Why Not Yet: Fixing a Top-k Ranking that Is Not Fair to Individuals

Zixuan Chen
Northeastern University
Boston, USA
chen.zixu@northeastern.edu

Panagiotis Manolios
Northeastern University
Boston, USA
pete@ccs.neu.edu

Mirek Riedewald
Northeastern University
Boston, USA
m.riedewald@northeastern.edu

ABSTRACT

This work considers why-not questions in the context of top-k queries and score-based ranking functions. Following the popular linear scalarization approach for multi-objective optimization, we study rankings based on the weighted sum of multiple scores. A given weight choice may be controversial or perceived as unfair to certain individuals or organizations, triggering the question why some entity of interest has not yet shown up in the top-k. We introduce various notions of such why-not-yet queries and formally define them as satisfiability or optimization problems, whose goal is to propose alternative ranking functions that address the placement of the entities of interest. While some why-not-yet problems have linear constraints, others require quantifiers, disjunction, and negation. We propose several optimizations, ranging from a monotonic-core construction that approximates the complex constraints with a conjunction of linear ones, to various techniques that let the user control the tradeoff between running time and approximation quality. Experiments with real and synthetic data demonstrate the practicality and scalability of our technique, showing its superiority compared to the state of the art (SOA).

PVLDB Reference Format:
Zixuan Chen, Panagiotis Manolios, and Mirek Riedewald. Why Not Yet: Fixing a Top-k Ranking that Is Not Fair to Individuals. PVLDB, 16(9): 2377 - 2390, 2023.
doi:10.14778/3598581.3598606

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/northeastern-datalab/why-not-yet.

1 INTRODUCTION

Top-k queries, which return tuples in the order dictated by a ranking function, have received significant attention, as indicated by a survey from 2008 [26] with more than 1,000 citations as of October 31, 2022 according to Google Scholar. Since the position in a ranking can determine outcomes of algorithm-influenced decisions, e.g., who gets hired, there is growing interest in critically evaluating the *fairness* of rankings [48, 49]. In this context, a lot of questions have been raised: Why has my favorite hotel not shown up in the top result of a booking application yet [21]? Why has our potential product not shown up in the preference list for this user yet [41]? Why have some good students not shown up in our admission list yet [3]? All these real-world examples from previous papers point to one question: **why** have some expected tuples **not** shown up in the top-k result **yet**? We study such why-not-yet problems: Given the absence of one or more expected tuples from the k top positions, how can the ranking function be fixed?

Example 1 (NBA). *Consider a web site where sports fans can explore rankings of NBA basketball players by controlling how much importance they assign to different features such as points scored (PTS), rebounds (REB), assists (AST), steals (STL), and blocks (BLK). Anita and Bo want to use this service to settle their argument if Luka Dončić, one of the current NBA superstars, belongs to the top-20 of all time. Bo quickly picks equal weights* $(0.2, 0.2, 0.2, 0.2, 0.2)$*, for which Luka does not make it into the top-20. In response, Anita uses our POINT approach, which, within less than a second, confirms that Luka is even a top-10 player for weight vector* $(0.27, 0.12, 0.46, 0.0, 0.14)$*. After a few more rounds of presenting weight combinations that support their respective claims, they agree that it would be more useful to explain in a simple way, under which conditions Luka would reach the top-20. Here Anita and Bo turn to our BOX method, which can find the largest hyper-rectangle such that all weight combinations contained in it would place Luka in the top-20. Depending on preferences, the search can be limited to "reasonable" regions, e.g., setting a minimum weight on PTS, or one can require a sufficiently large range of choices for each weight. Using BOX for target weights in range* $[0, 1.0]$*, they find out that Luka always makes the top-20 for* $w_1 \in [0.62, 0.96]$*,* $w_2 \in [0.55, 0.89]$*,* $w_3 \in [0.65, 1.0]$*,* $w_4 \in [0.0, 0.22]$*, and* $w_5 \in [0.0, 0.1]$*. Intuitively, Luka is in the top-20 if one values offensive skills (PTS, REB, AST are in medium-to-high range) over defense (STL and BLK are low). Now they can focus on arguing if offensive skills should be weighed higher than defensive ones for determining the best NBA players.*

While ranking functions come in many flavors, this work focuses on the common scenario where entities are sorted based on a linear combination of scoring attributes. (See Section 4.4 for more details.) We formally define and address several problems. SAT asks if there exists an attribute weighting, such that the entity of interest reaches rank $\leq k$. If so, then POINT asks to present an example. BEST asks for the best rank the entity of interest could ever reach. And BOX is concerned with presenting the largest (in terms of perimeter or volume) hyper-rectangle B, such that for *each* weight vector W in B, the entity of interest reaches the top-k. In general, a BOX solution compactly represents an infinite number of POINT solutions. And in contrast to a set of POINT solutions, the BOX solution guarantees that there are no "holes," i.e., weight vectors resulting in a low ranking in-between POINT solutions where the

Proceedings of the VLDB Endowment, Vol. 16, No. 9 ISSN 2150-8097.
doi:10.14778/3598581.3598606

expected entity is ranked high. Each of these problems can be constrained to consider only a certain "reasonable" region of the space of possible weight vectors or to enforce a certain "thickness" of the BOX hyper-rectangle.

Example 1 illustrated the use of SAT, POINT and BOX for an application where ranking creator and consumer are the same, i.e., where consumers of a ranking can control the weights of the ranking function [24]. This applies in many other scenarios, e.g., decisions about hiring and university admissions [48]. In cases where consumers cannot directly influence the ranking, e.g., university rankings published by US News and World Report, our technique can still be applied. For example, the ranking creator wants the target audience to trust the ranking. Omitting MIT from the top-10 of US universities in computer science (CS) would not inspire confidence in the ranking itself. Following the same methodology of testing and debugging software with test cases, the ranking creator can apply our approach to analyze and improve the ranking function. In addition to using SAT, POINT, and BOX as in Example 1 for understanding if and how an entity of interest can reach the top-k, BEST provides valuable insights about inherent limits of the ranking function. For instance, if MIT would not reach the top-10 in the CS ranking, *no matter what* weights are chosen, then either relevant scoring attributes are missing, or there are possibly errors in the scores themselves.

As we discuss in Section 7, previous work on reverse top-k and why-not on top-k queries explored related, but different, problems for finding weight assignments that achieve a certain ranking outcome for individual entities. While some can be extended to a sampling-based solution for SAT, BEST, and POINT, the answers obtained that way are inferior to our approach. We make the following main contributions:

(1) We define new why-not-yet problems (Section 2) and formalize them as constraint-satisfaction and optimization problems with linear constraints (Section 3). For modeling tuple ranks, we propose to use indicator variables, which avoid an $O(n^{k-1})$ blowup in the number of combinations of linear constraints. For BOX we propose a novel "monotonic core" approximation that eliminates quantifiers, disjunction, and negation from the constraints, resulting in dramatic performance improvement.

(2) While most of our problems are solvable in polynomial time, running time rapidly increases with data size. Hence we propose optimizations and approximation techniques that trade result quality for improved running time (Section 5). One of them—clustering—can achieve near unlimited scalability, at the cost of possibly low approximation quality.

(3) Our experiments demonstrate the practicality of our approach and its superiority over the SOA (Section 6). They also quantify the tradeoff between running time and approximation quality for our proposed scalability improvements.

2 PROBLEM DEFINITION

Table 1 summarizes our notation. A *top-k query* Q over relation R is defined in SQL as[1]

```
SELECT   id, f_W(A_1, A_2, ..., A_m) AS Score
FROM     R
WHERE    -- some conditions
ORDER BY Score DESC LIMIT k.
```

[1]Depending on the DBMS, the syntax for requesting the top-k rows may differ.

Table 1: Notation

Symbol	Definition
Q	Top-k query
R	Relation
$n = \|R\|$	Number of tuples in R
$A_1, \ldots, A_m$	Attributes of R used for ranking
f_W	Scoring function used to rank the R-tuples
$W = (w_1, \ldots, w_m)$	Weight vector defining f
$\mathcal{P}$	Predicate constraining the choices of W
$\rho_W(r)$	Rank of r for scoring function f_W
$R_W(k)$	Top-k result for scoring function f_W
id	Primary key of R
δ	Indicator of the relationship between two tuples

Conceptually, Q sorts all R-tuples by a scoring function f_W and returns the first k of them. The scoring function is the weighted sum over numerical attributes $A_1, \ldots, A_m$ of R, i.e.,

$$f_W(A_1, A_2, \ldots, A_m) = \sum_{i=1}^{m} w_i A_i, \tag{1}$$

where $W = (w_1, \ldots, w_m)$ and all w_i are non-negative. We will often omit W from the function name. Without loss of generality, we assume that we sort in descending order of scores. We discuss alternative design choices in Section 4.4.

While the notion of *rank* is intuitive—the rank of $r \in R$ is the position of r in sort order—we need to make sure it is well-defined *in the presence of ties*:

Definition 2 (top-k result, rank). *Given a top-k query Q, its* result $R_W(k)$ *is any subset of R of cardinality k that satisfies*

$$\forall r \in R_W(k), r' \in R \setminus R_W(k) : \; f(r) \geq f(r').$$

The rank *of a tuple $r \in R$ is*

$$\rho_W(r) = |\{r' \in R \mid f(r') > f(r)\}| + 1.$$

Let n_1, n_2, and n_3 denote the number of R-tuples whose score is higher than, equal to, and lower than the score of r, respectively. Sorting by score only guarantees that at least n_1 tuples appear before r and at least n_3 after it. However, the ordering of the n_2 tuples tied with r is arbitrary. Hence we define r's rank to be the best position it may reach, i.e., $n_1 + 1$. If $\rho_W(r) \leq k$, we say that r *ranks among the top-k* tuples.

Using this terminology, we are now ready to formally define a variety of interesting problems related to the question"why have I not yet seen tuple r in the output of Q?"

Definition 3 (Satisfiability (SAT), Best possible rank (BEST)). *Given a top-k query Q and the* id *of a tuple $r \in R$:*

- *SAT: Does there exist a weight vector W such that $\rho_W(r) \leq k$, i.e., r ranks among the top-k for some ranking function?*
- *BEST: What is the best rank, $\min_W \rho_W(r)$, tuple r can reach for any scoring function?*

Definition 4 (Inner box). *Given a top-k query Q and the* id *of a tuple $r \in R$, B is an* inner box *for Q and r iff*

(1) $B = [l_1, h_1] \times \cdots \times [l_m, h_m]$, where $\forall i : \; l_i \leq h_i$.

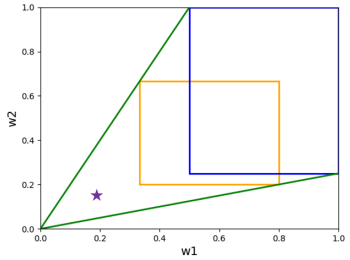


Figure 1: The satisfiability region for a given k (space between the two diagonal lines), an inner point (star), and two inner boxes. SAT asks whether the satisfiability region exists. BEST asks for the smallest k with a non-empty satisfiability region. POINT returns an inner point. BOX returns the largest inner box.

(2) $\forall W \in B : \rho_W(r) \leq k$.

The perimeter and volume of an inner box are defined as $\sum_i (h_i - l_i)$ and $\prod_i (h_i - l_i)$, respectively.

An inner box is an m-dimensional hyper-rectangle of weight vectors, such that tuple r ranks among the top-k for any weight vector contained in it. A special case is an *inner point*, i.e., an inner box where $\forall i : l_i = h_i$. Inner points are isomorphic to weight vectors.

Definition 5 (Inner point (POINT), Max inner box (BOX)). *Given a top-k query Q and the* id *of a tuple $r \in R$:*

- *POINT: Return a weight vector W such that $\rho_W(r) \leq k$, i.e., r ranks among the top-k.*
- *BOX: Return the inner box B with the largest perimeter or volume.*

Figure 1 illustrates the above problems in the 2D space. We are interested in the largest inner boxes, because they compactly describe a large region of the weight-combination space where the desired ranking outcome is achieved. The largest volume maximizes the number of weight combinations, while the largest perimeter optimizes for the largest number of choices per weight dimension.

3 FORMALIZING WHY-NOT-YET

We formalize the problems introduced in Section 2 in a way that enables efficient solutions, starting with a tuple's rank.

3.1 Tuple Rank

The rank of $r \in R$ is determined by the number of R-tuples with higher scores. Another tuple $s \in R$ *beats* r if it has a higher score, i.e., $f(r) < f(s)$:

$$\sum_{i=1}^{m} w_i(r.A_i - s.A_i) < 0. \tag{2}$$

This inequality is a linear constraint, which lets us leverage efficient linear programming approaches to solve problems related to satisfiability and optimization. Adding a constraint like Equation (2) to a linear program enforces that only those weight vectors where s beats r can be selected.

Challenge: How do we express "at most $k-1$ tuples in R beat r" using linear constraints? The main problem is that we do not know which of the R-tuples we want to satisfy Equation (2). For example, requiring r to beat s_1 and s_2 may not be satisfiable, but requiring it to beat s_1 and s_3 might be.

Solution 1: Brute force. To ensure that r ranks among the top-k, it is sufficient for it to beat or tie with $n-k$ tuples, no matter if it beats more than that. Hence we need to explore $\binom{n-1}{n-k} = \binom{n-1}{k-1} = O(n^{k-1})$ linear programs, each containing $n-k$ instances of the negation of Equation (2).

Solution 2: Indicator variables. We propose an approach that uses a single linear program. The core insight is to define *indicator variables*, which are supported by standard linear program solvers. For each $s \in R, s \neq r$, we define an indicator variable δ_s that is 1 if s beats r, and 0 otherwise. To enforce that r ranks among the top-k, we add a constraint for the sum of the indicator variables:

$$\begin{aligned} \delta_s = & \left(\sum_{i=1}^{m} w_i(r.A_i - s.A_i) < 0 \right), \quad s \in R, s \neq r \\ & \sum_{s \in R; s \neq r} \delta_s < k. \end{aligned} \tag{3}$$

Example 6. *Consider $R(A_1, A_2, A_3)$ with $r = (3, 2, 8)$, and $s_1 = (4, 1, 15)$, $s_2 = (1, 1, 14)$, $s_3 = (0, 2, 14)$, and $s_4 = (6, 5, 14)$.[a] For $k = 3$ the constraints are:*

$$\delta_1 = (-w_1 + w_2 - 7w_3 < 0), \delta_2 = (2w_1 + w_2 - 6w_3 < 0),$$
$$\delta_3 = (3w_1 - 6w_3 < 0), \delta_4 = (-3w_1 - 3w_2 - 6w_3 < 0),$$
$$\delta_1 + \delta_2 + \delta_3 + \delta_4 < 3$$

[a] We omit the id for readability in all examples.

3.2 SAT, BEST, and POINT

Since all constraints in Equation (3) are linear in W, we can directly solve SAT using SOA solvers such as Gurobi [19] or Z3 [12]. In addition to answering satisfiability, when the problem is satisfiable, these solvers return a *certificate*, which is an assignment of W. This assignment answers POINT.

To answer BEST, we perform a binary search on the value of k, solving SAT for the corresponding value until we find the smallest k for which Equation (3) is satisfied. Since $k \leq n$, this adds a time factor of $O(\log n)$ compared to answering SAT and POINT.

Example 7. *Continuing Example 6. For SAT the answer is "SATISFIABLE." A POINT solution is $(w_1 = 1, w_2 = 0, w_3 = 0)$. The BEST answer is 2 as illustrated in Figure 2. The weights w_1, w_2, w_3 are mapped to axes in the figure. The red 2D triangle in 3D space represents the set of all W where $w_1 + w_2 + w_3 = 1$, which is a commonly used constraint (see Section 4.2). Note that the star and the other lines fall into the triangle. The star represents a scoring function that ranks r in 5-th place. The three lines in the triangle show the boundaries for indicators $\delta_1, \delta_2, \delta_3$; indicator δ_4 is not visible because its hyperplane does not intersect with the triangle. (The inequality for δ_4 is satisfied for all W in the triangle, therefore r can never beat s_4.) When crossing a boundary, r swaps ranks with the corresponding tuple. The numbers indicate r's rank for the corresponding regions*

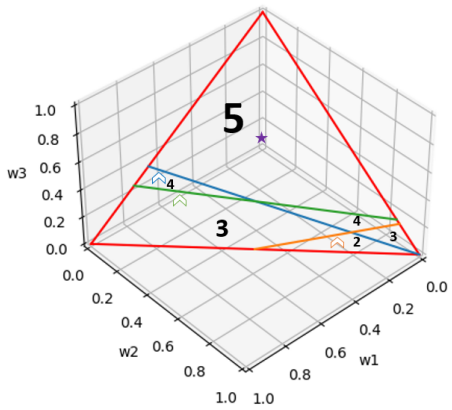


Figure 2: Example 7: Solution space (2D triangle in 3D space), ranking function placing r 5-th (star), and indicator boundaries (colored lines). The numbers indicate r's rank when selecting W from the corresponding region of the triangle.

of the triangle. It is easy to see that r reaches its best rank of 2 when w_1 is "small", w_2 is "large", and w_3 is close to zero.

3.3 BOX

The BOX problem is significantly more challenging to formalize than SAT, BEST, and POINT. First, we need to model the lower and upper bounds of the box. This requires introducing the l_i and h_i as additional variables that must satisfy $h_i - l_i \geq 0$. Second, BOX is an optimization problem (maximize perimeter or volume), not a satisfiability problem. The objective is either linear (for perimeter) or a degree-m polynomial (for volume). The latter often dramatically increases running time of a solver compared to the former. Third, we must tie the W to the l_i and h_i and add constraints to ensure that r is ranked among the top-k for all W in the inner box.

Challenge: How do we enforce that r is ranked among the top-k *for all* W in the inner box?

Solution 1: Direct encoding. We can encode the constraint directly as:

$$\forall w_1, \ldots, w_m : \left(\bigwedge_{i=1}^{m} l_i \leq w_i \leq h_i \right) \Rightarrow \left(\sum_{s \in R; s \neq r} \delta_s < k \right) \quad (4)$$

resulting in max-perimeter optimization problem (for volume, replace the objective function with the corresponding product)

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} (h_i - l_i) \\ \text{s.t.} \quad & l_i \leq h_i, \quad i = 1, \ldots, m \\ & \forall w_1, \ldots, w_m : \left(\bigwedge_{i=1}^{m} l_i \leq w_i \leq h_i \right) \Rightarrow \left(\sum_{s \in R; s \neq r} \delta_s < k \right), \text{where} \\ & \delta_s = \left(\sum_{i=1}^{m} w_i (r.A_i - s.A_i) \ < \ 0 \right), \quad s \in R, s \neq r \end{aligned} \quad (5)$$

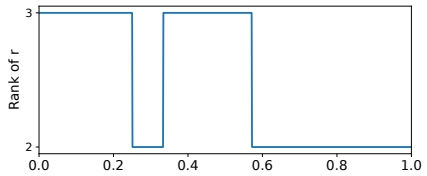


Figure 3: Example 8: The rank of r as a function of weight w_1 is neither convex nor monotonic.

Unfortunately, the use of nested quantifiers gives rise to a problem in the theory of reals, which, while decidable, requires advanced techniques such as cylindrical algebraic decomposition to solve. This is something that current solvers generally do not support and, if they do, the complexity introduced poses serious performance and scalability problems in practice.

Improvement attempt: Convexity and monotonicity. Structural properties of constraints, in particular *convexity* and *monotonicity*, provide opportunities for vastly more efficient optimization. Consider convex function $G(x) = x^2$. Given any interval $[l_x, h_x]$ of x-values, the maximum of $G(x)$ over that interval must be either l_x^2 or h_x^2, i.e., it must fall on one of the interval endpoints l_x and h_x. In general, for convex $G(x)$,

$$(\forall x : \ l_x \leq x \leq h_x \Rightarrow G(x) < \theta) \ \Leftrightarrow \ (G(l_x) < \theta \wedge G(h_x) < \theta).$$

This equivalence means that we can replace a complex formula like the left-hand side that contains quantifiers and implication with a simple conjunction like the one of the right. Similarly, for any monotonically increasing function $H(x)$ (and analogously for the decreasing case):

$$(\forall x : \ l_x \leq x \leq h_x \Rightarrow H(x) < \theta) \ \Leftrightarrow \ H(h_x) < \theta.$$

Notice the structural similarity of the left-hand side to Equation (4). There we could apply the same simplification, slightly generalized to m dimensions, as long as function $F(W) = \sum_{s \in R; s \neq r} \delta_s$ is monotonic or convex. Unfortunately, in general it is neither as the following counter example demonstrates even for $m = 2$. Intuitively, as we sweep the range of a weight w_i from l_i to h_i, the rank of tuple r, which is equal to $F(W)$, may increase and decrease repeatedly. This violates both convexity and monotonicity.

Example 8. *Consider $R(A_1, A_2) = \{(25, 5), (27, 4), (22, 9), (19, 7)\}$, where r is the first tuple. Assume the common Triangle constraint (Section 4.2), i.e., $w_1 + w_2 = 1$. (We can construct similar examples for other constraints on W.) Figure 3 shows the rank of r for different values of w_1. Since r's rank increases, decreases, and then increases again for increasing w_1, the function is neither convex nor monotonic.*

Solution 2: Monotonic core. Our goal here is to find an approximation of $F(W) = \sum_{s \in R; s \neq r} \delta_s$ that ideally delivers almost the same BOX solution, but is monotonic or convex. This is challenging, because small changes of W in any direction inside a box of W-combinations can result in up or down movement of r's rank as the above example demonstrated. Our solution is based on the following crucial insight:

Lemma 9. *Given hyper-rectangle $B = [l_1, h_1] \times \cdots \times [l_m, h_m]$. Let $F_s(W) = f_W(s) - f_W(r)$ and let $C = (c_1, \ldots, c_m)$, where $c_i = h_i$ if*

$s.A_i \geq r.A_i$; *and* $c_i = l_i$ *otherwise. Then*

$$\left(\forall w_1, \ldots, w_m : \bigwedge_{i=1}^{m} l_i \leq w_i \leq h_i \Rightarrow F_s(W) \leq 0\right) \Leftrightarrow F_s(C) \leq 0$$

Proof. To prove the theorem, it is sufficient to show that $F_s(W)$ reaches its maximum in corner point C of hyper-rectangle B. If that maximum value is at most 0, then F_s must be at most 0 in all of B, and vice versa.

To prove that F_s reaches its maximum in C, consider two weight vectors $W_1(w_1, w_2, \ldots, w_m)$ and $W_2 = (v_1, w_2, \ldots, w_m)$ in B that agree in all but the first dimension:

$$\begin{aligned} F_s(W_2) - F_s(W_1) &= v_1 s.A_1 + \sum_{i=2}^{m} w_i s.A_i - v_1 r.A_1 + \sum_{i=2}^{m} w_i r.A_i \\ &\quad - (w_1 s.A_1 + \sum_{i=2}^{m} w_i s.A_i - w_1 r.A_1 + \sum_{i=2}^{m} w_i r.A_i) \\ &= v_1 s.A_1 - v_1 r.A_1 - w_1 s.A_1 + w_1 r.A_1 \\ &= (v_1 - w_1)(s.A_1 - r.A_1) \end{aligned} \tag{6}$$

This implies that $F_s(W_2) > F_s(W_1)$ iff both factors are negative or both are positive. We can derive analogous results for each of the other dimensions.

Let $X = (x_1, \ldots, x_m) \in B$ be the weight vector for which F_s reaches its maximum in B. We show that either $X = C$ or $F_s(C) = F_s(X)$, which each implies that F_s reaches its maximum in corner C. If $C \neq X$, then they must differ in at least one dimension. Without loss of generality, let $c_1 \neq x_1$. We now show by replacing x_1 with c_1 in X, the value of F_s cannot decrease, i.e., $F_s(x_1, x_2, \ldots, x_m) \leq F_s(c_1, x_2, \ldots, x_m)$. From Equation (6) it follows that

$$F_s(c_1, x_2, \ldots, x_m) - F_s(x_1, x_2, \ldots, x_m) = (c_1 - x_1)(s.A_1 - r.A_1).$$

Case 1: $s.A_1 = r.A_1$. This implies $F_s(x_1, x_2, \ldots, x_m) = F_s(c_1, x_2, \ldots, x_m)$.

Case 2: $s.A_1 < r.A_1$. Then by definition $c_1 = l_1$ and hence $c_1 - x_1 \leq 0$. Together, $(c_1 - x_1)(s.A_1 - r.A_1) \geq 0$ and therefore $F_s(x_1, x_2, \ldots, x_m) \leq F_s(c_1, x_2, \ldots, x_m)$.

Case 3: $s.A_1 > r.A_1$. Then by definition $c_1 = h_1$ and hence $c_1 - x_1 \geq 0$. Together, $(c_1 - x_1)(s.A_1 - r.A_1) \geq 0$ and therefore $F_s(x_1, x_2, \ldots, x_m) \leq F_s(c_1, x_2, \ldots, x_m)$.

We apply the same construction to weight vector $(c_1, x_2, \ldots, x_m)$, showing that replacing x_2 with c_2 cannot decrease the value of F_s, then doing the same for replacing x_3 with c_3 and so on. This proves that the maximum of F_s over B is reached in corner C, concluding the proof of the theorem. □

In order to use Lemma 9 for solving BOX, we also need:

Lemma 10. *Given hyper-rectangle* $B = [l_1, h_1] \times \cdots \times [l_m, h_m]$. *If* $|\{s \in R \mid \forall W \in B : f_W(s) - f_W(r) \leq 0\}| \geq n - k$ *then* $\forall W \in B : \rho_W(r) \leq k$.

The lemma states that if there are at least $n - k$ other tuples that do not beat r anywhere in B, then r's rank is k or better. This follows directly from the definition of the rank.

Combining Lemmas 9 and 10, we obtain:

Theorem 11. *Given a top-k query* Q, *the* id *of a tuple* $r \in R$, *and hyper-rectangle* $B = [l_1, h_1] \times \cdots \times [l_m, h_m]$. *Let* $C = (c_1, \ldots, c_m)$, *where* $c_i = h_i$ *if* $s.A_i \geq r.A_i$; *and* $c_i = l_i$ *otherwise. If*

$$|\{s \in R \mid F_s(C) = f_C(s) - f_C(r) \leq 0\}| \geq n - k$$

then B *is an inner box for* Q *and* r.

Theorem 11 enables us to solve BOX using only a conjunction of linear constraints and the indicator variables, removing the nested quantifier and implication. The corresponding optimization problem for box perimeter is:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} (h_i - l_i) \\ \text{s.t.} \quad & l_i \leq h_i, \quad i = 1, \ldots, m \\ & \delta_s = \left(\sum_{i=1}^{m} c_i (r.A_i - s.A_i) \;\geq\; 0\right), \quad s \in R, s \neq r \\ & \sum_{s \in R; s \neq r} \delta_s \geq n - k, \end{aligned} \tag{7}$$

where $c_i = h_i$ if $s.A_i \geq r.A_i$; and $c_i = l_i$ otherwise.

Since Lemma 10 is an implication, not an equivalence, the conditions in Equation (7) are sufficient, but not necessary for enforcing that the corresponding solution be an inner box. In terms of practical implications, this means that the inner box found for Equation (7) may be smaller compared to Equation (5). Our experiments in Section 6.6 indicate that the size difference is small. And since the latter approach is infeasible for all but very small datasets, our proposed approximation Equation (7) currently is the only option for medium-to-large datasets.

3.4 Additional Weight Constraints

In addition to the constraints discussed so far, for all problems we need a predicate $\mathcal{P}$ to enforce desirable properties of the ranking function. First, we include the requirement for all w_i to be non-negative in $\mathcal{P}$. Second, for SAT, BEST, and POINT we also include $\sum_i w_i > 0$ to exclude the undesirable case where all weights are zero and hence all tuples have the same score of zero. For BOX, we instead require $\sum_i h_i > 0$.

Challenge: Preventing an unbounded solution space. Multiplying each w_i with the same constant c scales the score of each R-tuple by the same factor c and hence does not change the ranking.

Solution: Upper bounds on the weights. The issue is easily addressed by including in $\mathcal{P}$ linear constraints upper-bounding the w_i (or analogously the h_i for BOX). These are discussed in Section 4.2.

The predicate also enables the user to define constraints like:

- Search for the largest inner box in a specific region of the weight-vector space.
- Consider only scoring functions that place a particularly high (or low) weight on some attribute of interest, e.g., because of some natural notion of importance dictated by the application.
- Ensure that the BOX solution has a certain minimal "thickness" in a dimension.
- After returning a POINT find the maximal inner box that contains it.

For all constraints on W discussed here, the corresponding predicate is a conjunction of linear constraints.

4 ALGORITHM IMPLEMENTATION

As discussed above, solving the problems introduced in Section 2 requires solving the satisfiability and optimization problems in Equations (3), (5) and (7), respectively, possibly with additional linear constraints $\mathcal{P}$ on the weights. We first discuss the appropriate state-of-the-art solvers for these types of problems and then cover specific implementation aspects.

4.1 Solver Summary

We selected Gurobi [19] and Z3 [12] as they use different underlying technology and have different capabilities. Gurobi is a state-of-the-art, commercial mathematical programming solver that can handle MILP (Mixed Integer Linear Programming) problems. It can solve satisfiability queries and, when provided with objective functions, it can solve optimization versions of such problems. Gurobi has some of the most advanced algorithms of any similar tool, using standard techniques such as cutting planes and symmetry breaking, as well as various heuristics and the ability to run in parallel.

Z3 is a state-of-the-art SMT (Satisfiability Modulo Theories) solver. SMT solvers are used to solve problems in decidable fragments of first order logic that include multiple theories, such as linear arithmetic, uninterpreted functions and strings. SMT solvers use a Boolean SAT (Satisfiability) solver to orchestrate interaction between decision procedures for the theories they support. SMT solvers are now widely used in the context of formal methods, software engineering, security and programming languages. Z3 also provides support for quantifiers, which allows for a direct encoding for BOX in Equation (5). The use of quantifiers leads to undecidable fragments of logic, therefore, Z3's support for such problems is heuristic.

For our problems, Gurobi tends to be the better choice, typically outperforming Z3's execution time by an order of magnitude, or more. This can be explained by the numeric nature of our problems: Gurobi is designed to handle numeric constraints and that is the most natural way to encode our problems. When quantifiers and implications are used in our constraints, Z3 can only handle relatively small datasets.

4.2 Default Weight Constraints

To enforce non-negative weights, we add to Equation (3) constraints

$$w_i \geq 0, \quad i = 1, \ldots, m$$
$$\sum_{i=1}^{m} w_i > 0 \quad \text{(omitted for TRIANGLE)}$$

and to Equations (5) and (7)

$$l_i \geq 0, \quad i = 1, \ldots, m$$
$$\sum_{i=1}^{m} h_i > 0 \quad \text{(omitted for TRIANGLE)}$$

In addition, we also add to Equation (3) one of the following 2 constraints:

$$\text{TRIANGLE:} \quad w_m = 1 - \textstyle\sum_{i=1}^{m-1} w_i \ \wedge \ \sum_{i=1}^{m-1} w_i \leq 1 \tag{8}$$
$$\text{CUBE:} \quad w_i \leq 1, \quad i = 1, \ldots, m \tag{9}$$

Each is a linear constraint that prevents an unbounded solution space (Section 3.4); their names are inspired by the geometric shape of the resulting constrained space of possible W-values for $m = 3$. In the linear program for TRIANGLE, we do not include the left term of the conjunction, but instead replace all occurrences of w_m with $1 - \sum_{i=1}^{m-1} w_i$. TRIANGLE enforces $\sum_{i=1}^{m} w_i = 1$, which is used by all previous work on reverse top-k and why-not questions for top-k (see Section 7). CUBE directly bounds the solution space to desired ranges for each weight dimension. Different from TRIANGLE, where w_m is eliminated from the program, it treats all weight dimensions symmetrically. For Equations (5) and (7), the TRIANGLE or CUBE constraint is analogous, but uses h_i instead of w_i.

EXAMPLE 12. *In Figure 2, the TRIANGLE constraint limits the solution space to the large triangle. The CUBE constraint limits the solution space to the cube with range* $[0, 1]$ *in each dimension.*

4.3 Program Properties

The program for Equation (3) has m variables $w_1, \ldots, w_m$ and $n-1$ indicator variables δ_s. Each indicator is defined via a linear constraint. There is an additional linear constraint on the sum of the indicator variables. And $\mathcal{P}$ has $m+1$ (TRIANGLE) or $2m$ (CUBE) additional linear constraints limiting the range of the w_i. Hence in total, the number of variables and constraints is $O(m+n)$.

Similarly, the program for Equation (7) has $2m$ variables for the inner-box coordinates $l_1, \ldots, l_m, h_1, \ldots, h_m$ and $n-1$ indicator variables. There are m linear constraints for the box coordinates, $n-1$ linear constraints defining the indicator variables, 1 linear constraint for the indicator sum, and the $m+1$ or $2m$ linear constraints in $\mathcal{P}$. Hence the number of variables and constraints is $O(m+n)$.

The direct BOX encoding in Equation (5) has $3m$ variables for the m-dimensional points $w_1, \ldots, w_m$, the inner-box coordinates $l_1, \ldots, l_m, h_1, \ldots, h_m$ and $m+n-1$ indicator variables including m more indicators of whether $w_1, \ldots, w_m$ are in the range of $[l_1, h_1], \ldots, [l_m, h_m]$. There are m linear constraints for the box coordinates, $m+n-1$ linear constraints defining the indicator variables, 1 linear constraint for the indicator sum, the $m+1$ or $2m$ linear constraints in $\mathcal{P}$, and one more constraint for the quantifier including implication. The number of variables and constraints is also linear in m and n. However, it contains the complex one with quantification and implication, which dramatically slows down the solver compared to the linear-constraint-only programs.

4.4 Generality of the Approach

While more general ranking functions exist, ranking by a linear combination of individual scoring attributes or features, using non-negative weights, is powerful and widely used [2, 3, 9, 11, 17, 21, 22, 24, 28, 32, 41–47, 50]. There are good reasons for this.

First, in many ranking problems one has to combine multiple separate scoring features. For example, when buying a used car, one ideally would like to minimize scoring features such as age,

mileage, and price of the car. For multi-objective optimization problems like this, linear scalarization is arguably the default approach, especially when one needs a total order of competing solutions. Second, in a linear function like Equation (1), the weights directly reveal the importance assigned to each individual scoring attribute. This makes it easier to analyze properties of the ranking function, compared to a complex blackbox model that outputs a single score. Third, the linear ranking function is much more powerful than it may seem, because we can incorporate more complex functions as features. In the used-car example, we can add a data column that contains for each car the expected repair cost predicted by some non-linear AI model. Then the linear ranking function controls the impact between given (age, mileage, price) and derived (repair cost) scoring features all together. Similarly, for someone applying to college, one can add a score indicating predicted interest in different majors based on a model that analyzes the applicant's essay [48].

Our solution could directly handle any real-valued weights, not only non-negative ones. However, in agreement with previous work, we have not found a need for this. When combining multiple scores into a single one used for ranking, one may want to minimize some, e.g., car price, while maximizing others, e.g., expected resale value. In a maximization problem, instead of assigning a negative weight to an attribute like car price, one can equivalently work with the negative car price and a non-negative weight.

In theory, our approach also supports any general ranking function over the attributes of input R. The main difference is the structure of the constraints that encode the ranking order. If the ranking function is not linear, the constraints can become more complex. While some solvers can handle non-linear constraints, running time and even decidability of the problem may be affected. The construction of the monotonic core would also have to be revisited. We intend to explore suitable generalizations of the ranking function in future work.

5 SCALABILITY AND EXTENSIONS

In why-not-yet, the number of constraints, which significantly impacts solver performance is determined by the cardinality of the dataset $n = |R|$. For improved scalability, we propose techniques that reduce the number of constraints and/or enable the user to control the tradeoff between running time and result quality.

5.1 Removing Dominators and Dominatees

We can reduce in time $O(n)$ the number of constraints without impacting the why-not-yet solutions by removing all *dominators* and *dominatees*. A dominator $s \in R$ is a tuple whose values of the ranking attributes $A_1, \ldots, A_m$ are all greater than or equal to those of r, with at least one of them being strictly greater. Hence s will always beat r. (Recall that all w_i are non-negative.) Similarly, a dominatee is an R-tuple whose values of the ranking attributes are all less than or equal to those of r. A dominatee can never have a higher score than r and hence does not affect r's rank. We refer to the remaining R-tuples as *competitors*.

The number of dominators is only a *lower bound* for the best rank r could reach. Even though r could beat each competitor *individually* for the right choice of ranking function, there may be no ranking function where r beats all or even most of them. Hence solving SAT and BEST is not trivial.

Example 13. *Consider relation R with (A_1, A_2)-pairs $(2,2)$, $(1,4)$, and $(4,1)$. Even though $(2,2)$ is not dominated by any of the other 2 tuples, it can never reach rank 1: $2w_1 + 2w_2 \geq w_1 + 4w_2 \wedge 2w_1 + 2w_2 \geq 4w_1 + w_2$ implies $4(w_1 + w_2) \geq 5(w_1 + w_2)$, which is only satisfied for $w_1 = w_2 = 0$.*

5.2 Binary Search for BOX

BOX is more general than SAT, BEST, and POINT and therefore, as our experiments show, takes the solvers significantly longer, even with the monotonic approximation Equation (7). We propose an approach that lets the user control the tradeoff between running time and result quality, i.e., box size. The main idea is to replace the optimization problem (maximize inner-box perimeter) with an easier decision procedure (does an inner box of perimeter p exist). We show the resulting constraints, with default constraints for Equation (7) (it is analogous for Equation (5)):

$$\begin{aligned}
&\sum_{i=1}^{m} (h_i - l_i) \geq p \\
&0 \leq l_i \leq h_i, \quad i = 1, \ldots, m \\
&\sum_{i=1}^{m} h_i > 0 \\
&\delta_s = \left(\sum_{i=1}^{m} c_i (s.A_i - r.A_i) \;\leq\; 0 \right), \quad s \in R, s \neq r \\
&\sum_{s \in R; s \neq r} \delta_s \geq n - k
\end{aligned} \tag{10}$$

where $c_i = h_i$ if $s.A_i \geq r.A_i$; and $c_i = l_i$ otherwise.

First, we check satisfiability for $p = 0$, which is equivalent to solving POINT. If it is unsatisfiable, then no inner box exists. Else, we check satisfiability for the greatest possible value of p, i.e., $p = m$ for Cube and $p = 1$ for Triangle. If it is satisfiable, then the corresponding inner box is maximal and returned immediately. Otherwise we perform a binary search using Algorithm 1 to find the largest p for which Equation (10) is satisfiable, returning the corresponding inner box found.

The user has 2 ways of controlling the time-vs-quality tradeoff. First, they can set threshold t_{total} or a threshold on the precision $H - L$ of the convergence to stop the binary search early, forcing the algorithm to return the largest inner box B found so far. Second, they can set timeout t_{decision} for the decision procedure, i.e., the time it takes to solve Equation (10). A timeout is treated like an UNSATISFIABLE response, meaning the binary search continues with smaller perimeter candidates. Lower timeouts reduce running time at the cost of possibly ending up with an inner box of smaller perimeter. (The approach for volume is analogous.) Based on our experiments, We suggest setting the convergence threshold to 0.01, t_{decision} to 1 min and t_{total} to 10 mins. From here, the user can explore larger time thresholds if the results are returned quickly, and vice versa.

Algorithm 1: Binary search for BOX

Input: Satisfiability problem M (Equation (10)); upper bound H ($H = m$ for CUBE; $H = 1$ for TRIANGLE) and lower bound $L = 0$ for box perimeter
Output: inner box

```
perimeter p = (H + L)/2
repeat
    Set p in M
    M.decide() with timeout t_decision
    if M.status == SATISFIABLE then
        B = M.getBox()
        L = p
    else
        H = p
    p = (H + L)/2
until convergence or timeout t_total
Return inner box B
```

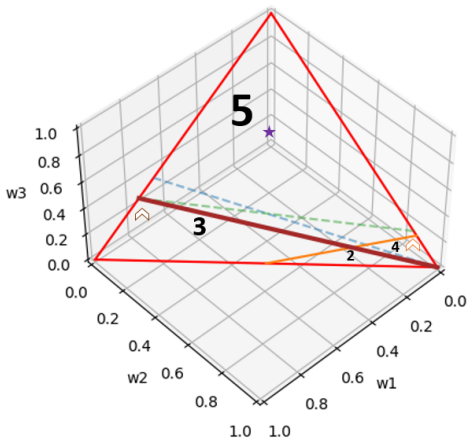


Figure 4: Example 14: Solution space after clustering s_2, s_3. The bold line shows the constraint for the cluster; the dotted lines show the original constraints for s_2, s_3.

5.3 Clustering

The binary search for BOX still solves a SAT problem in each iteration, which may take too long when the dataset is very large. Similarly, large data may result in a slow response for SAT, POINT, and BEST. We now discuss a data-reduction process that can speed up all our techniques, while providing an approximate solution. This solution provably never produces false positives, meaning SAT will never incorrectly claim satisfiability, BEST will never return a rank better than the true answer, and all answers returned by POINT and BOX will be valid weight combinations that achieve the desired ranking outcome. To achieve this guarantee, we propose the following construction.

The main idea is to partition R into z subsets, which we call *clusters*, and then replace each subset by a single *cluster representative*, similar to previous work like [6]. Performance improves because the solver deals with z instead of $n-1$ indicator variables. We need to address 2 challenges specific to our problem: (1) Introduce indicator variables and constraints for clusters and (2) choose an appropriate partitioning of R into z clusters. For simplicity, we explain our approach using Example 6.

Modified indicator constraints. Assume s_2 and s_3 in Example 6 form a cluster. Let the cluster representative be formed by taking the maximum value for each ranking attribute, i.e., for $s_2 = (1, 1, 14)$ and $s_3 = (0, 2, 14)$, this is $\varrho = (1, 2, 14)$. In Equation (3), we remove the 2 indicators $\delta_{s_2}, \delta_{s_3}$ and replace them with cluster indicator

$$\delta_\varrho = \left(\sum_{i=1}^{m} w_i(r.A_i - \varrho.A_i) \ < \ 0 \right).$$

We also replace indicator constraint $\delta_{s_1} + \delta_{s_2} + \delta_{s_3} < k$ with $\delta_{s_1} + 2\delta_\varrho < k$. It captures the fact that if representative ϱ beats r, then in the worst case both cluster members may beat r. The approximation is caused by the fact that while $f_W(r) \geq f_W(\varrho)$ implies $f_W(r) \geq f_W(s_2) \wedge f_W(r) \geq f_W(s_3)$, the reverse is not true. This means that the modified constraints are stricter than the original ones.

EXAMPLE 14. *Figure 4 shows how the solution space of Figure 2 changes after clustering s_2 and s_3. Their indicators are replaced by the indicator of their representative $\delta_\varrho = 2w_1 - 6w_3 < 0$. Since the indicator represents 2 tuples, its weight is 2, i.e., crossing the line changes r's rank by 2 positions.*

After partitioning R into z clusters of sizes $c_1, \ldots, c_z$, there are z indicator variables $\delta_1, \ldots, \delta_z$ for the corresponding cluster representatives, which are obtained by taking for each ranking attribute the maximum over the tuples in the cluster. The indicator constraint is $\sum_{i=1}^{z} c_i\delta_i < k$, where the *indicator weights* c_i are the cluster sizes. This construction applies analogously to all linear programs discussed above.

Cluster finding. Our construction is equivalent to replacing each indicator constraint $\sum_{i=1}^{m} w_i(r.A_i - s.A_i) < 0$ by $\sum_{i=1}^{m} w_i(r.A_i - \varrho.A_i) < 0$ for the corresponding cluster representative ϱ. This introduces error $w_i(\varrho.A_i - s.A_i)$ in dimension i. To minimize this error, we want the cluster representative to be as similar as possible to all cluster members. By construction of the representative, this is equivalent to requiring all cluster members to have small pairwise differences for all ranking attributes. This aligns with the design goal of popular clustering techniques like k-means and hierarchical clustering [20]. Our implementation uses the k-means algorithm [1] and Euclidean distance.

5.4 Multiple Expected Tuples

Our technique can be easily extended to why-not-yet problems over a set $\{r_1, r_2, \ldots\}$ of expected tuples. Different from previous work [21] that requires the same value of k for each r_i, we support a different k_i for each r_i. Hence the user now can ask for a ranking function that puts r_1 in the top-k_1, while at the same time putting r_2 in the top-k_2 etc. The idea is to create an instance of Equation (3) (and analogously for the other problems) for each r_i and combine all these constraints into a single program. Indicator names must be chosen so that for each pair $r_i, r_j, i \neq j$, their sets of indicator names are disjoint.

6 EXPERIMENTS

We now demonstrate that our techniques are practical and compare them against related work. For our approximation techniques, we quantify the tradeoff between running time and result quality.

6.1 Experimental Setup

The default ranking function f_{W_0} assigns the same weight to each ranking attribute. The *original rank* of a tuple r refers to its rank $\rho_{W_0}(r)$ according to this default function.

Environment and Implementation. All experiments are executed on an Ubuntu 20 Linux server with an Intel Xeon E5-2643 CPU and 128GB RAM. We implemented our technique in Java, testing it on both Java 11 and Java 18. To solve the satisfiability and optimization problems, we utilize the Java libraries of the leading commercial optimizer Gurobi [19]. We also use Z3 [12], a SOA open-source theorem prover to implement the direct encoding of BOX with quantifiers and implication (Equation (5)). For Gurobi and Z3 we used their default configurations, which meant that the former utilized 8 cores with up to 16 threads, while the latter ran on 1 core. To find clusters (Section 5.3), we use the Weka [15] k-means algorithm [1].

Competitors. No previous work exists for BOX. For SAT, BEST, and POINT there is no direct solution either, but we can adopt the why-not algorithm for top-k by He and Lo [21]. It cleverly samples weight vectors and applies pruning techniques to minimize top-k query computations. The algorithm can be tuned to balance improving a tuple's rank vs minimizing weight changes relative to a given weight vector. Since we are interested in the former, we set $k = 1$ and set the tuning parameter to not penalize weight modifications. (This way the algorithm tries to get expected tuple r as high in the ranking as possible.) We refer to this algorithm as *Sampling*, using the original C++ code shared by the authors. We also adopt the arrangement tree algorithm [3] for SAT, BEST, and POINT. It combines sampling with an exploration of partitions defined by hyper-planes that separate the half-space where a tuple beats another, from the half-space where it does not. Since the algorithm is designed to achieve a ranking where a certain number of members of a given group appears in the top-k, we can consider all expected tuples as members of the target group. We refer to this algorithm as *Tree*, and implemented it as described in the original paper. (For all calls to an ILP solver, we use Gurobi so that times are comparable.)

Datasets. Like previous work on related problems, e.g., [17, 21, 41], we use the latest version of the real NBA dataset, as well as synthetic datasets of different distributions.

The NBA dataset [34] contains 22467 tuples with statistics of all NBA players from seasons 1979/80 to 2021/22. Each tuple represents a *player-season combination*—uniquely identified by the PLR attribute, which consists of player name, age and team. As introduced in Example 1, the default ranking attributes are the player's average statistics during that season: PTS, REB, AST, STL, and BLK.

The synthetic datasets—*uniform*, *correlated*, and *anti-correlated*—allow us to explore the impact of correlations between the ranking attributes. In the uniform data, values for each ranking attribute are generated uniformly at random, and independent of the other attributes. In the correlated dataset, a tuple with a high (low) value in one ranking attribute is likely to also have high (low) values for the others. In the anti-correlated dataset, a tuple with a high (low) value in one ranking attribute is likely to also have high (low) values for half of the other attributes, but more likely to receive low (high) values for the other half. This pattern of generating synthetic data of different distributions dates back to [5].

6.2 Case Study: Why has Luka Dončić not shown up in the top-10 yet?

We study an NBA player who is widely perceived as a superstar, but who does not appear in the top-10 of all time when using equal default weights of 0.2 for each of the 5 ranking attributes. Our technique returns SATISFIABLE in 596 msec. The Triangle and Cube answers are $(w_1, w_2, w_3, w_4) \in [0.24, 0.3] \times [0.17, 0.17] \times [0.34, 0.34] \times [0.0, 0.18]$, and $(w_1, w_2, w_3, w_4, w_5) \in [0.54, 0.54] \times [0.94, 0.94] \times [1.0, 1.0] \times [0.0, 1.0] \times [0.0, 0.2]$ in 10708 msec and 6598 msec, respectively. Both answers convey similar information: to get Dončić into the top-10, we need higher weights on PTS, REB and AST, but lower weights on STL and BLK. The Cube answer is more interesting, indicating that with very high weights on the first three, one can pick any weight on STL. The answers match the perception that Dončić is a versatile player in terms of offense (points, rebounds, assists), but not a top defender (steals, blocks).

6.3 Performance on BEST

On SAT and POINT, our approach generally takes less than a second to respond, while *Sampling* and *Tree* are very fast when a large fraction of the weight-combination space is satisfiable, but extremely slow (several hours) when the satisfiable region is very small. Hence we present results for BEST, where our approach takes on average 6 sec to solve the instances discussed below. Since the competitors produce continually improving answers as they are given more time, we report the best ranking they find within 6 sec and 60 sec, i.e., giving them 10x more time. For *Sampling*, this corresponds to taking 100,000 and 1 million samples, respectively.

Figure 5 reports the best ranks found for all NBA data's player-season tuples originally ranked at odd positions between 20 and 50. *Sampling* finds the correct answer only in 40% and 60% of the cases, respectively; while *Tree* is correct in 26.67% and 40% of the cases. Note that in contrast to our approach, even when they find a sample for which the best rank is achieved, the competitors do not know if a better solution may exist. *Sampling* generally has no way to determine when it may have found the true answer to BEST. In contrast, since *Tree* combines sampling with systematic exploration of the weight-combination space, it can in theory determine the exact answer if it explores all nodes of the arrangement tree. For the datasets in our experiments, this was generally not feasible, resulting in hours or days of running time.

Why is it hard for sampling-based approaches to find the best possible rank? Sampling works well when the probability of finding the correct answer is sufficiently high. Otherwise success probability is low, forcing a sampling-based algorithm to continue exploring. While *Tree* partially addresses the issue by systematically exploring the space, low success probability still forces it to explore a large fraction of the huge arrangement tree, which has more than 250 million nodes. Since it solves a linear program at each node,

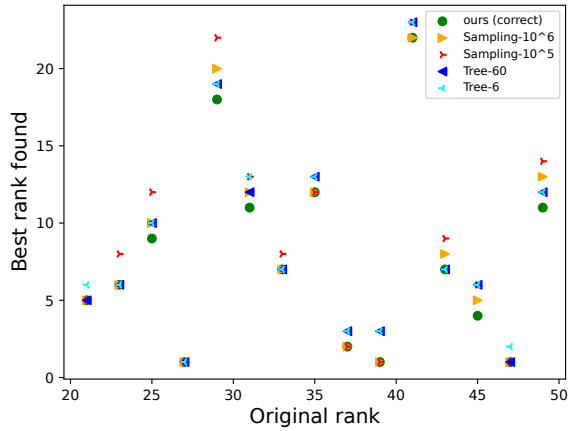
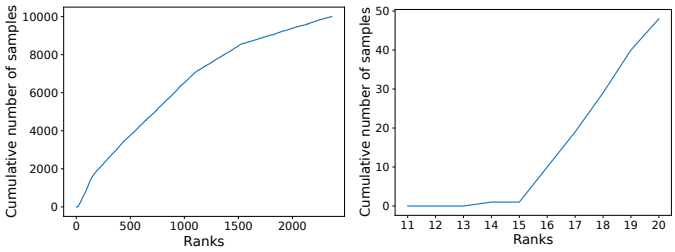


Figure 5: Performance on BEST. Lower rank found is better. Only our approach guarantees to find the correct answer.



(a) All samples **(b) Samples with top-20 result**

Figure 6: Rank distribution for 10,000 samples for a tuple originally ranked 31st, whose highest possible rank is 11.

running time exceeds 2 days when traversing the entire tree, even when applying their pruning techniques. Figure 6 illustrates the sampling success rate for the tuple originally ranked 31st, whose solution for BEST is 11. Out of 10,000 samples, only 48 rank among the top-20, and only 1 sample among the top-15 (Figure 6b).

6.4 Performance on BOX: Monotonic Core

Since no previous approach can solve BOX, we explore the performance of our main approach (Equation (7)), then compare it to the other versions and extensions. We also compare running time to SAT, which is easier than BOX.

On the NBA data, we vary the expected rank (k), the original rank of the expected player r ($\rho_{W_0}(r)$), the number of attributes (m) and the number of the expected tuples ($|\overrightarrow{r}|$). Table 2 shows the parameter settings; the default ones are bold. By default, we select the tuple with original rank $k+10$ as the expected tuple, i.e., we explore how this tuple could move up 10 places. Due to the use of heuristics whose effectiveness depends on the specific program instance, running time of solvers like Gurobi and Z3 can vary widely, even for seemingly similar inputs. Hence for every combination of k, $\rho_{W_0}(r)$, m and $|\overrightarrow{r}|$, here and in Sections 6.5 and 6.6, we conduct 5 runs—for the tuples at original ranks $\rho_{W_0}(r)$, $\rho_{W_0}(r)+1$, $\rho_{W_0}(r)+2$, $\rho_{W_0}(r)+3$, and $\rho_{W_0}(r)+4$—and take the median time of all satisfiable ones. In each experiment, we use Gurobi.

Table 2: Parameter settings

Parameter	Ranges
k	10, 20, 30, 40, **50**, 60, 70, 80, 90, 100
$\rho_{W_0}(r)$	$[k+1, k+21]$ (default is k + **10**)
m	2, **3**, 4, 5
$\lvert\overrightarrow{r}\rvert$	**1**, 2, 3, 4, 5

Varying k (Figure 7a). As k increases, execution time increases because the solver must consider a larger space of settings for the indicator variables. Execution time is acceptable even for large k.

Varying $\rho_{W_0}(r)$ (Figure 7b). There is no clear trend as tuples at different original ranks are selected as the expected tuple r. This is reasonable, because it depends on the specific tuple values how hard it is to determine weights to move it up 10 positions.

Varying m (Figure 7c). As the number of ranking attributes and hence program variables increases, so does running time. The impact is negligible for the SAT problem, whose satisfiability problem is easier than the optimization problem for BOX.

Varying $|\overrightarrow{r}|$ (Figure 7d). As we increase the number of the expected tuples, execution time increases approximately linearly. When there are 5 expected tuples, the problem is unsatisfiable, hence no time is recorded for BOX. (For SAT, the time reported is until UNSATISFIABLE is returned.)

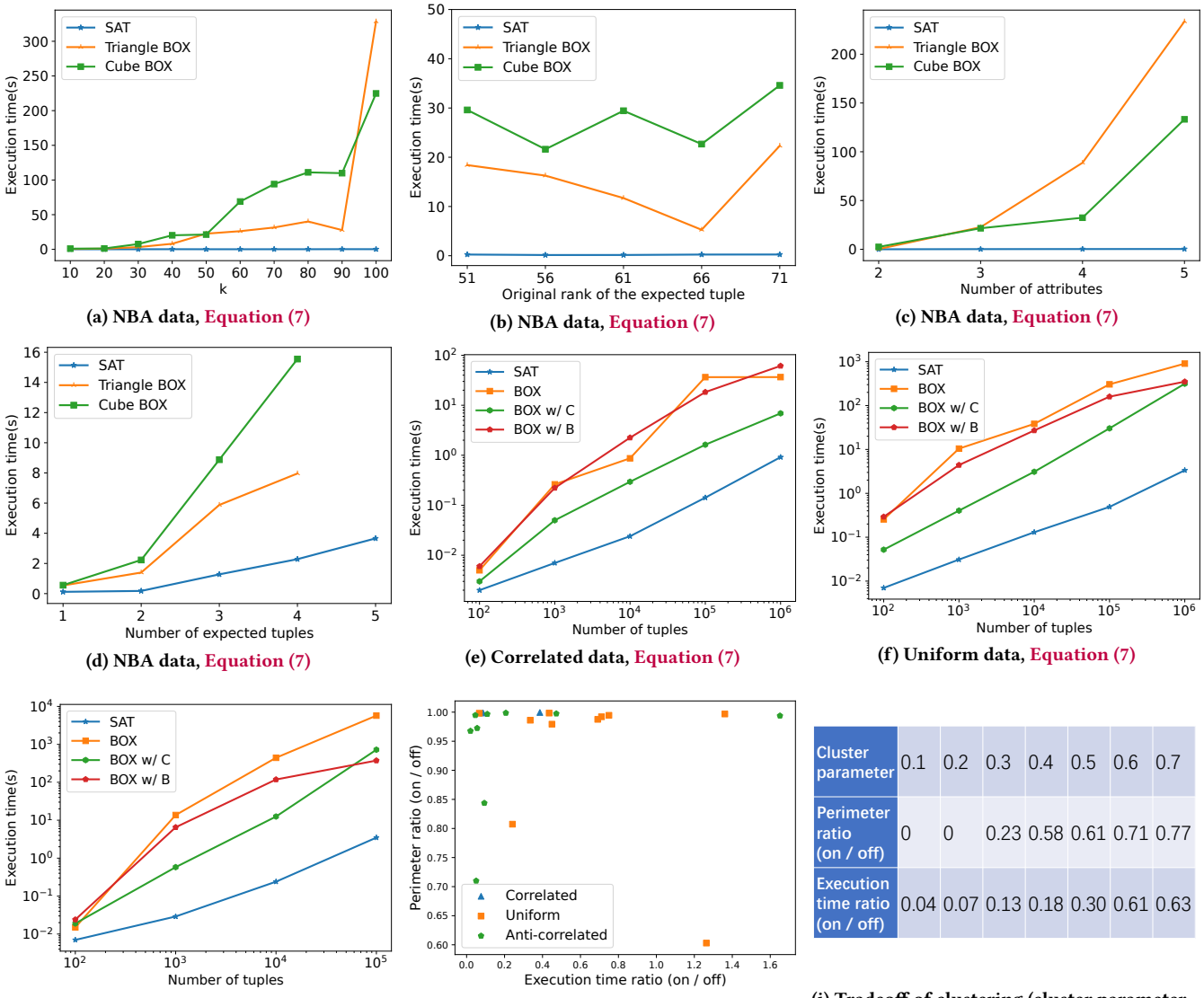
6.5 BOX: Scalability

We use the synthetic data to explore the effectiveness of the techniques introduced in Section 5. Here we set $k = 50$, $\rho_{W_0}(r) = 51$, $m = 3$, $|\overrightarrow{r}| = 1$, and the weight constraint to be Cube, since it generally is the slowest. After removing all dominators and dominatees, about 10%, 5%, and 90% of the tuples remain as competitors for the uniform, correlated, and anti-correlated distribution, respectively. Therefore, for the same data size, running time on the anti-correlated distribution is generally higher.

Varying n (Figures 7e to 7g). By default, we set the cluster number to be half of the competitor number, the binary-search timeout for each iteration to 1 min and the convergence threshold of Algorithm 1 to 0.01. Note the log scale on both axes. Both techniques reduce running time and can achieve more speedup for more aggressive parameter settings (fewer clusters, shorter timeout).

Tradeoff of binary search (Figure 7h). We explore the tradeoff between approximation quality and running time of binary search. Each data point represents a run from Figures 7e to 7g whose running time was greater than 1 min. Most points gather in the top left corner, demonstrating a good speed-up with little loss in quality.

Tradeoff of clustering (Figure 7i). We explore the impact of cluster number on inner-box perimeter and execution time. We use the uniform dataset for $n = 1,000,000$ with the other parameters as mentioned above. As cluster size is reduced, strong performance gains can be realized with fairly little impact on box size (until about 0.3). Then box size drops more rapidly, but the tradeoff may still be worth it, because a smaller box obtained in seconds is often better than no response for minutes or hours.



(a) NBA data, Equation (7)

(b) NBA data, Equation (7)

(c) NBA data, Equation (7)

(d) NBA data, Equation (7)

(e) Correlated data, Equation (7)

(f) Uniform data, Equation (7)

(g) Anti-correlated data, Equation (7)

(h) Tradeoff of binary search

Cluster parameter	0.1	0.2	0.3	0.4	0.5	0.6	0.7
Perimeter ratio (on / off)	0	0	0.23	0.58	0.61	0.71	0.77
Execution time ratio (on / off)	0.04	0.07	0.13	0.18	0.30	0.61	0.63

(i) Tradeoff of clustering (cluster parameter = cluster number / competitor number)

Figure 7: Performance on BOX. C indicates use of clustering; B binary search (Section 5).

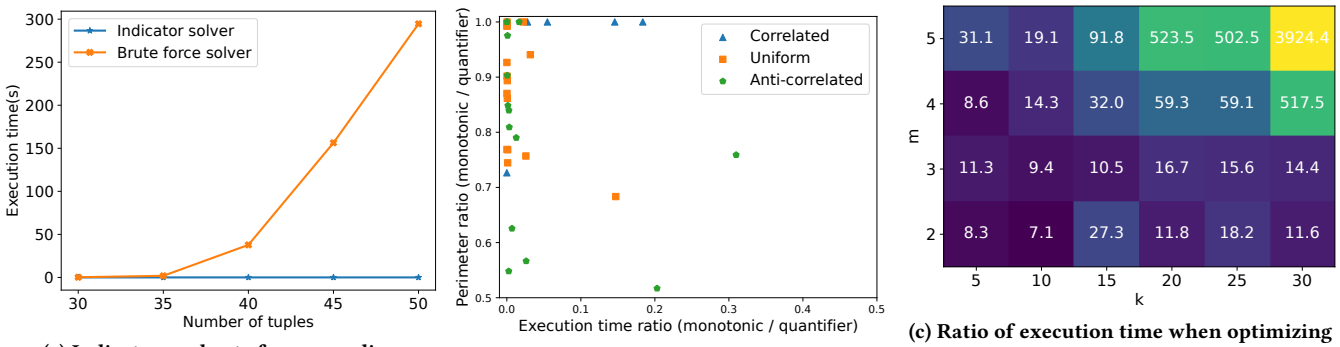
6.6 Impact of Other Design Choices

Why not use the brute force approach for SAT? We demonstrate the importance of using indicator variables (Section 3.1). In Figure 8a, we show representative results for uniform data when using Gurobi, setting $k = 10$, $\rho_{W_0}(r) = 11$, $m = 3$, $|\overrightarrow{r}| = 1$ and varying n from 30 to 50. Even for these tiny datasets, execution time of the brute force approach rises rapidly due to the combinatorial number of problem instances explored.

Why not use the direct BOX encoding with quantifiers, disjunction, and negation (Equation (5))? Both Gurobi and Z3 do not support optimization with quantified constraints, but we were able to implement Equation (5) in the Z3 theorem prover using our binary-search procedure (Section 5.2). We present results for $k \in \{10, 20, 30\}$ and set $\rho_{W_0}(r) = k + 1$, $n = 50$, $m = 3$, $|\overrightarrow{r}| = 1$, weight constraint to Cube, using 3 synthetic datasets. In Figure 8b, each point represents a run that returns a valid inner box.[2] We can see that our monotonic core approximates the direct-encoding solution well, with much faster execution time. Note also that the dataset here is **extremely small**. Direct encoding for $n > 100$ tuples resulted in solver timeouts and runtime exceptions.

Why optimize for box perimeter instead of volume? While our approach works for both volume and perimeter maximization, the objective function for perimeter is linear; for volume it is a polynomial of degree m, which slows down the solvers. Figure 8c presents a heatmap for the ratio of the execution time for volume vs perimeter optimization. We use synthetic uniform data, set $n = 100$,

[2]Sometimes the perimeter ratio can slightly exceed 1.0 because the solution of direct encoding is implemented in an approximate way through binary search. In that case, we set the parameter ratio to 1.0, giving an advantage to direct encoding.



(a) Indicators vs brute force encoding

(b) Monotonic core vs direct encoding

(c) Ratio of execution time when optimizing for inner-box volume vs perimeter

Figure 8: Experimental results for Section 6.6

$\rho_{W_0}(r) = k+1$, $|\overrightarrow{r}| = 1$, weight constraint to CUBE, and vary k and m. When k and m increase, the ratio increases rapidly. Note again the *extremely small* dataset. When solving the TRIANGLE BOX of the first case in Section 6.2, optimizing perimeter only takes about 3 sec, while volume takes more than 1 hour.

7 RELATED WORK

Why not? Chapman and Jagadish [7] introduce "why-not" as the problem of identifying the relational operator that is responsible for an expected tuple to be missing from the query output. Later research focused on formalizing why-not provenance and generalizing the queries supported [4], as well as automatically generating a refined query whose output includes the expected tuples [40].

Why-not on top-k queries. In a series of articles, He, Lo et al introduce why-not problems for top-k queries [21, 22, 47], which return a refined top-k query that includes the missing tuples. They explore tradeoffs between *increasing* k and minimal changes to the weight vector W. Our why-not-yet problems are different: we are interested in finding weight vectors that *improve* the ranking of expected output tuples, not increase k. Nevertheless, as we discuss in Section 6, it is possible to extend their approach into a solution for our problem. They rely on clever sampling and pruning strategies, making this the best known sampling-based solution for our problem. A similar sampling strategy was also proposed in the context of the "modifying W and k problem" for reverse top-k queries [17, 32]. As our experiments show, sampling is inherently limited when using it for SAT, BEST, and POINT; and it cannot solve BOX. Other related work on why-not on top-k queries explored specialized solutions in the context of applications that manage keyword queries and 2-dimensional spatial coordinates [8–10, 51].

Reverse top-k queries. Vlachou et al. [41, 42] introduce reverse top-k queries of 2 types. The monochromatic type finds *all* W where a query tuple ranks among the top-k, while in the bichromatic type the weight vectors must be chosen from a given set of candidates. Follow-up work explores variations and extensions [11, 17, 28, 32, 33, 43–45, 50]. In our why-not-yet problems, no candidate set for W is given, therefore solutions for the bichromatic type cannot be used. (They focus on efficiently eliminating given candidates, while in our problems the candidates have to be found.) For the monochromatic type, an exact solution only exists for $m = 2$ where $w_2 = 1 - w_1$. For larger m, previous work acknowledges the hardness of finding an exact solution [42] and hence resorts to sampling-based approaches [17, 32].

Fair score-based ranking. [48] survey the SOA for fair score-based ranking. The most related is work by Asudeh et al [3], which aims to modify a linear ranking function to ensure sufficient representation of groups in the top-k. We experimentally compare to this approach when adapted to our problem. A similar method is used in [2] to find stable rankings.

Other query explanation work. Our work fits into the general context of database usability [27] and reverse data management [36]. Some approaches there explain missing query answers by proposing data modifications [23, 25]; which is related to how-to queries that change input to get a desired output [37]. The most recent work in this space includes extensions to explain missing answers over nested data [13, 14] and the use of diagrams, examples, query graphs, conditional instances, respectively, to help users understand queries [18, 29–31, 38]. There is also renewed interest in studying the connection between causality and explanations [16, 35, 39].

8 CONCLUSION

We propose the first general *exact* solution for problems SAT, BEST, and POINT. Adopting sampling approaches from related work can only provide approximate answers or results in infeasible running time for BEST. In general, sampling becomes ineffective when only a small fraction of the space of possible weight vectors ranks the expected tuples among the top-k. For BOX, we propose the first known solution. To make it practical and scalable, we propose the notion of a monotonic core. Our clustering approach enables the user to improve running time for all problems as desired by controlling the number of clusters, with moderate loss in result quality even for large data.

Interesting avenues for future work are computing a compact description of the entire set of weight vectors that rank the expected tuples among the top-k and generalizing the approach to noisy and unreliable data.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation (NSF) under award number IIS-1956096.

REFERENCES

[1] D. Arthur and S. Vassilvitskii. 2007. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. 1027–1035. http://dl.acm.org/citation.cfm?id=1283383.1283494

[2] Abolfazl Asudeh, H. V. Jagadish, Gerome Miklau, and Julia Stoyanovich. 2018. On Obtaining Stable Rankings. *Proc. VLDB Endow.* 12, 3 (2018), 237–250. https://doi.org/10.14778/3291264.3291269

[3] Abolfazl Asudeh, H. V. Jagadish, Julia Stoyanovich, and Gautam Das. 2019. Designing Fair Ranking Schemes. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD 2019*. 1259–1276. https://doi.org/10.1145/3299869.3300079

[4] Nicole Bidoit, Melanie Herschel, and Katerina Tzompanaki. 2014. Query-Based Why-Not Provenance with NedExplain. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014*. 145–156. https://doi.org/10.5441/002/edbt.2014.14

[5] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline Operator. In *Proceedings 17th International Conference on Data Engineering, ICDE 2001*. 421–430. https://doi.org/10.1109/ICDE.2001.914855

[6] Matteo Brucato, Juan Felipe Beltran, Azza Abouzied, and Alexandra Meliou. 2016. Scalable Package Queries in Relational Database Systems. *Proc. VLDB Endow.* 9, 7 (2016), 576–587. https://doi.org/10.14778/2904483.2904489

[7] Adriane Chapman and H. V. Jagadish. 2009. Why not?. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, SIGMOD 2009*. 523–534. https://doi.org/10.1145/1559845.1559901

[8] Lei Chen, Yafei Li, Jianliang Xu, and Christian S. Jensen. 2017. Direction-Aware Why-Not Spatial Keyword Top-k Queries. In *2017 IEEE 33rd International Conference on Data Engineering, ICDE 2017*. 107–110. https://doi.org/10.1109/ICDE.2017.51

[9] Lei Chen, Xin Lin, Haibo Hu, Christian S. Jensen, and Jianliang Xu. 2015. Answering why-not questions on spatial keyword top-k queries. In *2015 IEEE 31st International Conference on Data Engineering, ICDE 2015*. 279–290. https://doi.org/10.1109/ICDE.2015.7113291

[10] Lei Chen, Jianliang Xu, Xin Lin, Christian S. Jensen, and Haibo Hu. 2016. Answering why-not spatial keyword top-k queries via keyword adaption. In *2016 IEEE 32nd International Conference on Data Engineering, ICDE 2016*. 697–708. https://doi.org/10.1109/ICDE.2016.7498282

[11] Sean Chester, Alex Thomo, S. Venkatesh, and Sue Whitesides. 2013. Indexing Reverse Top-k Queries in Two Dimensions. In *Database Systems for Advanced Applications, 18th International Conference, DASFAA 2013*. 201–208. https://doi.org/10.1007/978-3-642-37487-6_17

[12] Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008*. 337–340. https://doi.org/10.1007/978-3-540-78800-3_24

[13] Ralf Diestelkämper, Seokki Lee, Boris Glavic, and Melanie Herschel. 2021. Debugging Missing Answers for Spark Queries over Nested Data with Breadcrumb. *Proc. VLDB Endow.* 14, 12 (2021), 2731–2734. https://doi.org/10.14778/3476311.3476331

[14] Ralf Diestelkämper, Seokki Lee, Melanie Herschel, and Boris Glavic. 2021. To Not Miss the Forest for the Trees - A Holistic Approach for Explaining Missing Answers over Nested Data. In *Proceedings of the 2021 International Conference on Management of Data, SIGMOD 2021*. 405–417. https://doi.org/10.1145/3448016.3457249

[15] Eibe Frank, Mark A. Hall, and Ian H. Witten. 2016. The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition. https://www.cs.waikato.ac.nz/ml/weka/Witten_et_al_2016_appendix.pdf

[16] Sainyam Galhotra, Amir Gilad, Sudeepa Roy, and Babak Salimi. 2022. HypeR: Hypothetical Reasoning With What-If and How-To Queries Using a Probabilistic Causal Approach. In *Proceedings of the 2022 International Conference on Management of Data, SIGMOD 2022*. 1598–1611. https://doi.org/10.1145/3514221.3526149

[17] Yunjun Gao, Qing Liu, Gang Chen, Baihua Zheng, and Linlin Zhou. 2015. Answering Why-not Questions on Reverse Top-k Queries. *Proc. VLDB Endow.* 8, 7 (2015), 738–749. https://doi.org/10.14778/2752939.2752943

[18] Amir Gilad, Zhengjie Miao, Sudeepa Roy, and Jun Yang. 2022. Understanding Queries by Conditional Instances. In *Proceedings of the 2022 International Conference on Management of Data, SIGMOD 2022*. 355–368. https://doi.org/10.1145/3514221.3517898

[19] Gurobi Optimization, LLC. 2022. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[20] J. Han, M. Kamber, and J. Pei. 2011. *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann.

[21] Zhian He and Eric Lo. 2012. Answering Why-not Questions on Top-k Queries. In *2012 IEEE 28th International Conference on Data Engineerin, ICDE 2012*. 750–761. https://doi.org/10.1109/ICDE.2012.8

[22] Zhian He and Eric Lo. 2014. Answering Why-Not Questions on Top-K Queries. *IEEE Transactions on Knowledge and Data Engineering* 26, 6 (2014), 1300–1315. https://doi.org/10.1109/TKDE.2012.158

[23] Melanie Herschel and Mauricio A. Hernández. 2010. Explaining Missing Answers to SPJUA Queries. *Proc. VLDB Endow.* 3, 1 (2010), 185–196. https://doi.org/10.14778/1920841.1920869

[24] Vagelis Hristidis, Nick Koudas, and Yannis Papakonstantinou. 2001. PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, SIGMOD 2001*. 259–270. https://doi.org/10.1145/375663.375690

[25] Jiansheng Huang, Ting Chen, AnHai Doan, and Jeffrey F. Naughton. 2008. On the provenance of non-answers to queries over extracted data. *Proc. VLDB Endow.* 1, 1 (2008), 736–747. https://doi.org/10.14778/1453856.1453936

[26] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *Comput. Surveys* 40, 4 (2008), 11. https://doi.org/10.1145/1391729.1391730

[27] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. 2007. Making database systems usable. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2007*. 13–24. https://doi.org/10.1145/1247480.1247483

[28] Cheqing Jin, Rong Zhang, Qiangqiang Kang, Zhao Zhang, and Aoying Zhou. 2014. Probabilistic Reverse Top-k Queries. In *Database Systems for Advanced Applications, 19th International Conference, DASFAA 2014*. 406–419. https://doi.org/10.1007/978-3-319-05810-8_27

[29] Aristotelis Leventidis, Jiahui Zhang, Cody Dunne, Wolfgang Gatterbauer, H. V. Jagadish, and Mirek Riedewald. 2020. QueryVis: Logic-based Diagrams help Users Understand Complicated SQL Queries Faster. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, SIGMOD 2020*. 2303–2318. https://doi.org/10.1145/3318464.3389767

[30] Chenjie Li, Juseung Lee, Zhengjie Miao, Boris Glavic, and Sudeepa Roy. 2022. CaJaDE: Explaining Query Results by Augmenting Provenance with Context. *Proc. VLDB Endow.* 15, 12 (2022), 3594–3597. https://www.vldb.org/pvldb/vol15/p3594-li.pdf

[31] Chenjie Li, Zhengjie Miao, Qitian Zeng, Boris Glavic, and Sudeepa Roy. 2021. Putting Things into Context: Rich Explanations for Query Answers using Join Graphs. In *Proceedings of the 2021 International Conference on Management of Data, SIGMOD 2021*. 1051–1063. https://doi.org/10.1145/3448016.3459246

[32] Qing Liu, Yunjun Gao, Gang Chen, Baihua Zheng, and Linlin Zhou. 2016. Answering why-not and why questions on reverse top-k queries. *The VLDB Journal* 25, 6 (2016), 867–892. https://doi.org/10.1007/s00778-016-0443-4

[33] Qing Liu, Yunjun Gao, Linlin Zhou, and Gang Chen. 2017. IS2R: A System for Refining Reverse Top-k Queries. In *2017 IEEE 33rd International Conference on Data Engineering, ICDE 2017*. 1371–1372. https://doi.org/10.1109/ICDE.2017.182

[34] Sports Reference LLC. 2022. Basketball-Reference.com - Basketball Statistics and History. https://www.basketball-reference.com/, visited on October 2, 2022.

[35] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. 2010. The Complexity of Causality and Responsibility for Query Answers and non-Answers. *Proc. VLDB Endow.* 4, 1 (2010), 34–45. https://doi.org/10.14778/1880172.1880176

[36] Alexandra Meliou, Wolfgang Gatterbauer, and Dan Suciu. 2011. Reverse Data Management. *Proc. VLDB Endow.* 4, 12 (2011), 1490–1493. https://doi.org/10.14778/3402755.3402803

[37] Alexandra Meliou and Dan Suciu. 2012. Tiresias: the database oracle for how-to queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012*. 337–348. https://doi.org/10.1145/2213836.2213875

[38] Zhengjie Miao, Sudeepa Roy, and Jun Yang. 2019. Explaining Wrong Queries Using Small Examples. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD 2019*. 503–520. https://doi.org/10.1145/3299869.3319866

[39] Sudeepa Roy. 2022. Toward Interpretable and Actionable Data Analysis with Explanations and Causality. *Proc. VLDB Endow.* 15, 12 (2022), 3812–3820. https://www.vldb.org/pvldb/vol15/p3812-roy.pdf

[40] Quoc Trung Tran and Chee-Yong Chan. 2010. How to ConQueR why-not questions. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010*. 15–26. https://doi.org/10.1145/1807167.1807172

[41] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Nørvåg. 2010. Reverse top-k queries. In *2010 IEEE 26th International Conference on Data Engineering, ICDE 2010*. 365–376. https://doi.org/10.1109/ICDE.2010.5447890

[42] Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Kjetil Nørvåg. 2011. Monochromatic and Bichromatic Reverse Top-k Queries. *IEEE Transactions on Knowledge and Data Engineering* 23, 8 (2011), 1215–1229. https://doi.org/10.1109/TKDE.2011.50

[43] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, and Yannis Kotidis. 2010. Identifying the Most Influential Data Objects with Reverse Top-k Queries. *Proc. VLDB Endow.* 3, 1 (2010), 364–372. https://doi.org/10.14778/1920841.1920890

[44] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, and Yannis Kotidis. 2013. Branch-and-bound algorithm for reverse top-k queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013*. 481–492. https://doi.org/10.1145/2463676.2465278

[45] Biyan Wang, Zhengqing Dai, Cuiping Li, and Hong Chen. 2010. Efficient computation of monochromatic reverse top-k queries. In *Seventh International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2010*, Vol. 4. 1788–1792. https://doi.org/10.1109/FSKD.2010.5569416

[46] Dong Xin, Chen Chen, and Jiawei Han. 2006. Towards Robust Indexing for Ranked Queries. In *Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB 2006*. 235–246. http://dl.acm.org/citation.cfm?id=1164149

[47] Wenjian Xu, Zhian He, Eric Lo, and Chi-Yin Chow. 2016. Explaining Missing Answers to Top-k SQL Queries. *IEEE Transactions on Knowledge and Data Engineering* 28, 8 (2016), 2071–2085. https://doi.org/10.1109/TKDE.2016.2547398

[48] Meike Zehlike, Ke Yang, and Julia Stoyanovich. 2023. Fairness in Ranking, Part I: Score-Based Ranking. *ACM Comput. Surv.* 55, 6 (2023), 118:1–118:36. https://doi.org/10.1145/3533379

[49] Meike Zehlike, Ke Yang, and Julia Stoyanovich. 2023. Fairness in Ranking, Part II: Learning-to-Rank and Recommender Systems. *ACM Comput. Surv.* 55, 6 (2023), 117:1–117:41. https://doi.org/10.1145/3533380

[50] Zhao Zhang, Cheqing Jin, and Qiangqiang Kang. 2014. Reverse k-Ranks Query. *Proc. VLDB Endow.* 7, 10 (2014), 785–796. https://doi.org/10.14778/2732951.2732952

[51] Jingwen Zhao, Yunjun Gao, Gang Chen, and Rui Chen. 2018. Why-Not Questions on Top-k Geo-Social Keyword Queries in Road Networks. In *2018 IEEE 34th International Conference on Data Engineering, ICDE 2018*. 965–976. https://doi.org/10.1109/ICDE.2018.00091