenario $\mu = 1$ to $\mu = 1000$. While $\mu = 1$ guarantees that the minimum depth will be attained during the execution, larger values will have looser guarantees on the depth, but might drastically reduce the number of incurred $\mathcal{F}$-dominance tests and, consequently, the execution time. Figure 10a shows that increasing μ causes the final depth to be a multiple of the batch size μ itself, but this negative effect may not be overall prevalent. Indeed, while with $\mu = 1$ we reach the minimum depth (1691), this only increases to 1700 when $\mu = 10$ and to 1800 when $\mu = 100$, while the largest increase is experienced for $\mu = 1000$, with a depth of 2000. We observe that, while the depth is more or less stable for this dataset during the growing phase when μ varies (with values ranging from 949 to 1000), larger changes are found during the shrinking phase (values from 742 to 1000). However, the increase in depth is worthwhile if we look at the number of $\mathcal{F}$-dominance tests (Figure 10b) and execution times (Figure 10c): the number of $\mathcal{F}$-dominance tests plummets from 372 M when $\mu = 1$ to just 3.6 M when $\mu = 1000$ and times go from 147 s when $\mu = 1$ to just 2 s when $\mu = 1000$. Due to the almost negligible difference between the cases $\mu = 100$ and $\mu = 1000$, we chose the former as the default value to use in the experiments, as it causes the lesser harm to depth.

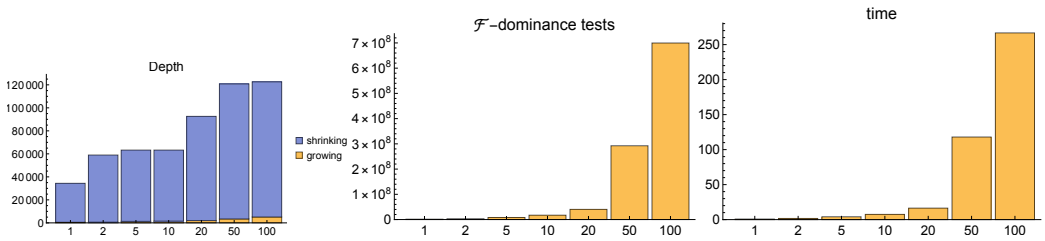


(**a**) `depth` (**b**) $\mathcal{F}$-dominance tests (**c**) Time

Figure 10. Depth (**a**), $\mathcal{F}$-dominance tests (**b**), and time (**c**) as batch size μ varies on `UNI`.

Other datasets. As we mentioned, we also executed our experiments against the `ANT` family of datasets. However, due to the very nature of these datasets, an `NRA`-based approach like the one described in Algorithm 1 is inherently ineffective. Indeed, even with the most favorable working conditions ($d = 2$, $N = 10$ K, $k = 1$, $\varepsilon = 0$, $\mu = 1000$), the depth explored by the algorithm is almost as large as the dataset size. In particular, with this specific configuration, the depth was 90% of the dataset size, and required 58 M $\mathcal{F}$-dominance tests with an execution time of 27.3 s. Clearly, larger dataset sizes and less favorable conditions would determine a full scan of the dataset, with consequently higher execution times and numbers of $\mathcal{F}$-dominance tests.

The real dataset `NBA`, instead, has a distribution that, albeit not uniform, is not anticorrelated either. Figure 11 shows the effect of k on the `NBA` dataset. First of all, we observe that the shrinking phase is prevalent in terms of depth (Figure 11a): the total depth ranges from 59,000 to 122,600, while the depth relative to the growing phase only varies between 500 and 5100. This has corresponding repercussions on the number of $\mathcal{F}$-dominance tests (Figure 11b) and execution times (Figure 11c), which however remain acceptable considering the dataset size and the adverse working conditions of an `NRA`-based algorithm.



(**a**) `depth` (**b**) $\mathcal{F}$-dominance tests (**c**) Time

Figure 11. Depth (**a**), $\mathcal{F}$-dominance tests (**b**), and time (**c**) as k varies on `NBA`.

Final observations. Our experiments show that the algorithmic scheme we proposed for computing ND_k is effective in computing the results, especially in scenarios regarding uniformly distributed data, which are more likely to allow early termination even in the absence of random access. In all other scenarios, particularly unfavorable configurations might be difficult to manage. In particular, when the growing phase first ends, the passage to the shrinking phase charges a heavy computational toll, since many tuples are in the buffer and many need to be removed, with non-negligible costs that are quadratic in the buffer size.

Our scheme proves to be versatile, in that it allows for a simple tuning of the batch size μ, which, while slightly worsening the total incurred depth, might heavily reduce the number of tests and, consequently, the overall execution times.

5. Related Work

Three decades of work on top-k queries and skyline queries have generated a large body of research, comprising numerous variants that have tried to enrich the expressivity of these queries and to overcome their main limitations.

This work lies at the culmination of a series of efforts to integrate and reconcile these two orthogonal approaches. Indeed, the ND_k operator may behave both as a top-k query and as a k-skyband (and, ultimately, as a skyline).

Many algorithms for computing classical skylines exist in the centralized case [8,11,14–21], but several works targeting a distributed setting for the top-k scenario are also available. In particular, *vertical partitioning*, which is also covered in the present work, was studied in [22] for the so-called *middleware scenario* and in several follow-up works [7,23–26], even for the case of non-monotone scoring functions [27–29]. The work [26] distinguishes itself for the idea of retrieving large chunks of records, in a way that is not dissimilar from the batch size parameter we used in the experiments.

The first attempts handling the "no random access" scenario are the Stream-Combine algorithm [30] and `NRA` [7,31]. As observed in [32], Stream-Combine does not stop until all scores of all top-k results are retrieved, thereby losing instance optimality, while `NRA` attains it but may return objects whose scores are not fully known. In [33], the authors present several probabilistic variants of `NRA`. In [13], the `LARA` 2-phase algorithm is proposed, which, by exploiting a lattice structure, achieves faster performance than `NRA`, but only marginal gains in terms of accesses. A different category of algorithms (`3P-NRA`) is proposed in [34], where the phases are three and some heuristics are proposed to improve the runtime cost. In [35,36], the authors identify a selective, non-symmetric approach that avoids some unnecessary accesses. As already mentioned, none of these approaches can handle flexible skylines.

Many strategies also exist for *horizontal partitioning*, especially targeted at skylines and exploiting parallel computation paradigms such as MapReduce or Spark. These studies include works leveraging MapReduce for skyline processing over large-scale datasets [37–42], addressing challenges such as data partitioning, load balancing, and minimizing communication overhead to enhance performance and scalability. Other studies explore the utilization of Graphics Processing Units (GPUs) to accelerate skyline computations [43], especially geared toward efficient dominance checks by leveraging GPU parallelism. Further works investigate skyline computation in distributed environments, including peer-to-peer networks [44–46], focusing on issues like data distribution, network latency, and decentralized control. Several advanced skyline variants and applications are addressed in [47–51], including reverse skyline, spatial skyline, and their applications.

Top-k queries exist in a variety of formats and diversified goals and application contexts, including joins [52–54], incomplete data [55], probabilistic and uncertain data [56–58], spatio-temporal data [59,60], RDF [61], geometric approaches [62], crowdsourcing [63], reverse top-k queries [64], streaming data [65], blockchain [66], privacy preservation [67], and more [68]. Such queries have recently been hybridized with skylines so as to obtain the benefits of both paradigms [9,69,70]. In particular, several previous approaches have tried to empower skylines with preferences and with the ability to limit their output [10,12,70]. A radically different approach tries to minimize the regret associated with the selected result set [71].

Further limitations of top-k queries, which the advent of flexible skylines has tried to address, reside in the expressivity of their scoring functions [72], and the robustness of their results, which can be measured through various indicators [73–77].

We also observe that both skylines and ranking queries are commonly included as typical parts of complex data preparation pipelines for subsequent processing based,

e.g., on Machine Learning or Clustering algorithms [78–81] as well as crowdsourcing applications [82–84] and flexible query-answering systems [85].

Finally, we point out that the access constraints imposed by sorted accesses or random accesses (when available) are akin to the access limitations that have been studied extensively in the field of Web data access, which, indeed, were also studied under the top-k perspective [86]. Originally introduced in [87], such constraints are surveyed in [88,89] and their interaction with query answering is studied in [90–97] for conjunctive queries, in [98–103] for more expressive query constructs, and in [104] for their application in the Deep Web.

6. Conclusions

In this paper, we studied the problem of computing the non-k-dominated flexible skyline—a complex, skyline-based operator that encompasses the common characteristics of top-k queries and skyline queries and is based on a set of scoring functions $\mathcal{F}$, instead of just one, as in the case of top-k queries, or none, as in the case of skyline queries. In particular, we studied the application scenario in which data are vertically distributed (the so-called "middleware scenario") in several ranked lists. Moreover, random access is not available, so data can only be accessed from the lists, from top to bottom. We propose an algorithm for computing the results in two phases: a growing phase, in which all candidate results are incorporated into a buffer, and a shrinking phase, in which all tuples not part of the final result are removed from the buffer. Our algorithmic scheme is not only correct but is also instance-optimal within the class of algorithms that make no random access.

We conducted extensive experiments on various configurations, targeting different datasets. We observed that, in adverse conditions, an algorithm that cannot exploit random access tends to need to consume the entire dataset, with little or no chance of experiencing an early exit. In more convenient scenarios, for instance those regarding non-anticorrelated data, we obtain acceptable execution conditions and can even exploit optimization opportunities, which grant a good trade-off between the final depth of the execution (i.e., a measure of the I/O cost) and the number of tests that need to be performed in order to compute the results.

Future work may try to further optimize the execution by leveraging other optimization opportunities that have been adopted for scenarios in which random access was available, such as memoization-based techniques. Another interesting line of research regards the computation, in a no-random-access scenario, of other flexible skyline variants such as the PO_k operator.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the author.

Acknowledgments: The author wishes to thank Paolo Ciaccia for an insightful discussion on the manuscript.

Conflicts of Interest: The author declares no conflicts of interest.

Notes

1 More precisely, if we call α the score of a tuple with attribute values x and y, we have $\alpha = 0.5x + 0.5y$, corresponding to the straight line $y = -x + 2\alpha$, whose angular coefficient is -1, i.e., $-45°$.

2 The original algorithm uses lower and upper bounds. For generality, with respect to the adopted convention, we prefer to talk about the worst and best bounds here.

References

1. Fagin, R. Combining Fuzzy Information from Multiple Systems. In Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Montreal, QC, Canada, 3–5 June 1996; pp. 216–226. [CrossRef]
2. Ertz, M.; Leblanc-Proulx, D.; Sarigöllü, E.; Morin, C. Web Scraping Techniques and Applications: A Literature Review. *J. Bus. Res.* **2023**, *142*, 1–13. [CrossRef]
3. Carro, M. NoSQL Databases. *arXiv* **2014**, arXiv:1401.2101.
4. Zhang, W.; Liu, J.; Chen, L. Automatic Web Data API Creation via Cross-Lingual Neural Pagination. In Proceedings of the 2022 International Conference on Web Engineering, Bari, Italy, 5–8 July 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 115–130. [CrossRef]
5. McSherry, F.; Lattuada, A.; Schwarzkopf, M.; Roscoe, T. Shared Arrangements: Practical Inter-Query Sharing for Streaming Dataflows. *Proc. VLDB Endow.* **2020**, *13*, 1793–1806. [CrossRef]
6. Alabdulkarim, A.; Bhowmick, S.S. Efficient and Secure Multiparty Querying over Federated Graph Databases. In Proceedings of the 2024 International Conference on Data Engineering (ICDE), Utrecht, The Netherlands, 13–16 May 2024; pp. 1234–1245. [CrossRef]
7. Fagin, R.; Lotem, A.; Naor, M. Optimal Aggregation Algorithms for Middleware. In Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Santa Barbara, CA, USA, 21–23 May 2001. [CrossRef]
8. Börzsönyi, S.; Kossmann, D.; Stocker, K. The Skyline Operator. In Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany, 2–6 April 2001; pp. 421–430. [CrossRef]
9. Ciaccia, P.; Martinenghi, D. Reconciling Skyline and Ranking Queries. *Proc. VLDB Endow.* **2017**, *10*, 1454–1465. [CrossRef]
10. Ciaccia, P.; Martinenghi, D. Flexible Skylines: Dominance for Arbitrary Sets of Monotone Functions. *ACM Trans. Database Syst.* **2020**, *45*, 18:1–18:45. [CrossRef]
11. Papadias, D.; Tao, Y.; Fu, G.; Seeger, B. Progressive skyline computation in database systems. *ACM Trans. Database Syst.* **2005**, *30*, 41–82. [CrossRef]
12. Ciaccia, P.; Martinenghi, D. FA + TA < FSA: Flexible Score Aggregation. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, 22–26 October 2018; pp. 57–66. [CrossRef]
13. Mamoulis, N.; Yiu, M.L.; Cheng, K.H.; Cheung, D.W. Efficient top-*k* aggregation of ranked inputs. *ACM Trans. Database Syst.* **2007**, *32*, 19. [CrossRef]
14. Kung, H.T.; Luccio, F.; Preparata, F.P. On Finding the Maxima of a Set of Vectors. *J. ACM* **1975**, *22*, 469–476. [CrossRef]
15. Chomicki, J.; Godfrey, P.; Gryz, J.; Liang, D. Skyline with Presorting. In Proceedings of the 19th International Conference on Data Engineering, Bangalore, India, 5–8 March 2003; pp. 717–719. [CrossRef]
16. Godfrey, P.; Shipley, R.; Gryz, J. Maximal Vector Computation in Large Data Sets. In Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, 30 August–2 September 2005; Böhm, K., Jensen, C.S., Haas, L.M., Kersten, M.L., Larson, P., Ooi, B.C., Eds.; ACM: New York, NY, USA, 2005; pp. 229–240.
17. Bartolini, I.; Ciaccia, P.; Patella, M. SaLSa: Computing the skyline without scanning the whole sky. In Proceedings of the 2006 ACM CIKM International Conference on Information and Knowledge Management, Arlington, VA, USA, 6–11 November 2006; Yu, P.S., Tsotras, V.J., Fox, E.A., Liu, B., Eds.; ACM: New York, NY, USA, 2006; pp. 405–414. [CrossRef]
18. Godfrey, P.; Shipley, R.; Gryz, J. Algorithms and analyses for maximal vector computation. *VLDB J.* **2007**, *16*, 5–28. [CrossRef]
19. Chomicki, J. Semantic optimization techniques for preference queries. *Inf. Syst.* **2007**, *32*, 670–684. [CrossRef]
20. Lee, K.C.K.; Zheng, B.; Li, H.; Lee, W. Approaching the Skyline in Z Order. In Proceedings of the 33rd International Conference on Very Large Data Bases, Vienna, Austria, 23–27 September 2007; Koch, C., Gehrke, J., Garofalakis, M.N., Srivastava, D., Aberer, K., Deshpande, A., Florescu, D., Chan, C.Y., Ganti, V., Kanne, C., et al., Eds.; ACM: New York, NY, USA, 2007; pp. 279–290.
21. Sheng, C.; Tao, Y. On finding skylines in external memory. In Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, Athens, Greece, 12–16 June 2011; Lenzerini, M., Schwentick, T., Eds.; ACM: New York, NY, USA, 2011; pp. 107–116. [CrossRef]
22. Fagin, R. Fuzzy Queries in Multimedia Database Systems. In Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Seattle, WA, USA, 1–3 June 1998; pp. 1–10. [CrossRef]
23. Akbarinia, R.; Pacitti, E.; Valduriez, P. Best Position Algorithms for Top-k Queries. In Proceedings of the 33rd International Conference on Very Large Data Bases, Vienna, Austria, 23–27 September 2007; Koch, C., Gehrke, J., Garofalakis, M.N., Srivastava, D., Aberer, K., Deshpande, A., Florescu, D., Chan, C.Y., Ganti, V., Kanne, C., et al., Eds.; ACM: New York, NY, USA, 2007; pp. 495–506.
24. Bast, H.; Majumdar, D.; Schenkel, R.; Theobald, M.; Weikum, G. IO-Top-k: Index-access Optimized Top-k Query Processing. In Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Republic of Korea, 12–15 September 2006; Dayal, U., Whang, K., Lomet, D.B., Alonso, G., Lohman, G.M., Kersten, M.L., Cha, S.K., Kim, Y., Eds.; ACM: New York, NY, USA, 2006; pp. 475–486.

25. Schnaitter, K.; Polyzotis, N. Evaluating rank joins with optimal cost. In Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, Vancouver, BC, Canada, 9–11 June 2008; Lenzerini, M., Lembo, D., Eds.; ACM: New York, NY, USA, 2008; pp. 43–52. [CrossRef]
26. Lange, D.; Naumann, F. Bulk sorted access for efficient top-k retrieval. In Proceedings of the Conference on Scientific and Statistical Database Management, SSDBM '13, Baltimore, MD, USA, 29–31 July 2013; Szalay, A., Budavari, T., Balazinska, M., Meliou, A., Sacan, A., Eds.; ACM: New York, NY, USA, 2013; pp. 39:1–39:4. [CrossRef]
27. Luo, Y.; Wang, W.; Lin, X.; Zhou, X.; Wang, J.; Li, K. SPARK2: Top-k Keyword Query in Relational Databases. *IEEE Trans. Knowl. Data Eng.* **2011**, *23*, 1763–1780. [CrossRef]
28. Luo, Y.; Lin, X.; Wang, W.; Zhou, X. Spark: Top-k keyword query in relational databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, 12–14 June 2007; Chan, C.Y., Ooi, B.C., Zhou, A., Eds.; ACM: New York, NY, USA, 2007; pp. 115–126. [CrossRef]
29. Xin, D.; Han, J.; Chang, K.C. Progressive and selective merge: Computing top-k with ad-hoc ranking functions. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, 12–14 June 2007; Chan, C.Y., Ooi, B.C., Zhou, A., Eds.; ACM: New York, NY, USA, 2007; pp. 103–114. [CrossRef]
30. Güntzer, U.; Balke, W.; Kießling, W. Towards Efficient Multi-Feature Queries in Heterogeneous Environments. In Proceedings of the 2001 International Symposium on Information Technology (ITCC 2001), Las Vegas, NV, USA, 2–4 April 2001; pp. 622–628. [CrossRef]
31. Fagin, R.; Lotem, A.; Naor, M. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.* **2003**, *66*, 614–656. [CrossRef]
32. Fagin, R. Combining Fuzzy Information: An Overview. *SIGMOD Rec.* **2002**, *31*, 109–118. [CrossRef]
33. Theobald, M.; Weikum, G.; Schenkel, R. Top-k Query Evaluation with Probabilistic Guarantees. In Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, ON, Canada, 31 August–3 September 2004; Nascimento, M.A., Özsu, M.T., Kossmann, D., Miller, R.J., Blakeley, J.A., Schiefer, K.B., Eds.; Morgan Kaufmann: San Francisco, CA, USA, 2004; pp. 648–659. [CrossRef]
34. Gurský, P.; Vojtás, P. Speeding Up the NRA Algorithm. In Proceedings of the Scalable Uncertainty Management, Second International Conference, SUM 2008, Naples, Italy, 1–3 October 2008; Lecture Notes in Computer Science; Greco, S., Lukasiewicz, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5291, pp. 243–255. [CrossRef]
35. Yuan, J.; Sun, G.; Tian, Y.; Chen, G.; Liu, Z. Selective-NRA Algorithms for Top-k Queries. In Proceedings of the Advances in Data and Web Management, Joint International Conferences, APWeb/WAIM 2009, Suzhou, China, 2–4 April 2009; Lecture Notes in Computer Science; Li, Q., Feng, L., Pei, J., Wang, X.S., Zhou, X., Zhu, Q., Eds.; Springer: Berlin/Heidelberg, Germany, 2009, Volume 5446, pp. 15–26. [CrossRef]
36. Yuan, J.; Sun, G.; Luo, T.; Lian, D.; Chen, G. Efficient processing of top-k queries: Selective NRA algorithms. *J. Intell. Inf. Syst.* **2012**, *39*, 687–710. [CrossRef]
37. Chen, L.; Hwang, K.; Wu, J. MapReduce Skyline Query Processing with a New Angular Partitioning Approach. In Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPS 2012, Shanghai, China, 21–25 May 2012; pp. 2262–2270. [CrossRef]
38. Mullesgaard, K.; Pederseny, J.L.; Lu, H.; Zhou, Y. Efficient Skyline Computation in MapReduce. In Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, 24–28 March 2014. [CrossRef]
39. Zhang, J.; Jiang, X.; Ku, W.; Qin, X. Efficient Parallel Skyline Evaluation Using MapReduce. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 1996–2009. [CrossRef]
40. Koh, J.; Chen, C.; Chan, C.; Chen, A.L.P. MapReduce skyline query processing with partitioning and distributed dominance tests. *Inf. Sci.* **2017**, *375*, 114–137. [CrossRef]
41. Kim, J.; Kim, M.H. An efficient parallel processing method for skyline queries in MapReduce. *J. Supercomput.* **2018**, *74*, 886–935. [CrossRef]
42. Wang, W.; Zhang, J.; Sun, M.; Ku, W. Efficient Parallel Spatial Skyline Evaluation Using MapReduce. In Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, 21–24 March 2017; Markl, V., Orlando, S., Mitschang, B., Andritsos, P., Sattler, K., Breß, S., Eds.; OpenProceedings.org: Konstanz, Germany, 2017; pp. 426–437. [CrossRef]
43. Li, C.; Gu, Y.; Qi, J.; Yu, G. SkyCell: A Space-Pruning Based Parallel Skyline Algorithm. *arXiv* **2021**, arXiv:2107.09993.
44. Cui, B.; Chen, L.; Xu, L.; Lu, H.; Song, G.; Xu, Q. Efficient Skyline Computation in Structured Peer-to-Peer Systems. *IEEE Trans. Knowl. Data Eng.* **2009**, *21*, 1059–1072. [CrossRef]
45. Lee, J.; Hwang, S. Scalable skyline computation using a balanced pivot selection technique. *Inf. Syst.* **2014**, *39*, 1–21. [CrossRef]
46. Han, X.; Wang, B.; Li, J.; Gao, H. Ranking the big sky: Efficient top-k skyline computation on massive data. *Knowl. Inf. Syst.* **2019**, *60*, 415–446. [CrossRef]
47. Song, B.; Liu, A.; Ding, L. Efficient Top-k Skyline Computation in MapReduce. In Proceedings of the 12th Web Information System and Application Conference, WISA 2015, Jinan, China, 11–13 September 2015; pp. 67–70. [CrossRef]

48. Liu, A. Top-k Skyline Result Optimization Algorithm in MapReduce. In Proceedings of the 14th International Conference on Computer Science & Education, ICCSE 2019, Toronto, ON, Canada, 19–21 August 2019; pp. 466–471. [CrossRef]
49. Li, C.; Gu, Y.; Qi, J.; Yu, G. Parallel Skyline Processing Using Space Pruning on GPU. In Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, 17–21 October 2022; Hasan, M.A., Xiong, L., Eds.; ACM: New York, NY, USA, 2022; pp. 1074–1083. [CrossRef]
50. Tang, M.; Yu, Y.; Aref, W.G.; Malluhi, Q.M.; Ouzzani, M. Efficient Parallel Skyline Query Processing for High-Dimensional Data. In Proceedings of the 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, 8–11 April 2019; pp. 2113–2114. [CrossRef]
51. Wijayanto, H.; Wang, W.; Ku, W.; Chen, A.L.P. LShape Partitioning: Parallel Skyline Query Processing Using MapReduce. *IEEE Trans. Knowl. Data Eng.* **2022**, *34*, 3363–3376. [CrossRef]
52. Patil, M.; Shah, R.; Thankachan, S.V. Top-*k* join queries: Overcoming the curse of anti-correlation. In Proceedings of the 17th International Database Engineering & Applications Symposium, IDEAS '13, Barcelona, Spain, 9–11 October 2013; Desai, B.C., Larriba-Pey, J.L., Bernardino, J., Eds.; ACM: New York, NY, USA, 2013; pp. 76–85. [CrossRef]
53. Martinenghi, D.; Tagliasacchi, M. Proximity Rank Join. *Proc. VLDB Endow.* **2010**, *3*, 352–363. [CrossRef]
54. Martinenghi, D.; Tagliasacchi, M. Cost-Aware Rank Join with Random and Sorted Access. *IEEE Trans. Knowl. Data Eng.* **2012**, *24*, 2143–2155. [CrossRef]
55. Miao, X.; Gao, Y.; Zheng, B.; Chen, G.; Cui, H. Top-k Dominating Queries on Incomplete Data. *IEEE Trans. Knowl. Data Eng.* **2016**, *28*, 252–266. [CrossRef]
56. Soliman, M.A.; Ilyas, I.F.; Chang, K.C. Probabilistic top-*k* and ranking-aggregate queries. *ACM Trans. Database Syst.* **2008**, *33*, 13:1–13:54. [CrossRef]
57. Liu, D.; Wan, C.; Xiong, N.; Yang, L.T.; Chen, L. Two Novel Semantics of Top-k Queries Processing in Uncertain Database. In Proceedings of the 10th IEEE International Conference on Computer and Information Technology, CIT 2010, Bradford, UK, 29 June–1 July 2010; pp. 651–659. [CrossRef]
58. Lian, X.; Chen, L. Top-k dominating queries in uncertain databases. In Proceedings of the EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, 24–26 March 2009; ACM International Conference Proceeding Series; Kersten, M.L., Novikov, B., Teubner, J., Polutin, V., Manegold, S., Eds.; ACM: New York, NY, USA, 2009; Volume 360, pp. 660–671. [CrossRef]
59. Li, F.; Yi, K.; Le, W. Top-*k* queries on temporal data. *VLDB J.* **2010**, *19*, 715–733. [CrossRef]
60. Rocha-Junior, J.B.; Nørvåg, K. Top-k spatial keyword queries on road networks. In Proceedings of the 15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, 27–30 March 2012; Proceedings; Rundensteiner, E.A., Markl, V., Manolescu, I., Amer-Yahia, S., Naumann, F., Ari, I., Eds.; ACM: New York, NY, USA, 2012; pp. 168–179. [CrossRef]
61. Wang, D.; Zou, L.; Zhao, D. Top-k queries on RDF graphs. *Inf. Sci.* **2015**, *316*, 201–217. [CrossRef]
62. Mouratidis, K. Geometric Approaches for Top-k Queries. *Proc. VLDB Endow.* **2017**, *10*, 1985–1987. [CrossRef]
63. Lee, J.; Lee, D.; Hwang, S. CrowdK: Answering top-k queries with crowdsourcing. *Inf. Sci.* **2017**, *399*, 98–120. [CrossRef]
64. Vlachou, A.; Doulkeridis, C.; Kotidis, Y.; Nørvåg, K. Reverse top-k queries. In Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, Long Beach, CA, USA, 1–6 March 2010; Li, F., Moro, M.M., Ghandeharizadeh, S., Haritsa, J.R., Weikum, G., Carey, M.J., Casati, F., Chang, E.Y., Manolescu, I., Mehrotra, S., et al., Eds.; IEEE Computer Society: Los Alamitos, CA, USA, 2010; pp. 365–376. [CrossRef]
65. Farazi, S.; Rafiei, D. Top-K Frequent Term Queries on Streaming Data. In Proceedings of the 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, 8–11 April 2019; pp. 1582–1585. [CrossRef]
66. Cheng, J.; Qi, S.; An, B.; Qi, Y.; Wang, J.; Qiao, Y. Lightweight verifiable blockchain top-k queries. *Future Gener. Comput. Syst.* **2024**, *156*, 105–115. [CrossRef]
67. Li, X.; Bai, L.; Miao, Y.; Ma, S.; Ma, J.; Liu, X.; Choo, K.R. Privacy-Preserving Top-kk Spatial Keyword Queries in Fog-Based Cloud Computing. *IEEE Trans. Serv. Comput.* **2023**, *16*, 504–514. [CrossRef]
68. Ilyas, I.F.; Beskales, G.; Soliman, M.A. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.* **2008**, *40*, 1–58. [CrossRef]
69. Mouratidis, K.; Tang, B. Exact Processing of Uncertain Top-k Queries in Multi-criteria Settings. *Proc. VLDB Endow.* **2018**, *11*, 866–879. [CrossRef]
70. Mouratidis, K.; Li, K.; Tang, B. Marrying Top-k with Skyline Queries: Relaxing the Preference Input while Producing Output of Controllable Size. In Proceedings of the SIGMOD '21: International Conference on Management of Data, Virtual Event, China, 20–25 June 2021; pp. 1317–1330. [CrossRef]
71. Nanongkai, D.; Sarma, A.D.; Lall, A.; Lipton, R.J.; Xu, J.J. Regret-Minimizing Representative Databases. *Proc. VLDB Endow.* **2010**, *3*, 1114–1124. [CrossRef]

72. Soliman, M.A.; Ilyas, I.F.; Martinenghi, D.; Tagliasacchi, M. Ranking with uncertain scoring functions: Semantics and sensitivity measures. In Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, 12–16 June 2011; pp. 805–816. [CrossRef]
73. Papadias, D.; Tao, Y.; Fu, G.; Seeger, B. An Optimal and Progressive Algorithm for Skyline Queries. In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, CA, USA, 9–12 June 2003; Halevy, A.Y., Ives, Z.G., Doan, A., Eds.; ACM: New York, NY, USA, 2003; pp. 467–478. [CrossRef]
74. Lin, X.; Yuan, Y.; Zhang, Q.; Zhang, Y. Selecting Stars: The k Most Representative Skyline Operator. In Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, Istanbul, Turkey, 15–20 April 2007; Chirkova, R., Dogac, A., Özsu, M.T., Sellis, T.K., Eds.; IEEE Computer Society: Los Alamitos, CA, USA, 2007; pp. 86–95. [CrossRef]
75. Tao, Y.; Ding, L.; Lin, X.; Pei, J. Distance-Based Representative Skyline. In Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, Shanghai, China, 29 March–2 April 2009; Ioannidis, Y.E., Lee, D.L., Ng, R.T., Eds.; IEEE Computer Society: Los Alamitos, CA, USA, 2009; pp. 892–903. [CrossRef]
76. Mouratidis, K.; Zhang, J.; Pang, H. Maximum Rank Query. *Proc. VLDB Endow.* **2015**, *8*, 1554–1565. [CrossRef]
77. Ciaccia, P.; Martinenghi, D. Directional Queries: Making Top-k Queries More Effective in Discovering Relevant Results. *Proc. ACM Manag. Data* **2024**, *2*, 1–26. [CrossRef]
78. Masciari, E. Trajectory Clustering via Effective Partitioning. In Proceedings of the Flexible Query Answering Systems, 8th International Conference, FQAS 2009, Roskilde, Denmark, 26–28 October 2009; pp. 358–370. [CrossRef]
79. Masciari, E.; Mazzeo, G.M.; Zaniolo, C. Analysing microarray expression data through effective clustering. *Inf. Sci.* **2014**, *262*, 32–45. [CrossRef]
80. Fazzinga, B.; Flesca, S.; Masciari, E.; Furfaro, F. Efficient and effective RFID data warehousing. In Proceedings of the International Database Engineering and Applications Symposium (IDEAS 2009), Cetraro, Calabria, Italy, 16–18 September 2009; ACM International Conference Proceeding Series; Desai, B.C., Saccà, D., Greco, S., Eds.; ACM: New York, NY, USA, 2009; pp. 251–258. [CrossRef]
81. Fazzinga, B.; Flesca, S.; Furfaro, F.; Masciari, E. RFID-data compression for supporting aggregate queries. *ACM Trans. Database Syst.* **2013**, *38*, 11. [CrossRef]
82. Galli, L.; Fraternali, P.; Martinenghi, D.; Tagliasacchi, M.; Novak, J. A Draw-and-Guess Game to Segment Images. In Proceedings of the 2012 International Conference on Privacy, Security, Risk and Trust, PASSAT 2012, and 2012 International Confernece on Social Computing, SocialCom 2012, Amsterdam, The Netherlands, 3–5 September 2012; pp. 914–917. [CrossRef]
83. Loni, B.; Menéndez, M.; Georgescu, M.; Galli, L.; Massari, C.; Altingövde, I.S.; Martinenghi, D.; Melenhorst, M.S.; Vliegendhart, R.; Larson, M.A. Fashion-focused creative commons social dataset. In Proceedings of the Multimedia Systems Conference 2013, MMSys '13, Oslo, Norway, 27 February–1 March 2013; Griwodz, C., Ed.; ACM: New York, NY, USA, 2013; pp. 72–77. [CrossRef]
84. Bozzon, A.; Catallo, I.; Ciceri, E.; Fraternali, P.; Martinenghi, D.; Tagliasacchi, M. A Framework for Crowdsourced Multimedia Processing and Querying. In Proceedings of the First International Workshop on Crowdsourcing Web Search, Lyon, France, 17 April 2012; CEUR Workshop Proceedings; CEUR-WS.org: Aachen, Germany, 2012; Volume 842, pp. 42–47.
85. Martinenghi, D.; Torlone, R. Querying Context-Aware Databases. In Proceedings of the Flexible Query Answering Systems, 8th International Conference, FQAS 2009, Roskilde, Denmark, 26–28 October 2009; Lecture Notes in Computer Science; Andreasen, T., Yager, R.R., Bulskov, H., Christiansen, H., Larsen, H.L., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5822, pp. 76–87. [CrossRef]
86. Deutch, D.; Milo, T.; Polyzotis, N. Top-k queries over web applications. *VLDB J.* **2013**, *22*, 519–542. [CrossRef]
87. Dembinski, P.; Maluszynski, J. AND-Parallelism with Intelligent Backtracking for Annotated Logic Programs. In Proceedings of the 1985 Symposium on Logic Programming, Boston, MA, USA, 15–18 July 1985; pp. 29–38.
88. Halevy, A.Y. Answering Queries Using Views: A Survey. *Very Large Database J.* **2001**, *10*, 270–294. [CrossRef]
89. Millstein, T.D.; Halevy, A.Y.; Friedman, M. Query containment for data integration systems. *J. Comput. Syst. Sci.* **2003**, *66*, 20–39. [CrossRef]
90. Florescu, D.; Levy, A.Y.; Manolescu, I.; Suciu, D. Query Optimization in the Presence of Limited Access Patterns. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, PA, USA, 1–3 June 1999; pp. 311–322.
91. Li, C.; Chang, E. Query Planning with Limited Source Capabilities. In Proceedings of the Sixteenth IEEE International Conference on Data Engineering (ICDE 2000), San Diego, CA, USA, 29 February–3 March 2000; pp. 401–412.
92. Li, C.; Chang, E. On Answering Queries in the Presence of Limited Access Patterns. In Proceedings of the Eighth International Conference on Database Theory (ICDT 2001), London, UK, 4–6 January 2001; pp. 219–233.
93. Li, C.; Chang, E. Answering Queries with Useful Bindings. *ACM Trans. Database Syst.* **2001**, *26*, 313–343. [CrossRef]
94. Li, C. Computing Complete Answers to Queries in the Presence of Limited Access Patterns. *Very Large Database J.* **2003**, *12*, 211–227. [CrossRef]

95. Calì, A.; Martinenghi, D. Conjunctive Query Containment under Access Limitations. In Proceedings of the Conceptual Modeling—ER 2008, 27th International Conference on Conceptual Modeling, Barcelona, Spain, 20–24 October 2008; Lecture Notes in Computer Science; Li, Q., Spaccapietra, S., Yu, E.S.K., Olivé, A., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5231, pp. 326–340. [CrossRef]
96. Calì, A.; Martinenghi, D. Querying Data under Access Limitations. In Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, Cancún, Mexico, 7–12 April 2008; Alonso, G., Blakeley, J.A., Chen, A.L.P., Eds.; IEEE Computer Society: Los Alamitos, CA, USA, 2008; pp. 50–59. [CrossRef]
97. Calì, A.; Calvanese, D.; Martinenghi, D. Dynamic Query Optimization under Access Limitations and Dependencies. *J. Univ. Comput. Sci.* **2009**, *15*, 33–62. [CrossRef]
98. Duschka, O.M.; Levy, A.Y. Recursive Plans for Information Gathering. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97), Nagoya, Japan, 23–29 August 1997; pp. 778–784.
99. Rajaraman, A.; Sagiv, Y.; Ullman, J.D. Answering Queries Using Templates with Binding Patterns. In Proceedings of the Fourteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'95), San Jose, CA, USA, 22–25 May 1995.
100. Deutsch, A.; Ludäscher, B.; Nash, A. Rewriting queries using views with access patterns under integrity constraints. *Theor. Comput. Sci.* **2007**, *371*, 200–226. [CrossRef]
101. Nash, A.; Ludäscher, B. Processing first-order queries under limited access patterns. In Proceedings of the Twentythird ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2004), Paris, France, 14–16 June 2004; pp. 307–318.
102. Ludäscher, B.; Nash, A. Processing union of conjunctive queries with negation under limited access patterns. In Proceedings of the Ninth International Conference on Extending Database Technology (EDBT 2004), Heraklion, Crete, Greece, 14–18 March 2004; pp. 422–440.
103. Yang, G.; Kifer, M.; Chaudhri, V.K. Efficiently ordering subgoals with access constraints. In Proceedings of the Twentyfifth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2006), Chicago, IL, USA, 26–28 June 2006; p. 22.
104. Calì, A.; Martinenghi, D. Querying the deep web. In Proceedings of the EDBT 2010, 13th International Conference on Extending Database Technology, Lausanne, Switzerland, 22–26 March 2010; ACM International Conference Proceeding Series; Manolescu, I., Spaccapietra, S., Teubner, J., Kitsuregawa, M., Léger, A., Naumann, F., Ailamaki, A., Özcan, F., Eds.; ACM: New York, NY, USA, 2010; Volume 426, pp. 724–727. [CrossRef]