Query Refinement for Diverse Top-*k* Selection

FELIX S. CAMPBELL, Ben-Gurion University of the Negev, Israel
ALON SILBERSTEIN, Ben-Gurion University of the Negev, Israel
JULIA STOYANOVICH, New York University, United States of America
YUVAL MOSKOVITCH, Ben-Gurion University of the Negev, Israel

Database queries are often used to select and rank items as decision support for many applications. As automated decision-making tools become more prevalent, there is a growing recognition of the need to diversify their outcomes. In this paper, we define and study the problem of modifying the selection conditions of an `ORDER BY` query so that the result of the modified query closely fits some user-defined notion of diversity while simultaneously maintaining the intent of the original query. We show the hardness of this problem and propose a mixed-integer linear programming (MILP) based solution. We further present optimizations designed to enhance the scalability and applicability of the solution in real-life scenarios. We investigate the performance characteristics of our algorithm and show its efficiency and the usefulness of our optimizations.

CCS Concepts: • **Information systems → Data management systems**; • **Social and professional topics → Socio-technical systems**.

Additional Key Words and Phrases: Query refinement, diversity, provenance, ranking, top-k

ACM Reference Format:
Felix S. Campbell, Alon Silberstein, Julia Stoyanovich, and Yuval Moskovitch. 2024. Query Refinement for Diverse Top-*k* Selection. *Proc. ACM Manag. Data* 2, 3 (SIGMOD), Article 166 (June 2024), 27 pages. https://doi.org/10.1145/3654969

1 INTRODUCTION

Ranking-based decision making is prevalent in various application domains, including hiring [18] and school admission [36]. Typically, this process involves selecting qualifying candidates based on specific criteria (e.g., for a job position) and ranking them using a quantitative measure to identify the top candidates among those who qualify (e.g., for a job interview or offer). This process may be automated and expressed using SQL queries, with the `WHERE` clause used to select candidates who meet certain requirements, and the `ORDER BY` clause used to rank them. We next illustrate this idea using a simple example in the context of awarding scholarships.

Example 1.1. Consider a foundation that wishes to grant six high-performing students scholarships to universities in order to encourage participation in STEM programs. The foundation utilizes a database of all students seeking scholarships provided by their schools, which may be filtered according to the requirements of the foundation. Table 1 shows the students dataset, consisting of five attributes: a unique ID, gender, family's income level, grade point average (GPA), and SAT score. The schools also provide information on the student's involvements with extracurricular activities, which are shown in Table 2 as a dataset with two attributes: the student's ID and an

Authors' addresses: Felix S. Campbell, Ben-Gurion University of the Negev, Israel, felixsal@post.bgu.ac.il; Alon Silberstein, Ben-Gurion University of the Negev, Israel, alonzilb@post.bgu.ac.il; Julia Stoyanovich, New York University, United States of America, stoyanovich@nyu.edu; Yuval Moskovitch, Ben-Gurion University of the Negev, Israel, yuvalmos@bgu.ac.il.

ACM 2836-6573/2024/6-ART166
https://doi.org/10.1145/3654969

Table 1. `Students`

ID	Gender	Income	GPA	SAT $\downarrow$
t_1	M	Medium	3.7	1590
t_2	F	Low	3.8	1580
t_3	F	Low	3.6	1570
t_4	M	High	3.8	1560
t_5	F	Medium	3.6	1550
t_6	F	Low	3.7	1550
t_7	M	Low	3.7	1540
t_8	F	High	3.9	1530
t_9	F	Medium	3.8	1530
t_{10}	M	High	3.7	1520
t_{11}	F	Low	3.8	1490
t_{12}	M	Medium	4.0	1480
t_{13}	M	High	3.5	1430
t_{14}	F	Low	3.7	1410

Table 2. `Activities`

ID	Activity
t_1	SO
t_2	SO
t_3	GD
t_4	RB
t_4	TU
t_5	MO
t_6	SO
t_7	RB
t_8	RB
t_8	TU
t_{10}	RB
t_{11}	RB
t_{12}	RB
t_{14}	RB

abbreviation representing the activity in which they participated. The set of activities in Table 2 consists of robotics (*RB*), Science Olympiad (*SO*), Math Olympiad (*MO*), game development (*GD*), and a STEM tutoring organization (*TU*).

The foundation would like to award these scholarships to students who have displayed interest in STEM fields through their involvement in extracurricular activities and have maintained a minimum GPA. The selected students are ranked by their SAT exam scores, and the foundation grants funding to the best six students and additional funding to the top three students. These requirements can be expressed using the following query, which selected students who have participated in an extracurricular robotics club with a minimum GPA of 3.7:

```
SELECT DISTINCT ID, Gender, Income
FROM Students NATURAL JOIN Activities
WHERE GPA >= 3.7 AND Activity = 'RB'
ORDER BY SAT DESC
```

We refer to this query throughout as the *scholarship query*. Evaluating this query over the datasets in Tables 1 and 2 produces the ranking $[t_4, t_7, t_8, t_{10}, t_{11}, t_{12}]$. Therefore, the foundation awards students t_4, t_7, t_8 with an extra scholarship and students t_{10}, t_{11}, and t_{12} with the regular amount.

If the query is part of some high-stakes decision-making process, stating diversity requirements as cardinality constraints over the presence of some demographic groups in the top-k result is natural. For instance, in the above example, the foundation may wish to promote female students in STEM by awarding a proportional number of scholarships to male and female applicants, i.e., the top-6 tuples in the output should include at least three females. Moreover, to expand access to STEM education, the foundation may also wish to limit the extended scholarships granted to students from high-income families. Namely, the top-3 results should include at most one student with a high income. The *scholarship query* does not satisfy these constraints since the top-6 tuples are $t_4, t_7, t_8, t_{10}, t_{11}$ and t_{12}, which includes only two females (t_8 and t_{11}), and the top-3 includes two students from high-income families (t_4 and t_8).

In this paper, we propose a novel *in-processing* method to improve the diversity of a ranking by refining the query that produces it.

Example 1.2. The *scholarship query* may be refined by adjusting the condition on `Activity` to include students involved in Science Olympiad (*SO*), resulting in the following query:

```
SELECT DISTINCT ID, Gender, Income
FROM Students NATURAL JOIN Activities
WHERE GPA >= 3.7 AND (Activity = 'RB' OR Activity = 'SO')
ORDER BY SAT DESC
```

Note that the essence of the query (selecting students who have displayed interest in STEM) is maintained by the refined query, while the constraints are satisfied as the top-6 tuples (t_1, t_2, t_4, t_6, t_7, and t_8) consist of three women (t_2, t_6 and t_8) where the top-3 includes only a single student (t_4) with high income.

The notion of refining queries to satisfy a set of diversity constraints was recently presented in [26, 27], however, this work focuses on cardinality constraints over the *entire* output and does not consider the order of tuples. The problem of ensuring diverse outputs in ranking queries has received much recent attention from the research community [1, 9, 10, 23, 45, 46]. For instance, in [10, 45, 46], output rankings are modified directly in a *post-processing step*, in order to satisfy a given set of constraints over the cardinality of protected groups in the ranking. E.g., to fulfill the desired constraints in the above example, the foundation may manipulate the output, awarding t_4, t_5, t_6, t_7, t_8 and t_{10} with a scholarship, where t_4, t_5 and t_6 will get the extended grant. However, this leaves open the question of *how one may obtain such a ranking in the first place.*

Further, post-processing may be problematic to use to improve diversity, for two reasons: (1) by definition, it modifies the results after they were computed, raising a procedural fairness concern, and (2) it may explicitly use information about demographic or otherwise protected group membership, raising a disparate treatment concern. In contrast, in-processing is usually legally permissible, essentially because it applies the same evaluation process to all individuals. That is, by modifying the query we produce a new set of requirements, and test all individuals against these same requirements. In contrast, post-processing methods may decide to include or exclude individuals based on which groups they belong to, therefore treading individuals differently depending on group membership. Alternative in-processing solutions involve adjustments to the ranking algorithm [1] or modifying items to produce a different score [9, 23]. Our approach, conversely, assumes the ranking algorithms and scores of different items are well-designed and fixed, and we aim to modify the set of tuples to be ranked.

Our goal is to find minimal refinements to the original query that fulfill a specified set of constraints, however, we note that the notion of minimality may be defined in different ways, depending, for example, on the legal requirements or on the user's preferences.

Example 1.3. We may refine the *scholarship query* by relaxing the GPA requirement to 3.6 and including students who participated in a game development activity (GD), obtaining the following query:

```
SELECT DISTINCT ID, Gender, Income
FROM Students NATURAL JOIN Activities
WHERE GPA >= 3.6 AND (Activity = 'RB' OR Activity = 'GD')
ORDER BY SAT DESC
```

Similarly to the refined query from Example 1.2, the top-6 students (t_3, t_4, t_7, t_8, t_{10}, and t_{11}) include three women (t_3, t_8, and t_{11}), and there is only a single high-income student (t_3) among the top-3. While the predicates of this refined query are intuitively more distant from the original query than our prior refinement in Example 1.2 (two modifications compared to a single one), its output is more similar to the output of the original query (the top-3 sets differ by one tuple).

To accommodate different alternative query relaxation objectives, as illustrated above, we propose a framework that allows the user to specify their preferred notion of minimality.

To the best of our knowledge, our work is the first to intervene on the ranking process by modifying *which* items are being considered by the ranking algorithm. This admits a large class of ranking algorithms while keeping the relative order of tuples consistent. This—of course—does not come for free; the coarseness of refinements means that there may be no refinement that produces a satisfactorily diverse ranking. Therefore, we study the problem of finding a refined query that is within a specified maximum distance from satisfying all of the constraints, if one exists.

Contributions & roadmap. We begin by formalizing the Best Approximation Refinement problem of obtaining a refined query that is closest, according to a given distance measure, to the original query, while still adhering to a set of cardinality constraints within a maximum distance, and show that this problem is NP-hard (Section 2). We thus propose modeling the problem as a mixed-integer linear program (MILP) and utilizing it to derive an approximate solution (Section 3). Inspired by the use of data annotations (provenance) to perform what-if analysis, i.e., to efficiently reevaluate queries using algebraic expression without constantly accessing a DBMS (see, e.g., [3, 15, 16, 31]), we construct the MILP using data annotation variables. This formulation offers two advantages: it allows us to leverage the effectiveness of existing MILP solvers while avoiding the costly reevaluation of queries on the DBMS.

Existing MILP solvers may solve the problem efficiently however their performance is sensitive to the size of the program. The program generated in Section 3 is linear in the data size, which can be challenging in real-life scenarios as we demonstrate in our experimental evaluation. We, therefore, propose optimizations to make our approach more scalable, using the relevancy of the data and the structure of the set of cardinality constraints to prune and relax our problem (Section 4). In Section 5, we present an extensive experimental evaluation. We developed a dedicated benchmark consisting of real-life datasets and considering realistic scenarios. Our results show the efficiency and scalability of our approach with respect to different parameters of the problem.

2 PROBLEM OVERVIEW

In this paper, we consider the class of conjunctive Select[1]-Project-Join (SPJ)[2] queries with an ORDER BY s clause, generating a ranked list of tuples, where s is a *score function* of a single tuple t. A query Q may have numerical and categorical selection predicates, denoted $\mathsf{Num}(Q)$ and $\mathsf{Cat}(Q)$, respectively. Numerical predicates are of the form $A \diamond C$, where A is a numerical attribute, $C \in \mathbb{R}$, and $\diamond \in \{<, \leq, =, >, \geq\}$. Categorical predicates are of the form $\bigvee_{c \in C} A = c$, where A is a categorical attribute and C is a set of constants from the domain of A. Selection operators combine predicates by taking their conjunction. We use $\mathsf{Preds}(Q)$ to denote the set of attributes appearing in the selection predicates of Q. In the rest of the paper, we simply use query to refer to such queries.

2.1 Preliminaries

Cardinality constraints. Imposing constraints on the cardinality of tuples belonging to a certain group in a query result to mitigate bias and improve diversity was studied in [27, 32]. In the context of ranking, cardinality constraints are used over the top-k of the ranking for various values of k (see, e.g., [10, 33, 45]). Following this vein of research, we allow users to define constraints on the cardinality of groups (i.e., data subgroups) for this setting.

A *group* is a collection of tuples that share the same value(s) for one or more (categorical) attributes and is defined by a conjunction of conditions over values of the attributes. For instance, in Example 1.1, the group including women students is defined by the condition $\texttt{Gender} = F$ and consists of students t_2, t_3, t_5, t_6, t_8, t_9, t_{11}, and t_{14}. The group of low-income women candidates is then defined by the condition $\texttt{Gender} = F \wedge \texttt{Income} = Low$ and consists of students t_2, t_3, t_6, t_{11}, and t_{14}. A cardinality constraint $\ell_{G,k} = n$ (or $u_{G,k} = n$) specifies a lower (or an upper) bound of n

[1] DISTINCT is supported, and is used to select individuals uniquely if they should not appear more than once in the output.
[2] We note that the system may be extended easily to handle unions, but we omit its description here due to space constraints.

tuples belonging to a group G appearing within the top-k tuples of the result. For instance, in our running example, the constraint "at least 3 of the top-6 candidates are women", can be expressed as $\ell_{\mathsf{Gender}=F,k=6} = 3$. Multiple cardinality constraints may be composed together, forming a constraint set that we denote by $\mathcal{C}$.

Refinements. We use the notion of query refinement defined in [30]. Given a query Q, a refinement of Q modifies its selection predicates. A numerical predicate $A \diamond C \in \mathsf{Num}(Q)$ is a modification to the value of C. For categorical predicates $\bigvee_{c \in C} A = c \in \mathrm{Cat}(Q)$, a refinement is done by adding and/or removing predicates from the set of values C. We say that a query Q' is a *refinement* of query Q if Q' is obtained from Q by refining some predicates of Q.

Example 2.1. The *scholarship query* has two predicates: a numerical predicate `GPA ≥ 3.7` and a categorical predicate `Activity = 'RB'`. A possible refinement of the numerical predicate may be `GPA ≥ 3.6`. The categorical predicate may be refined by adding `'GD'` to C. The refined query resulting from Q by applying these refinements is the query depicted in Example 1.3.

2.2 Refinement Distance

Our objective is to find a refinement Q' that fulfills a specified set of constraints *and* preserves the essence of the intent of query Q, i.e., is in some sense close to Q. A key question is how to measure the distance between a query Q and a refinement Q'.

Recall from Example 1.3 that there may be multiple ways to define such distance. In this paper, we support distance functions of two kinds — those that compare the predicates of Q and Q' (*predicate-based*) and those that compare the top-k results of Q and Q', either as sets or in ranked order (*outcome-based*). In both cases, a distance function returns a real number, with a smaller value indicating closer proximity between Q and Q'. As we will discuss later, we use mixed-integer linear programming to find query refinements. Hence, the distance function must be linear (or able to be linearized) in the variables of its input. However, this limitation still permits a diverse set of valuable distance measures, as we demonstrate next.

Predicate-based distance. Given a query Q and a refinement Q', a natural distance measure with respect to a numerical predicate $n_Q = A \diamond C \in \mathsf{Num}(Q)$ is $|n_Q.C - n_{Q'}.C|$, where $n_Q.C$ is the value of C in n_Q and $n_{Q'}.C$ is the value of C in Q'. The distance between all numerical predicates may be (normalized and) aggregated as $\sum_{n_Q \in \mathsf{Num}(Q)} \frac{|n_Q.C - n_{Q'}.C|}{n_Q.C}$. The distance between categorical attributes may be measured using the Jaccard distance, defined for a pair of sets R and S as $J(R,S) = 1 - \frac{|R \cap S|}{|R \cup S|}$. We may aggregate the distance across categorical predicates as $\sum_{c_Q \in \mathrm{Cat}(Q)} J(c_Q.C, c_{Q'}.C)$, where $c_{Q'} \in \mathrm{Cat}(Q')$ is the corresponding categorical attribute of $c_Q \in \mathrm{Cat}(Q)$ (c_Q and $c_{Q'}$ are of the form $\bigvee_{c \in C} A = c$).

Combining numerical and categorical components, we formulate the predicate-based distance between Q and Q' as:

$$DIS_{pred}(Q,Q') = \sum_{n_Q \in \mathsf{Num}(Q)} \frac{|n_Q.C - n_{Q'}.C|}{n_Q.C} + \sum_{c_Q \in \mathrm{Cat}(Q)} J(c_Q.C, c_{Q'}.C)$$

Example 2.2. Let Q' and Q'' be the refinements of the *scholarship query* Q shown in Examples 1.2 and 1.3, respectively. We compute $DIS_{pred}(Q,Q') = \frac{3.7-3.7}{3.7} + (1 - \frac{|\{RB\}|}{|\{RB,SO\}|}) = 0.5$, which is smaller than $DIS_{pred}(Q,Q'') = \frac{3.7-3.6}{3.7} + (1 - \frac{|\{RB\}|}{|\{RB,GD\}|}) \approx 0.53$.

Outcome-based distance. Distance measures in this family compare the top-k items $Q(D)_k$ and $Q'(D)_k$, for some value of k. We consider two types of distance measures: those that look at

Table 3. Relation used for proof of Theorem 2.5

X	Y	Z
A	C	6
A	D	5
A	D	4
B	C	3
A	C	2
B	D	1

the top-k as sets, and those that are sensitive to the ranked order among the top-k items. We give a couple of examples below, noting that many other set-wise and rank-aware distance metrics can be defined.

A natural distance metric computes the Jaccard distance between the *sets* of top-k items of Q and Q': $DIS_{Jaccard}(Q, Q', k) = J(Q(D)_k, Q'(D)_k)$.

Example 2.3. Let Q' and Q'', again, be the refinements of the *scholarship query* Q shown in Examples 1.2 and 1.3, respectively. Then, $DIS_{Jaccard}(Q, Q', k = 3) = 1 - \frac{|\{t_4\}|}{|\{t_1,t_2,t_4,t_7,t_8\}|} = 0.8$, while $DIS_{Jaccard}(Q, Q'', k = 3) = 1 - \frac{|\{t_4,t_7\}|}{|\{t_3,t_4,t_7,t_8\}|} = 0.5$.

Observe that Q'' is closer to Q according to $DIS_{Jaccard}$ at top-3, while Q' is closer to Q according to DIS_{pred}.

Recall that query refinement does not reorder tuples. That is, tuples that belong to $Q(D)_k \cap Q'(D)_k$ will appear in the same relative order in both top-k lists. As another alternative, a rank-aware measure may, for example, use a variant of Kendall's τ [22] that was proposed by Fagin et al. [17] to compare the top-k items of Q and Q'. In a nutshell, this measure, which we denote $DIS_{Kendall}(Q, Q', k)$, considers the new tuples in the top-k (i.e., $Q'(D)_k \setminus Q(D)_k$), and computes how much the tuples in the original top-k ($Q(D)_k$) were displaced. (Cases 2 and 3 from [17] apply in our setting.)

Intuitively, if $DIS_{Kendall}(Q, Q', k = 3) < DIS_{Kendall}(Q, Q'', k = 3)$, then the tuples $Q''(D)_k \setminus Q(D)_k$ are positioned closer to the top of the list than those in $Q'(D)_k \setminus Q(D)_k$.

Example 2.4. To illustrate $DIS_{Kendall}$, we introduce a new refinement Q''', which we define as:

```
SELECT DISTINCT ID, Gender, Income
FROM Students NATURAL JOIN Activities
WHERE GPA >= 3.6 AND (Activity = 'CS' OR Activity = 'MO')
ORDER BY SAT DESC
```

Observe that $DIS_{pred}(Q, Q'') = DIS_{pred}(Q, Q''')$ and $DIS_{Jaccard}(Q, Q'', k = 3) = DIS_{Jaccard}(Q, Q''', k = 3)$. However, the resulting ranking of $Q'''(D)$ is $[t_4, t_5, t_7, t_8, t_{10}, t_{11}, t_{12}]$. This refinement includes a new tuple t_5 in the output where it ranks second, while in Q'', the new tuple included (t_3) ranks first in $Q''(D)$. However, we find that $DIS_{Kendall}(Q, Q'', k = 3) > DIS_{Kendall}(Q, Q''', k = 3)$, meaning that Q''' is preferable to Q'' according to this measure.

These measures can be combined to formulate new measures that take into account both the queries' predicate distance and the outputs, e.g., using a weighted function.

2.3 Problem Formulation

Given a query Q, a set of cardinality constraints C, and a distance measure, our goal is to find a refinement with minimal distance from Q that satisfies the set of constraints. However, we can show that such a refinement may not exist.

Theorem 2.5. *There exists a database D, a query Q over D, and a constraint set C such that no refinement of Q evaluated over D satisfies C.*

Proof. We prove this claim by a simple example. Let Q be the query `SELECT * FROM "Table 3" WHERE Y = 'C' OR Y = 'D' ORDER BY Z DESC`. Let us require that 2 tuples from group `X = 'B'` (or just B for brevity) appear in the top-3 of the ranking, i.e., setting $\ell_{X='B',k=3} = 2$. The original query evaluated over Table 3 selects the entire table, resulting in a ranking with no tuples belonging to B in the top-3. There are then only two possible refinements on the original query: `Y = 'C'` or `Y = 'D'`. In both cases, there is only 1 item of B in the top-3. Neither the original query nor any of its possible refinements result in a query that satisfies the constraints. □

Theorem 2.5 motivates the need to find a refinement that deviates as little as possible from satisfying the constraint set in the case that exact constraint satisfaction is impossible, which allows us to provide results that are more useful to the user than simply stating its infeasibility. To measure the deviation from the satisfaction of a given set of constraints C, we leverage the notion of the *mean absolute percentage error*, as was done in [4]. Specifically, we use it to measure the deviation from the constraints over groups in C and their cardinalities in the output of the (refined) query. We modify its definition to not penalize rankings that are above (below) the cardinalities specified in lower (upper) bound constraints for a group.

Definition 2.6 (Deviation). Recall that $Q(D)_k$ denotes the top-k tuples in the output of the query Q over a database D. The deviation between C and Q, $DEV(Q(D), C)$ is given by

$$\frac{1}{|C|} \sum_{(\mathscr{c}_{G,k}=n)\in C} \frac{\max\left(\text{Sign}(\mathscr{c}) \cdot (n - |Q(D)_k \cap G|), 0\right)}{n}$$

where $\text{Sign}(\mathscr{c})$ is 1 for lower-bound constraints (ℓ) and -1 for upper-bound constraints ($\mathscr{u}$). Larger values represent a larger violation of the constraint set.

When computing deviation, we assume that the output of $Q(D)$ has at least the number of tuples of the largest k with a constraint in C. We refer to this quantity throughout as k^*. We are now ready to formally define the Best Approximation Refinement problem.

Definition 2.7 (Best Approximation Refinement). Given a database D, a query Q, a constraint set C, a maximum deviation from the constraint set $\varepsilon \geq 0$, and a distance measure $DIS : Q \times \mathcal{R} \times k \longrightarrow \mathbb{R}$, the answer to the Best Approximation Refinement problem is the refinement Q' in

$$\underset{Q' \in \mathcal{R}}{\text{argmin}} \; DIS(Q, Q', k) \;\; \text{such that} \;\; DEV(Q'(D), C) \leq \varepsilon$$

where $\mathcal{R}$ is the set of possible refinements of Q that have at least k^* tuples in in their output. Note that the k parameter is optional in the distance measure (e.g., DIS_{pred} does not include it). A special value is returned if there is no refinement Q' with constraint set deviation at most ε.

Best Approximation Refinement provides the most similar (according to the given similarity definition) refinement with an acceptable deviation from satisfying the constraint set.

We can show that this problem is NP-hard.

Theorem 2.8. *Best Approximation Refinement is* NP-hard.

The proof is based on a reduction from Vertex-Cover, a well-known NP-complete decision problem [21]. To this end, we define the following corresponding decision problem. Given a database D, a query Q, a constraint set C, a maximum deviation from the constraint set $\varepsilon \geq 0$, a value k, a distance measure $DIS : Q \times \mathcal{R} \times k \longrightarrow \mathbb{R}$ and a maximum distance $\delta \geq 0$, determine whether there exists a refinement $Q' \in \mathbb{R}$ such that $DEV(Q'(D), C) \leq \varepsilon$ and $DIS(Q, Q', k) \leq \delta$.

An input to the Vertex-Cover problem consists of an undirected graph $G = (V, E)$ and a number S, and the goal is to determine whether there exists a *vertex cover*, i.e., a subset of vertices $V' \subseteq V$ such that for every edge (u, v) in E, one or both of its endpoints, u and v, are in V' and $|V'| \leq S$. Given $G = (V, E)$ and S, we create an input to our problem as follows. The database D consists of a single relation that encodes the graph's edges, as well as dummy edges necessary to satisfy the minimum cardinality assumption made for calculating deviation. The query Q is a query with a categorical predicate selecting the edges with an endpoint in a given set of vertices with an ORDER BY clause that always places dummy edges below real edges. We construct a set of cardinality constraints that are perfectly satisfied if and only if all the real edges are selected. Then, by using the DIS_{pred} measure, we set δ such that there is only a refinement if there are at most S vertices selected as the covering and set ε to 0. The details of the reduction and its correctness proof are given in [8].

3 FINDING THE BEST APPROXIMATION

Our problem may be solved naïvely by an exhaustive search over the possible refinements. However, the search space of refinements becomes intractably large even with relatively modest datasets, as the number of possible refinements is exponential in the number of the query's attributes. Beyond the high cost of an exhaustive search, a naïve solution would require the evaluation of each refinement query on the DBMS to check its deviation from the constraint set.

To address these challenges, we propose a solution based on a mixed-integer linear program (MILP). Mixed-integer linear programming is a model for optimizing a linear objective function subject to a set of expressions (equalities and inequalities) linear in the discrete or continuous variables of the problem, limiting the space of feasible assignments. Solvers for such programs have been developed with techniques to solve even large problems efficiently in practice, as discussed in [43]. By incorporating the concepts introduced in [27, 32], we utilize data annotations to depict potential refinements. These annotations serve as variables in the MILP, and enable us to quantify the deviation from the constraint set without having to reevaluate refinements across the DBMS.

Briefly, a solution for a MILP is an assignment for the variables in the expressions, such that they are satisfied and the objective function is minimized. Intuitively, given a database D, a query Q, a constraint set C, a maximum deviation from the constraint set $\varepsilon \geq 0$, and a distance measure DIS, we construct an instance of MILP such that the solution corresponds to a minimal refinement that produces a ranking such that its deviation is within the maximum deviation ε from the constraint set C while minimizing DIS. By formulating Best Approximation Refinement as a MILP, we can leverage existing tools to streamline the search process.

It is important to note that by using MILP to represent the problem, we are limited to distance measures that can be modeled by a linear program. However, this limitation still allows a wide range of useful distance measures, including the ones defined in Section 2.2. Some of these measures may require additional modeling techniques to become linearized. For example, when modeling the Jaccard distance, we can use the Charnes-Cooper transformation [11]. Similarly, we can introduce auxiliary variables to model the version of Kendall's τ for top-k lists introduced in [17].

The MILP instance we construct consists of two main groups of expressions: those that require that all tuples selected by the refinement are in the ranking according to the ORDER BY expression of Q, and those that enforce that the derived ranking's deviation from the constraint set does not exceed the input bound ε. We next explain the construction of the expressions in each set, and, in order to give a more intuitive picture of the process, demonstrate in Figure 2 how variables are generated from a running example and how they are combined by these expressions.

Table 4. Summary of variables used in our MILP model

Var.	Domain	Description
$C_{A,\diamond}$	$\mathbb{R}$	Refined C for a num. predicate on A with operator $\diamond$
A_v	$\{0,1\}$	Whether a value v is selected by the cat. predicate on A
$A_{v,\diamond}$	$\{0,1\}$	Whether a value v is in the range of the num. predicate on A with operator $\diamond$
r_t	$\{0,1\}$	Whether tuple t is selected by the refinement
s_t	$\mathbb{R}$	Rank of tuple t in the ranking generated by the refinement
$l_{t,k}$	$\{0,1\}$	Whether tuple t is present in the top-k of the ranking generated by the refinement
$E_{G,k}$	$\mathbb{R}$	Number of tuples to add (remove) to satisfy lower-bound (upper-bound) cardinality constraint

Table 5. $\widetilde{Q}$ obtained from *scholarship query*

ID	Gender	Income	Lineage(t)
t_1	M	Medium	$\{Activity_{SO}, GPA_{3.7}\}$
t_2	F	Low	$\{Activity_{SO}, GPA_{3.8}\}$
t_3	F	Low	$\{Activity_{GD}, GPA_{3.6}\}$
t_4	M	High	$\{Activity_{RB}, GPA_{3.8}\}$
t'_4	M	High	$\{Activity_{TU}, GPA_{3.8}\}$
t_5	F	Medium	$\{Activity_{MO}, GPA_{3.6}\}$
t_6	F	Low	$\{Activity_{SO}, GPA_{3.7}\}$
t_7	M	Low	$\{Activity_{RB}, GPA_{3.7}\}$
t_8	F	High	$\{Activity_{RB}, GPA_{3.9}\}$
t'_8	F	High	$\{Activity_{TU}, GPA_{3.9}\}$
t_{10}	M	High	$\{Activity_{RB}, GPA_{3.7}\}$
t_{11}	F	Low	$\{Activity_{RB}, GPA_{3.8}\}$
t_{12}	M	Medium	$\{Activity_{RB}, GPA_{4.0}\}$
t_{14}	F	Low	$\{Activity_{RB}, GPA_{3.7}\}$

3.1 Modeling Refinement Output Using Expressions

Inspired by the use of provenance for query refinements [27, 32], we utilize the notion of data annotations to model refinements through a set of expressions. This set is divided into two parts. The first part is used to model the tuples that satisfy the refinement query's predicates, while the second part ensures that the selected tuples are ordered correctly by the `ORDER BY` expression of the input query. We start by describing the variables used in the expressions.

Variables. Given a query Q and a database D, for each categorical predicate in $\mathrm{Cat}(Q)$ over an attribute A, we define a variable $A_v \in \{0,1\}$ for each value v in the domain of A in D. Intuitively, a solution to the MILP where $A_v = 1$ corresponds to a refinement that includes $A = v$ in the categorical predicates. For each numerical predicate $A \diamond C \in \mathrm{Num}(Q)$, we define a variable $C_{A,\diamond}$ whose value is in the range of values of A in D, and a set of variables $A_{v,\diamond}$ for each value v in the domain of A in D.

Example 3.1. $Activity_{RB}$ and $Activity_{SO}$ are two of the variables generated by the categorical predicate `Activity = 'RB'` since these values are present in the database. The variable $C_{GPA,\geq}$ is generated from the numerical predicate `GPA >= 3.7`. Additionally, the variable $GPA_{3.7,\geq}$ is generated since there exists a tuple in the data with the value 3.7 in the `GPA` attribute.

The value of $C_{A,\diamond}$ represents the value of the constant C in the refinement query, and the variables $A_{v,\diamond}$ are used to determine whether a given tuple t in D (with the value v in A) satisfies that predicate over A in the refined query. More concretely, the variable $A_{v,\diamond}$ is used to reflect whether $v \diamond C_{A,\diamond}$. Finally, we use a variable r_t to denote the existence of a tuple t in the output of a refinement query and a variable s_t to indicate the position of t in the output.

Expressions. We formulate a set of expressions such that the assignment generated by a solver to the MILP instance corresponds to the set of tuples selected by the corresponding refinement query. A tuple is part of a query's output if it satisfies its predicates set. We first define expressions for numerical predicates. Intuitively, a tuple t with value v in attribute A satisfies the predicate $A \diamond C_{A,\diamond}$ if $v \diamond C_{A,\diamond}$. For lower-bound predicates, i.e., when $A \geq C$ or $A > C$, we model this using the following MILP expressions for each predicate in $\mathsf{Num}(Q)$ and each value V in the domain of A in D.

$$\begin{aligned} C_{A,\diamond} + M_A \cdot A_{v,\diamond} &\geq v + (1 - \mathrm{St}(\diamond)) \cdot \delta \\ C_{A,\diamond} - M_A \cdot (1 - A_{v,\diamond}) &\leq v - \mathrm{St}(\diamond) \cdot \delta \end{aligned} \tag{1}$$

where M_A is a constant larger than the maximum absolute value in the domain of the attribute A in D, $\mathrm{St}(\diamond)$ is 1 if $\diamond$ is a strict inequality and 0 otherwise, and δ is some small constant added when $\diamond$ is strict in order to relax the inequality as MILP expressions do not support strict inequalities. We choose δ to be smaller than the smallest pairwise difference between the values in the domain of A, ensuring the relaxation does not include another value from the domain. For upper-bound predicates, we instead use the following set of expressions.

$$\begin{aligned} C_{A,\diamond} - M_A \cdot A_{v,\diamond} &\leq v - (1 - \mathrm{St}(\diamond)) \cdot \delta \\ C_{A,\diamond} + M_A \cdot (1 - A_{v,\diamond}) &\geq v + \mathrm{St}(\diamond) \cdot \delta \end{aligned} \tag{2}$$

Intuitively, the first expressions in (1) and (2) ensure that $A_{v,\diamond}$ is 1 if $v \diamond C_{A,\diamond}$ is true and the second expressions are used to ensure that $A_{v,\diamond}$ is 0 if $v \diamond C_{A,\diamond}$ is false. Together, they enforce that $A_{v,\diamond}$ is 1 if and only if $v \diamond C_{A,\diamond}$ is true.

Example 3.2. Continuing with our example, the following expressions are generated using the variables $C_{GPA,\geq}$ and $GPA_{3.7,\geq}$ for the numerical predicate `GPA ≥ 3.7` in the *scholarship query*.

$$\begin{aligned} C_{GPA,\geq} + 5 \cdot GPA_{3.7,\geq} &\geq 3.701 \\ C_{GPA,\geq} - 5 \cdot (1 - GPA_{3.7,\geq}) &\leq 3.7 \end{aligned}$$

Here M_A is set to 5, a value greater than any value of the attribute `GPA` in the data, and $\mathrm{St}(\diamond)$ is 0 (since the inequality in the predicate is not strict). Consider an assignment that assigns the 3.7 to $C_{GPA,\geq}$. This assignment corresponds to a query with the predicate `GPA ≥ 3.7`. Assigning this value to the above expression results in

$$\begin{aligned} 3.7 + 5 \cdot GPA_{3.7,\geq} &\geq 3.701 \\ 3.7 - 5 \cdot (1 - GPA_{3.7,\geq}) &\leq 3.7 \end{aligned}$$

In this case, the value of $GPA_{3.7,\geq}$ should be 1 as well, indicating that tuples with a value ≥ 3.7 in the `GPA` attribute meet the condition. Indeed, these expressions can be satisfied if and only if the variable $GPA_{3.7,\geq}$ is assigned the value 1. Notice that adding δ in the first expression is necessary in order to guarantee that the only valid assignment for $GPA_{3.7,\geq}$ is 1.

Next, we construct expressions that model the existence of a tuple in the query's output (represented using the variable r_t). The expressions should be able to model any possible refinement. Note that the output of a refinement may include tuples that are not part of the output of the original query. To this end, we use $\widetilde{Q}$ to denote the query obtained from Q by omitting the selection predicates and any `DISTINCT` statement. Intuitively, the output of $\widetilde{Q}$ over D contains the output of every possible refinement query. A tuple t is in the output of a query Q if t satisfies all the predicates in Q. To indicate whether t is part of the output, we leverage the notion of *lineage*. The lineage of a tuple $t \in \widetilde{Q}(D)$ is the set of variables A_v and $A_{v,\diamond}$ that correspond to the values of t for each attribute in $\mathsf{Attr}(Q)$: $\mathsf{Lineage}(t) = \{A_{t.A} \mid \forall\,(\bigvee_{c\in C} A = c) \in \mathsf{Cat}(Q)\} \cup \{A_{t.A,\diamond} \mid \forall (A \diamond C) \in \mathsf{Num}(Q)\}$. Table 5 shows the result of $\widetilde{Q}(D)$ in our running example with the lineage annotation for each tuple.

Since the value of each variable $A_{t.A} = 1$ or $A_{t.A,\diamond} = 1$ indicates the satisfaction of a predicate over A by t, a tuple t is in the output of Q if all predicates in Q are true for t, i.e., $\sum_{p\in\mathsf{Lineage}(t)} p = |\{\mathsf{Preds}(Q)\}|$. We use this property to construct an expression that models the behavior of r_t for each tuple $t \in \widetilde{Q}(D)$. Note that tuples appearing once in $Q(D)$ may appear multiple times in $\widetilde{Q}(D)$. E.g., when using `DISTINCT` selection after a join operation, as the case in the *scholarship query*, where the tuples denoted by t_4 and t_4' represent the same student ID (that appears once in the output). To address this case, we define $S(t) = \{t' \mid t' \in \widetilde{Q}(D),\ \forall a \in \mathsf{Distinct}(Q)\ t.a = t'.a,\ \widetilde{Q}(D)(t') < \widetilde{Q}(D)(t)\}$ where $\mathsf{Distinct}(Q)$ is the set of attributes selected distinctly by Q. Namely, for a tuple t, $S(t)$ is the set of tuples with the same values on attributes selected distinctly that are ranked closer to the top than t. For instance, in our example $S(t_4') = \{t_4\}$. Intuitively, at most one tuple from $S(t) \cup \{t\}$ can appear in the output of the refined query (depending on its selection predicates). We therefore add the following expression to the MILP.

$$\begin{aligned} 0 \le \sum_{p\in\mathsf{Lineage}(t)} p + \sum_{t'\in S(t)} (1 - r_{t'}) - (|\mathsf{Preds}(Q)| + |S(t)|) \cdot r_t \\ \le |\mathsf{Preds}(Q)| + |S(t)| - 1 \end{aligned} \tag{3}$$

The lower bound of this expression prevents r_t from being assigned 1 if *not* all attributes of the tuple satisfy the predicate of the corresponding refinement or *any* tuples sharing its distinct values ranked better than it were already selected. Similarly, the upper bound is used to ensure that r_t is assigned the value 1 in case that *all* of the attributes of the tuple satisfy the predicate of the corresponding refinement and *none* of the tuples sharing its distinct values ranked better than it were already selected[3].

Example 3.3. Consider t_6 in Table 5. The lineage of t_6 is the set of variables $\{Activity_{SO}, GPA_{3.7,\ge}\}$, $|\mathsf{Preds}(Q)| = 2$, and the set $S(t_6)$ is empty given there is only 1 tuple with ID 6 in $\widetilde{Q}(D)$. Thus, the MILP instance has the expression

$$0 \le Activity_{SO} + GPA_{3.7,\ge} - 2 \cdot r_{t_6} \le 1$$

Assuming $GPA_{3.7,\ge} = 1$ and $Activity_{SO} = 1$, r_{t_6} must be assigned 1, indicating that t_6 is part of the output in this case.

Given these r_t variables, we may enforce that there at least k^* tuples in the output of the refinement by the expression

$$\sum_{t\in\widetilde{Q}(D)} r_t \ge k^* \tag{4}$$

The last part required to complete the correspondence between the solution to the MILP instance and the output of a refinement query is modeling the order of the output tuples (according to the `ORDER BY` expression of the input query) through the MILP expressions. We use the set of variables s_t for each tuple in $\widetilde{Q}(D)$, which represents the position of t in the output of the corresponding refinement query. Intuitively, the position of a tuple t in the output of the refinement query Q' is

[3] Allowing the same entity (e.g., student ID in our example) appear multiple times in the output can be done by removing `DISTINCT` from the input query.

one plus the number of tuples $t' \in \widetilde{Q}(D)$ that are part of the output Q' and ranked higher than t (i.e., $\widetilde{Q}(D)(t') < \widetilde{Q}(D)(t)$). For tuples t that are not part of the output of Q', the variable s_t will be assigned a value larger than $|\widetilde{Q}(D)|$. This is modeled using the following set of expressions.

$$1 + |\widetilde{Q}(D)| \cdot (1 - r_t) + \sum_{\substack{t' \in \widetilde{Q}(D), \\ \widetilde{Q}(D)(t') < \widetilde{Q}(D)(t)}} r_{t'} = s_t \tag{5}$$

for each t in $\widetilde{Q}(D)$. Given this, we may further limit s_t to be in the range $[1, 2 \cdot |\widetilde{Q}(D)|]$.

Example 3.4. In our running example $|\widetilde{Q}(D)| = 14$. Thus the expression $1 + 14 \cdot (1 - r_{t_6}) + r_{t_1} + r_{t_2} + r_{t_3} + r_{t_4} + r_{t'_4} + r_{t_5} = s_{t_6}$ is the expressions generated for the tuple t_6. Assuming r_{t_1}, r_{t_2}, r_{t_4} and r_{t_6} are 1 (and the rest of the variables are 0), the value of s_{t_6} must be 4, indicating its position in the ranking in this case.

3.2 Bounding Maximum Deviation

The second part of the solution consists of expressions whose goal is to limit the refinement query's output's deviation from the constraint set C to be at most ε. For each cardinality constraint $ȼ_{G,k} = n$ in C, we are interested in the number of tuples belonging to group G in the top-k of the refined ranking to determine the number of tuples of group G needs to be added or removed to satisfy $ȼ_{G,k} = n$. To model this property, we introduce two sets of new variables $l_{t,k}$ and $E_{G,k}$. The variables $l_{t,k}$ are used to indicate whether a tuple t appears in the top-k ranked output of the corresponding refinement query, and $E_{G,k}$ represents the number of tuples from G in the top-k that need to be added (removed) to satisfy lower-bound (upper-bound) cardinality constraints (i.e., $E_{G,k}$ is equivalent to the numerator in the summation of Definition 2.6 for each cardinality constraint). Intuitively, we may further specify that $E_{G,k} \in [0, k]$.

We use a similar construction to the expressions in (1) to ensure that $l_{t,k} = 1$ if and only if the tuple t appears in the top-k as follows.

$$\begin{gathered} s_t + (2 \cdot |\widetilde{Q}(D)| + 1) \cdot l_{t,k} \geq k + \delta \\ s_t - (2 \cdot |\widetilde{Q}(D)| + 1) \cdot (1 - l_{t,k}) \leq k \end{gathered} \tag{6}$$

where $(2 \cdot |\widetilde{Q}(D)| + 1)$ is the constant coefficient that plays the role of M_A in (1) and δ is a small additive constant as in (1) and (2). We utilize the variables $l_{t,k}$ to determine the values of $E_{G,k}$ using the following expressions for each cardinality constraint $ȼ_{G,k} = n$ in C.

$$\begin{aligned} & E_{G,k} \geq 0 \\ & E_{G,k} \geq \mathsf{Sign}(ȼ) \cdot \left(n - \sum_{t \in \widetilde{Q}(D) \cap G} l_{t,k} \right) \end{aligned} \tag{7}$$

where $\sum_{t \in \widetilde{Q}(D) \cap G} l_{t,k}$ is the number of tuples belonging to group G in the top-k.

Finally, to restrict the deviation of the refinement's output to at most ε, we construct the following expression

$$\frac{1}{|C|} \sum_{(ȼ_{G,k} = n) \in C} \frac{E_{G,k}}{n} \leq \varepsilon \tag{8}$$

Example 3.5. Consider again the database shown in Example 1.1 and the cardinality constraint $\ell_{Gender=\text{'}Female\text{'},k=6} = 3$. Tuples t_2, t_3, t_5, t_6, t_8, t'_8, t_{11}, and t_{14} are in the group `Gender='Female'`.

Thus, we generate the expressions

$$E_{Gender=\text{`}Female\text{'},6} \geq 0$$
$$E_{Gender=\text{`}Female\text{'},6} \geq 3 - (l_{t_2,6} + l_{t_3,6} + l_{t_5,6} + l_{t_6,6} + l_{t_8,6} + l_{t'_8,6} + l_{t_{11},6} + l_{t_{14},6})$$

where the value of $l_{t_6,6}$, for instance, is used in the expressions

$$s_{t_6} + 25 \cdot l_{t_6,6} \geq 6.001$$
$$s_{t_6} - 25 \cdot (1 - l_{t_6,6}) \leq 6$$

Continuing Example 3.4, assuming s_{t_6} is assigned the value 4, forcing the assignment of 1 to $l_{t_6,6}$. Assuming $s_{t_2} = 2$ and $s_{t_8} = 6$, we would similarly get that the value of $l_{t_2,6}$ and $l_{t_8,6}$ must be 1. Using these values in expression generated by (7) results in

$$E_{Gender=\text{`}Female\text{'},6} \geq 0$$
$$E_{Gender=\text{`}Female\text{'},6} \geq 3 - (1 + 1 + 1) \geq 0$$

This intuitively means that no additional tuples from the group `Gender='Female'` are required to satisfy the constraint.

We summarize our mixed-integer linear program in Figure 1, and its variables in Table 4. By satisfying all of these expressions together, we produce rankings that are both valid and sufficiently satisfactory of the constraint set. In fact, we can show that any satisfying assignment α to the variables in the expressions generated by (1)-(8) corresponds to a valid refinement that is sufficiently satisfactory.

Theorem 3.6 (Solution correctness). *Let D be a dataset, Q be a query over D, C be a set of cardinality constraints, and $\varepsilon \geq 0$ be a threshold over the deviation from C. There is an assignment α satisfying the expressions generated by (1-8) if and only if there is a refinement Q' for Q such that*

① *For each $(\bigvee_{c \in C} A = c) \in Cat(Q')$, $\alpha(A_c) = 1 \iff c \in C$*
② *For each $(A \diamond C) \in Num(Q')$, $\alpha(C_{A,\diamond}) = C$*
③ *$DEV(Q'(D), C) \leq \frac{1}{|C|} \sum_{(\not{c}_{G,k}=n) \in C} \frac{\alpha(E_{G,k})}{n} \leq \varepsilon$*

To prove this, we first show that there is an assignment α satisfying the expressions (1)-(6) if and only if there is a refinement of the query with properties ① and ② and that for each cardinality constraint $(\not{c}_{G,k} = n) \in C$ and $t \in G$, it holds that $t \in Q'(D)_k \cap G \iff \alpha(l_{t,k}) = 1$. We then use Definition 2.6 and expression (7) to prove the claim. See [8] for details.

Model limitation. Given an input to our program, we construct a MILP program. The correctness of the solution generated by the program, as stated in Theorem 3.6, relies on three properties. First, every possible refinement may be represented as an assignment to the variables of the program. Second, the tuples in the output of any potential refinement are in the same relative order, and finally, every tuple in the output satisfies all the predicates of the corresponding refinement. We define the problem for SPJ queries, and thus, our model is designed to handle SPJ queries, and these properties hold for them. We note that supporting other classes of queries may require modifications to the problem definition as well as to our proposed model. For instance, in union queries, it is enough for a tuple in the output to satisfy the predicates of one branch of the union, in contrast to the third property. This may be handled straightforwardly, as noted in Section 2. Handling nested queries is more challenging since they may contain multiple selection statements at different nesting levels. The problem definition should first be extended to properly define how such a query can be refined, e.g., whether refinements at different nesting levels are allowed. Our proposed model cannot capture the refinement of selection statements in different nesting levels

$$
\begin{array}{lll}
\min & \mathit{DISTANCE} & \\
\text{s.t.} & C_{A,\diamond} + M_A \cdot A_{v,\diamond} \geq v + (1 - \mathsf{St}(\diamond)) \cdot \delta & \forall (A \diamond C) \in \mathsf{Num}^{>}(Q) \\
& C_{A,\diamond} - M_A \cdot (1 - A_{v,\diamond}) \leq v - \mathsf{St}(\diamond) \cdot \delta & \forall (A \diamond C) \in \mathsf{Num}^{>}(Q) \\
& C_{A,\diamond} - M_A \cdot A_{v,\diamond} \leq v - (1 - \mathsf{St}(\diamond)) \cdot \delta & \forall (A \diamond C) \in \mathsf{Num}^{<}(Q) \\
& C_{A,\diamond} + M_A \cdot (1 - A_{v,\diamond}) \geq v + \mathsf{St}(\diamond) \cdot \delta & \forall (A \diamond C) \in \mathsf{Num}^{<}(Q) \\
& \begin{array}{l} 0 \leq \sum\limits_{p \in \mathsf{Lineage}(t)} p + \sum\limits_{t' \in S(t)} (1 - r_{t'}) \\ \quad - (|\mathsf{Preds}(Q)| + |S(t)|) \cdot r_t \\ \quad \leq |\mathsf{Preds}(Q)| + |S(t)| - 1 \end{array} & \forall t \in \widetilde{Q}(D) \\
& \sum\limits_{t \in \widetilde{Q}(D)} r_t \geq k^* & \\
& 1 + |\widetilde{Q}(D)| \cdot (1 - r_t) + \sum\limits_{\substack{t' \in \widetilde{Q}(D), \\ \widetilde{Q}(D)(t') < \widetilde{Q}(D)(t)}} r_{t'} = s_t & \forall t \in \widetilde{Q}(D) \\
& s_t + (2 \cdot |\widetilde{Q}(D)| + 1) \cdot l_{t,k} \geq k + \delta & \forall t \in \widetilde{Q}(D), (\ell_{G,k} = n) \in C \\
& s_t - (2 \cdot |\widetilde{Q}(D)| + 1) \cdot (1 - l_{t,k}) \leq k & \forall t \in \widetilde{Q}(D), (\ell_{G,k} = n) \in C \\
& E_{G,k} \geq 0 & \forall (\ell_{G,k} = n) \in C \\
& E_{G,k} \geq \mathsf{Sign}(\ell) \cdot \left(n - \sum\limits_{t \in \sigma_G(\widetilde{Q}(D))} l_{t,k} \right) & \forall (\ell_{G,k} = n) \in C \\
& \dfrac{1}{|C|} \sum\limits_{(\ell_{G,k} = n) \in C} \dfrac{E_{G,k}}{n} \leq \varepsilon &
\end{array}
$$

Fig. 1. Summary of our MILP model

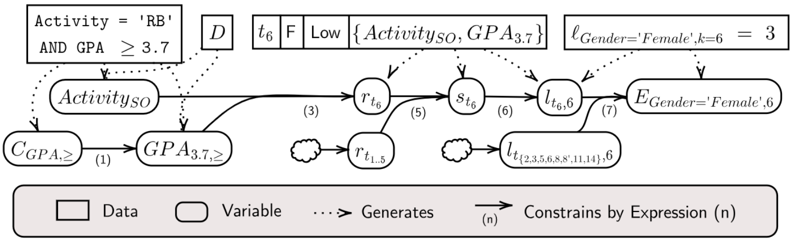


Fig. 2. Diagram illustrating the expression generation for our running example. The predicate `Activity = ‘RB’ AND GPA ≥ 3.7` generates the variables $Activity_{SO}$ and $GPA_{3.7,\geq}$ as ‘SO’ and 3.7 are values that appear for those attributes respectively in the database D. $C_{GPA,\geq}$ is also generated by the predicate to hold the new constant of the predicate, and constrains the value of $GPA_{3.7,\geq}$ by (1). The tuple $t_6 \in \widetilde{Q}$ generates the variable r_6, whose value is constrained through (3) by the values of $Activity_{SO}$ and $GPA_{3.7,\geq}$ due to its lineage. It also generates the variable s_6, which is then constrained by the value of the r_t values for the tuples that rank better than it, i.e., $r_{t_{1..5}}$, through (5). Finally, the constraint $\ell_{Gender=‘Female’,k=6} = 3$ combines with t_6 to generate the variable $l_{t_6,6}$ which is constrained by the value of s_{t_6} by (6). The constraint generates the variable $E_{Gender=‘Female’,6}$ which is constrained through (7) by the values of all the $l_{t,6}$ variables for which t is a part of the group (listed in Example 3.5).

and, therefore, does not fulfill the first property. Moreover, if the `ORDER BY` clause relates to an inner query, refining the inner query may change the relative order of the tuples in contrast to the second property.

4 OPTIMIZATIONS

In Section 3, we have presented a MILP formulation designed to solve the Best Approximation Refinement problem. While this approach enables us to leverage existing MILP solvers that can solve the problem efficiently, they often encounter difficulties when dealing with extensive programs (containing numerous expressions), and a large number of variables [43]. While the number of expressions and variables in the MILP we generate is linear in the data size, MILP solvers struggle to scale and solve the generated programs as we show in Section 5.

To this end, we propose three optimizations for the construction of the MILP problem: one is a general optimization that applies in all cases, and the other two are limited to some cases. The first optimization is relevancy-based and removes from consideration tuples that are irrelevant to determining the satisfaction of the constraint set. The second optimization reduces the number of binary variables in the MILP problem by combining redundant variables. This optimization cannot be applied for queries with a `DISTINCT` statement. The third optimization relaxes the expression used to determine the score of a tuple in the new ranking. This optimization is only applicable to tuples belonging to groups with only lower-bound or only upper-bound constraints, but not both.

Relevancy. We propose a relevancy-based optimization to reduce the number of expressions and variables in our problem. Recall that we use k^* to denote the maximal k that appears in the constraint set C. Then, by removing tuples that could never appear in the top-k^* in any refinement, we are able to avoid adding their variables and expressions to our problem. We determine the relevancy of these tuples by selecting the top-k^* of the groups of tuples that share the same lineage. Let $[\mathtt{Lineage}(t)]$ be the equivalence class of tuples that share the same lineage as a tuple t. Then for a tuple t, let $T(t)$ be the ranking generated by ranking the tuples of $[\mathtt{Lineage}(t)]$ according to the `ORDER BY` clause of Q. We see trivially that it is not possible for tuples past position k^* in $T(t)$ for all t in $\widetilde{Q}(D)$ to be included in the top-k^* of any refinement. Thus, it is sufficient to consider only the top-k^* of $T(t)$, denoted by $T(t)_{k^*}$, in the generated program, and we replace $\widetilde{Q}(D)$ in the expressions referencing it in Figure 1 with $\bigcup_{t\in\widetilde{Q}(D)} T(t)_{k^*}$ ranked according to the `ORDER BY` clause of Q.

Example 4.1. Consider t_{14} from Table 5. Its equivalence class $[\mathtt{Lineage}(t_{14})]$ is the set $\{t_7, t_{10}, t_{14}\}$. Assume we are interested in satisfying a single constraint $\ell_{Gender=\text{'}Female\text{'},k=2} = 1$. Note that the tuple t_{14} can never appear in the top-2 of any refinement query, as any refinement that includes t_{14} includes tuples with its same lineage, i.e., t_7 and t_{10}. Therefore, it is safe to remove all variables and expressions related to t_{14} from consideration.

This optimization is most effective when k^* is small, and there are few lineage equivalence classes. In Section 5, we show that this is often the case in queries over real data sets. We further demonstrate the effect of k^* on the running time in Figure 4.

Selecting lineages. Recall that the program we generate includes a binary variable r_t for each tuple in $\widetilde{Q}(D)$. However, tuples sharing the same lineage all have equal values for their r_t variables. Therefore, we can use a single variable for all tuples in the same lineage equivalence classes.

Example 4.2. To demonstrate this idea, consider the *scholarship query* without its `DISTINCT` statement and consider again the tuple t_{14} with its equivalence class shown in Example 4.1. If t_{14} satisfies the selection condition of a refinement on the *scholarship query*, then t_7 and t_{10} must satisfy the conditions as well, as they share the same lineage. Therefore, we have the equivalence $r_{t_{14}} = r_{t_7} = r_{t_{10}}$. The variables r_{t_7} and $r_{t_{10}}$ are then made redundant, as they always have the same value as $r_{t_{14}}$.

In order to avoid such redundancy, instead of constructing the set of r_t variables, we construct a set of variables $r_{[\mathsf{Lineage}(t)]}$ for every tuple t in $\widetilde{Q}(D)$. Using (3) as a basis, we are able to model $r_{[\mathsf{Lineage}(t)]}$ being assigned 1 if and only if the tuples in $[\mathsf{Lineage}(t)]$ satisfy the selection condition of the corresponding refinement query. Instead of constructing expression (3) for each tuple in $\widetilde{Q}(D)$, we construct the following expression for each $r_{[\mathsf{Lineage}(t)]}$ variable: $0 \leq \sum_{p \in \mathsf{Lineage}(t)} p - |\mathsf{Preds}(Q)| \cdot r_{[\mathsf{Lineage}(t)]} \leq |\mathsf{Preds}(Q)| - 1$. Furthermore, in order to ensure that the s_t values are modeled as before, we modify (5) by changing r_t to $r_{[\mathsf{Lineage}(t)]}$ and $r_{t'}$ to $r_{[\mathsf{Lineage}(t')]}$. We note that this optimization cannot be applied if the input query includes a `DISTINCT` statement, as we need this information in order to not select tuples that already have at least one tuple sharing its distinct value(s) in the output.

Relaxation for single-constraint-type tuples. We present another optimization that is possible when a tuple belongs to groups that have only either lower-bound (ℓ) or upper-bound (u) cardinality constraints made on them. We define the set of tuples belonging only to groups with lower-bound constraints as $L = \{t \in \widetilde{Q}(D) \mid \nexists (u_{G,k} = n) \in C, t \in \widetilde{Q}(D) \cap G\}$. We define a similar set U for upper-bound tuples, replacing $u_{G,k} = n$ in the quantifier with $\ell_{G,k} = n$. Then, for a tuple $t \in L$, we relax expression (5) to $1 + |\widetilde{Q}(D)| \cdot (1 - r_t) + \sum_{t' \in \widetilde{Q}(D), \widetilde{Q}(D)(t') < \widetilde{Q}(D)(t)} r_{t'} \leq s_t$. For tuples in U we set an upper bound instead ($\geq s_t$). This relaxation makes finding feasible solutions for this model easier and may be used by presolving techniques in MILP solvers.

In order to understand why this maintains the correctness of our solution, consider the lower-bound constraints. Intuitively, we can allow the s_t variables of tuples belonging to the group defined in the constraint in a given top-k to be assigned a value larger than the position of t in the ranking, as this could only result in a higher deviation for lower-bound constraints as determined by (8). Suppose the deviation as calculated by (8) is higher than the true deviation of the corresponding refinement. Then, the refinement returned by the MILP is still a correct answer as (8) bounds the calculated deviation by the input ε value, and therefore the true deviation of the corresponding refinement cannot be more than ε. On the other hand, if s_t were assigned a value *smaller* than the position of t in the ranking of the corresponding refinement's output, then the deviation as calculated by (8) may be lower than that of the true deviation of the corresponding refinement, which could cause the MILP to return a refinement with a deviation higher than permitted. The case for upper-bound constraints is symmetric.

5 EXPERIMENTS

We performed an experimental analysis of our proposed algorithm on real-life and synthetic datasets considering realistic scenarios. We first examine the effect of different parameters on the running time. We show that our solution scales, performs well on realistic scenarios and that the optimization presented in Section 4 are effective. We then compare our solution to [27, 32] that studies a similar problem for queries without ranking. We demonstrate the differences between solutions and compare their outputs and performance through a use case.

5.1 Evaluation Benchmark

To the best of our knowledge, we are the first to consider this problem, and there is no benchmark consisting of datasets, including ranking queries and sets of cardinality constraints. To this end, we have developed a dedicated benchmark that involves real-life datasets used in the context of ranking as follows.

Table 6. Queries and constraints

Dataset	Query	Predicates	Order by (DESC)	Constraints

Astronauts	Q_A	`"Graduate Major" = 'Physics'`
		`AND "Space Walks" <= 3`
		`AND "Space Walks" >= 1`
			`"Space Flight (hrs)"`	(1) $\ell_{Gender='F',k} = \frac{k}{2}$ (4) $\ell_{Status='Management',k} = \frac{k}{5}$
				(2) $\ell_{Gender='M',k} = \frac{k}{2}$ (5) $\ell_{Status='Retired',k} = \frac{k}{5}$
				(3) $\ell_{Status='Active',k} = \frac{k}{5}$

Law Students	Q_L	`Region = 'GL'`
		`AND GPA <= 4.0`
		`AND GPA >= 3.5`
			`LSAT`	(1) $\ell_{Sex='F',k} = \frac{k}{2}$ (4) $\ell_{Race='White',k} = \frac{k}{5}$
				(2) $\ell_{Sex='M',k} = \frac{k}{2}$ (5) $\ell_{Race='Asian',k} = \frac{k}{5}$
				(3) $\ell_{Race='Black',k} = \frac{k}{5}$

MEPS	Q_M	`Age > 22`
		`AND "Family Size" >= 4`
			`Utilization`	(1) $\ell_{Sex='F',k} = \frac{k}{2}$ (4) $\ell_{Race='Black',k} = \frac{k}{5}$
				(2) $\ell_{Sex='M',k} = \frac{k}{2}$ (5) $\ell_{Race='White',k} = \frac{k}{5}$
				(3) $\ell_{Race='Asian',k} = \frac{k}{5}$

TPC-H	Q_5	`Region = 'ASIA'`	`Revenue`	(1) $\ell_{OrderPrio='5-LOW',k} = \frac{k}{2}$ (4) $\ell_{MktSeg='BUILDING',k} = \frac{k}{5}$ (2) $\ell_{OrderPrio='3-MEDIUM',k} = \frac{k}{5}$ (5) $\ell_{MktSeg='MACHINERY',k} = \frac{k}{5}$ (3) $\ell_{MktSeg='AUTOMOBILE',k} = \frac{k}{5}$

- **Astronauts**[4]: A dataset of 19 attributes containing 357 NASA astronauts and information about their careers. Astronauts are ranked in descending order by their number of space flight hours, as was done in [39].
- **Law Students [25, 44]**: A dataset of 8 attributes containing 21,790 law students and various evaluations such as grade point average, LSAT examination scores, and first year grade average. Students are ranked by their LSAT scores, as in [47].
- **MEPS**[5]: A dataset of 1,941 attributes containing 34,655 individuals and information related to their usage of healthcare. Patients are ranked in descending order by a combination of utilization metrics (office-based visits + ER visits + in-patient nights + home health visits), as was done in [45].

To evaluate scalability, we use Synthetic Data Vault (SDV) [35] to learn the distributions of our real-life datasets and subsequently synthesize scaled-up versions. We also use the **TPC-H Benchmark**, which includes complex queries involving multiple tables. We generate a TPC-H dataset of scale factor 1, which is approximately 1 GB of data. We use Query 5 (Q5) from the TPC-H specification and remove the predicates filtering on date types.

Queries and constraints. Table 6 summarizes the queries and constraints used. We generated queries and constraints for each dataset, showcasing real-life scenarios. Each row in the table represents a query. For example, the first line represents the following query Q_A over the Astronauts dataset.

```
SELECT * FROM Astronauts
WHERE "Space Walks" <= 3 AND "Space Walks" >= 1
AND "Graduate Major" = 'Physics'
ORDER BY "Space Flight (hrs)" DESC
```

This query may be used in the selection process of astronauts for a mission. The mission requires specific training (number of space walks) and background (graduate major), and the candidates are ordered by their experience (space flight hours). Similarly, the query Q_L for the Law Students dataset may be used to rank outstanding students (based on their GPA) from a particular region based on their SAT scores for a scholarship. Finally, Q_M is defined for the MEPS dataset. Such a query may be used to invite the best-fitting patients (based on their utilization) with specific criteria, for a study.

We defined result diversity constraints for each dataset (listed in Table 6). For instance, in the Astronauts dataset, the result should include women and candidates of varying ranks in the organizational hierarchy. The constraints' bounds are parameterized with a value k, and we set

[4] https://www.kaggle.com/datasets/nasa/astronaut-yearbook

[5] https://meps.ahrq.gov/data_stats/download_data/pufs/h192/h192doc.shtml

them to values that produce a valid refinement in most cases. Specifically, out of 132 performed experiments, we were not able to find a solution in only 2.

Parameters setting. When using ranked-retrieval in decision-making contexts (e.g., when deciding how many people to invite for in-person interviews), one expects the number of items a user will consider (k) to be relatively low. In general, rankings are subject to position bias — a geometric drop in visibility of items in lower ranks — and so are best-suited for cases where the user interacts with a small number of top-ranked items [2]. Thus, unless otherwise specified, we use $k = 10$ as a default value. Furthermore, we let the default maximum deviation ε be 0.5, aiming to strike a balance between being sufficiently close to the constraints but realistically possible in the datasets. In practice, this parameter may be chosen by specifying a worst-case scenario that is still acceptable, and then use the deviation of this scenario as calculated by Definition 2.6 to set ε. We also set the constraints set to include a single constraint (constraint (1) from Table 6 for each dataset). We used the three distance measures mentioned in Section 2.2: the queries predicates distance measure DIS_{pred} (abbr. QD in the figures), the Jaccard distance over the output, $DIS_{Jaccard}$ (JAC in the figures), and Kendall's τ, $DIS_{Kendall}$, for top-k lists defined in [17] (KEN in the figures).

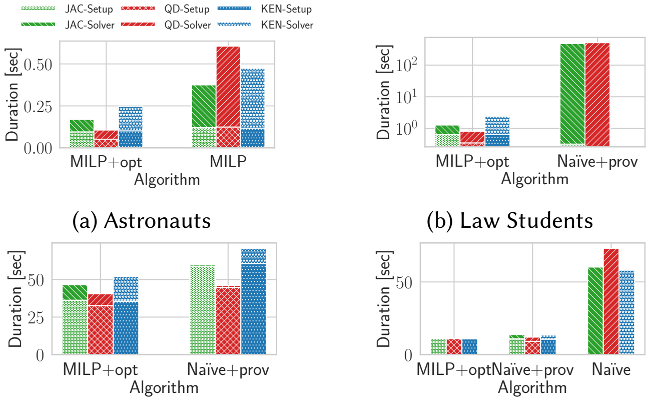
Compared algorithms. To our knowledge, our problem is novel and has no competing algorithms other than the naïve exhaustive search. Therefore, we compare our baseline MILP-based algorithm (MILP), our optimized MILP-based algorithm (MILP+opt), which includes the optimization described in Section 4, an exhaustive search over the space of refinements (Naïve), and a version the exhaustive search that uses our provenance annotations to evaluate the refinements (Naïve+prov). We report the total running time and show the setup time (constructing the MILP for MILP-based solutions, and generating the provenance for Naïve+prov). The MILP solver time is the gap between total and setup. The reported times are an average of 5 executions.

Platform & implementation details. Our experiments were performed on macOS 13.4 with an Apple M2 processor and 16 GB of memory. Our algorithm was implemented with IBM's CPLEX 22.1.1.0[6] to solve the mixed-integer linear program and DuckDB 0.8.0 [37] for query evaluation. The algorithm to construct the problem and the naïve method were written and evaluated with Python 3.9.6 and PuLP 2.7.0 (the library used for modeling the MILP problem). DIS_{pred} is linearized by computing the Jaccard distance for categorical predicates through the Charnes-Cooper transformation [11]. In addition, for numerical predicates, additional variables are generated that represent the absolute difference between the refined and original constants. As it does not consider the output, we skip generating s_t and $l_{t,k}$ variables for tuples that do not belong to any group G in C. $DIS_{Jaccard}$ is evaluated over the output, thus we leverage the fact that there are at least k^* tuples in the output and aim at maximizing the number of original tuples output, thereby maximizing the Jaccard distance. For $DIS_{Kendall}$, only Cases 2 (a tuple leaves the top-k) and 3 (a tuple enters the top-k) as defined in [17] may occur in our model. We create a variable for each case for each tuple, which is then equal to the sum of the case if the tuple is selected and zero otherwise. For more details, see [7, 8].

5.2 Results

Running time for compared algorithms. We begin by comparing the running time of all algorithms using the default parameters and setting a timeout of one hour. Recall that the size of the generated MILP program (without optimization) is linear in the data size and that MILP solvers are typically sensitive to the program size. Thus, we expect the MILP algorithm to struggle with large-scale datasets. On the other hand, the naïve approaches perform a brute-force search over the

[6] https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer



(a) Astronauts

(b) Law Students

(c) MEPS

(d) TPC-H

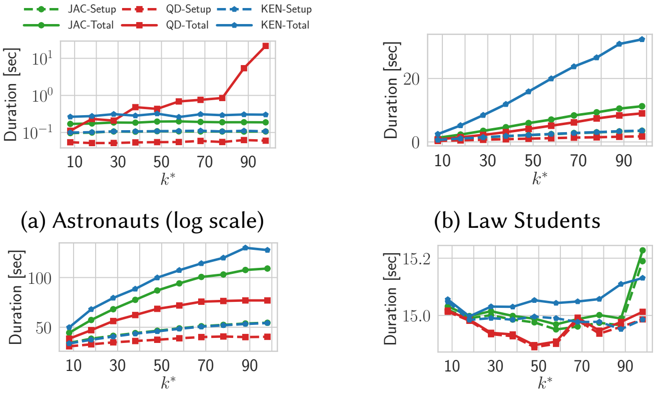
Fig. 3. Running time of compared algorithms, for cases where computation completed within a 1-hour timeout (method or distance omitted when timed out). MILP+opt consistently outperforms other methods.

possible refinements, where their number is exponential in the number of predicates in the query (and their domain). Thus, datasets with high cardinality in the domain of the query predicate are likely to be challenging for the naïve solutions.

Figure 3a presents performance for the Astronauts dataset. The optimized MILP solution outperforms the unoptimized MILP, and we observe a speedup of up to 6 times. Given that there are 114 different values for "Graduate Major", the space of refinements is extremely large and both Naïve and Naïve+prov timed out (and thus omitted from the graph). Figures 3b to 3d show the results for the remaining datasets. In these cases, due to the data size, the unoptimized MILP was unable to terminate before the time-out. MEPS and TPC-H have a relatively small space of refinements for the posed queries, making Naïve+prov competitive with MILP+opt. However, Law Students has a considerably larger space of refinements (although modest compared to Astronauts), making Naïve time out and Naïve+prov significantly slower than MILP+opt. Essentially, MILP+opt is well-posed to deal with scaling both the data size and the space of the possible refinements. The naïve brute-force search methods and unoptimized MILP method fail to scale, and we exclude them from the rest of the experiments.

Effect of k^*. We study the effect of k^*, the largest k with a constraint in the constraint set, on the running time of our algorithm by increasing the parameter k of the constraint from 10 to 100 in increments of 10. The results are presented in Figure 4. Recall that the relevancy-based optimization from Section 4 aims at reducing the program size using k^*. We expect to see its effect degrade as k^* increases, as shown in Figures 4b and 4c. The optimization is less effective for Astronauts (Figure 4a), as the number of different lineage equivalent classes is large, and each consists of a relatively small number of tuples (fewer than 10). Therefore, the expressions generated for very few tuples may be removed from the program. The optimization is particularly effective for Q_5 of TPC-H (Figure 4d), as the vast majority of expressions are removed as there are only 5 lineage equivalence classes. Moreover, we see that most of the time is spent setting up the problem rather than solving it.

Effect of maximum deviation (ε)*.* While an increase in ε may make finding feasible refinements easier, the solver must still find the minimal refinement, which remains a difficult task. Therefore,



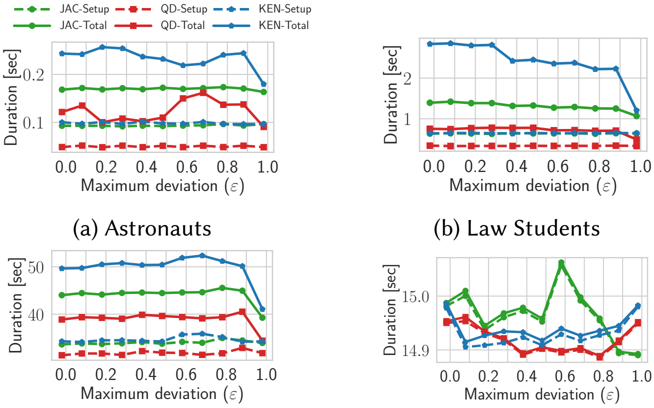
(a) Astronauts (log scale)

(b) Law Students

(c) MEPS

(d) TPC-H

Fig. 4. Running time vs. k^*, showing DIS_{pred} is often the fastest to compute, while $DIS_{Kendall}$ can be sensitive to increasing k^*.



(a) Astronauts

(b) Law Students

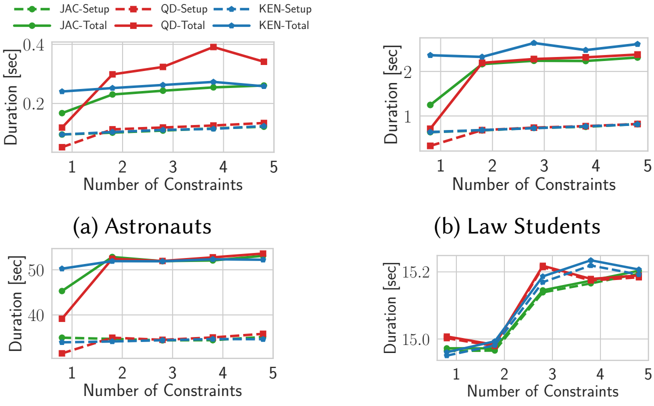
(c) MEPS

(d) TPC-H

Fig. 5. Running time vs. maximum deviation (ε), showing that the effect of ε is limited.

the value of ε should not significantly affect the running time. Figure 5 shows that the running time is fairly stable. We observed a decrease when ε reaches 1.0. This is because we use only lower-bound constraints in this experiment, where the deviation of any (refined) query is bounded by 1.0, i.e. finding a satisfying refinement is trivial as all refinements are good enough. In Figure 5d, the solver time is negligible and depends mostly on the setup time, which is very similar across all values of ε (differing by at most 1%).

Effect of constraint quantity. The number of generated expressions of the form (6) and (7) is linear in the number of constraints. Thus, when increasing the number of constraints, the program size increases and as a result, we expect to see an increase in the running time. We gradually added constraints to the constraint set in the order they listed in Table 6. To ensure that the set of constraints can be satisfied along with the default value of ε, we slightly adjust the value of the first two constraints for Astronauts, Law Students, and MEPS to have a lower-bound of $\frac{k}{3}$. As shown in



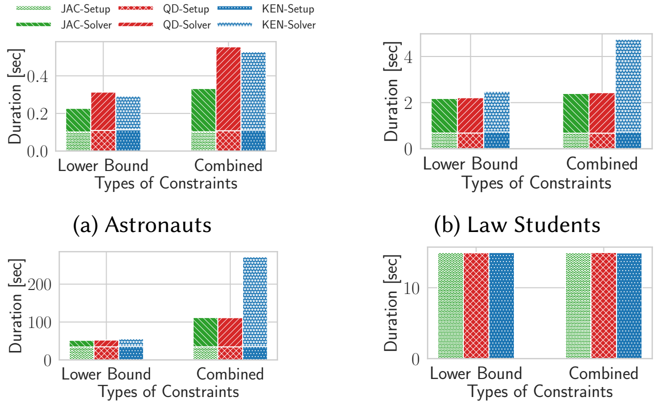
(a) Astronauts

(b) Law Students

(c) MEPS

(d) TPC-H

Fig. 6. Running time vs. the number of constraints: the impact of the number of constraints is limited.



(a) Astronauts

(b) Law Students

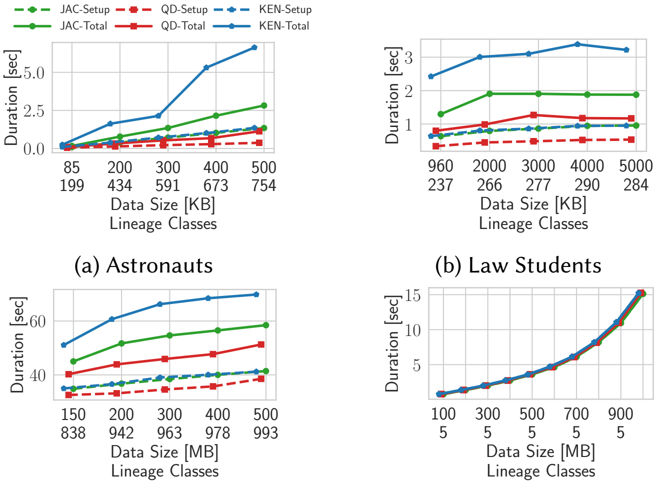
(c) MEPS

(d) TPC-H

Fig. 7. Running time vs. constraint type, showing the efficacy of one of our optimizations.

Figure 6, we observed a slight increase in the running time as the number of constraints increased in contrast to increasing the value of k. The number of expressions is linear in the number of constraints and tuples, however there are significantly fewer constraints than tuples, which is why the number of constraints does not have a pronounced effect on the runtime. TPC-H (Figure 6d) shows a negligible difference as the vast majority of the time is set up the MILP problem, as the solver has only 5 lineage equivalence classes to explore.

Effect of constraint types. In Section 4, we presented an optimization that is effective when tuples belong to groups with only either lower-bound or upper-bound constraints made on them. To demonstrate the effect of this optimization, we generate two sets of constraints for each dataset: C_L with lower bound constraints only, and C_M with a mixed set of upper bound and lower bound constraints. In particular, each dataset, C_L includes constraints (1) and (2) from Table 6, and C_M includes constraints (1) and (2), where constraint (2) is turned into an upper-bound constraint. Notice that these particular attributes are binary, and, as we assume there are at least k^* tuples in



(a) Astronauts

(b) Law Students

(c) MEPS

(d) TPC-H

Fig. 8. Running time vs. data size. The setup time is mostly impacted by the cost to capture lineage from the input query, while the solving time is mostly impacted by the number of lineage classes.

the output, the two constraints of C_M are equivalent (except for TPC-H, which lacks any binary attributes). We then compared the running time when using C_L and C_M (for which the optimization was disabled for C_M). The results are presented in Figure 7. As expected, the running times for the case of C_L are typically better, as shown in Figures 7a to 7c, indicating the usefulness of the optimization. We note that the experiment in Figure 7d depicting the experiment for TPC-H shares the same performance characteristics as the previous experiments.

Effect of dataset size. We use SDV [35] to synthesize scaled-up versions of the real datasets. Not only does this increase the data size, but new lineage classes are created according to the distribution of the dataset as well. For TPC-H, we generate different scales of the dataset according to its standard, but no new lineage classes are created. The number of variables and expressions of the generated MILP is linear in the number of tuples in the dataset. However, given that solving MILP is in NP, we expect a non-polynomial increase in running time with an increase in the data size. The results are plotted in Figure 8, each plot starting from the original size of the dataset. For the Astronauts, Law Students, and MEPS datasets, we observed a modest increase in the runtime as the data size grows. This could be explained by the low increase in the number of lineage classes, which impacts the efficiency of our optimization and has a greater effect on the running time. In TPC-H (Figure 8d), the vast majority of the running time is spent building the MILP problem, and in this case, constructing the set of lineages for Q5 involves a non-trivial amount of join processing.

Effect of chosen distance measure. We observed that in most cases, $DIS_{Kendall}$ is the hardest to compute as it involves extra variables in order to linearize the measure. When the refinement space is extremely large, such as for Astronauts, DIS_{pred} takes longer to prove optimality (as seen in Figures 4a and 6a).

5.3 Comparison with Erica [26, 27]

We conclude with a comparison to Erica [26, 27], which presents a similar framework for query refinements to satisfy cardinality constraints over groups representation in the query's output.

We note Erica focuses on cardinality constraints over the *entire* output, without considering the order of tuples. By restricting the overall output size to k, Erica may be used to refine a given query to satisfy constraints over the top-k tuples. However, as we next demonstrate, this additional constraint over the output size also limits the possible refinements to those that have at most k tuples. Moreover, this adjustment of Erica cannot be used to constrain over different values of k simultaneously (as in our running example). Additionally, since satisfying the constraints in our setting is more challenging, we focus on finding approximate solutions that are close to satisfying the constraints, while Erica only finds solutions that satisfy the constraints exactly. Finally, our framework allows the user to define different distance measures between queries whereas Erica uses a single distance measure based on the predicate distance. We compare the systems by refining the query Q_L except with the predicates `Region = ‘GL’ AND GPA >= 3.0`. subject to the singleton constraint set $C = \{\ell_{Sex='F',k=100} = 50\}$. To be consistent with [26, 27], we aim to minimize the predicate distance (using DIS_{pred} as the distance measure) and allow only results that satisfy the constraints exactly, i.e., $\varepsilon = 0$. When running Erica, we added a constraint requiring that exactly 100 results are returned to ensure the top-100 tuples contains at least 50 female candidates, and that there are enough results to satisfy the assumption in our problem definition. Using our optimized MILP-based approach, we were able to find a minimal refinement in ≈ 11 seconds. The refinement selects candidates from the regions `‘GL’` or `‘SC’` with a GPA of at least 4.0. Erica found 5 different refinements in ≈ 53 seconds, though none of them are closer to Q than the refinement found by our framework. In fact, all of the refinements require a GPA of at least 4.0 and select *3* regions. The refinement found by our system was not generated by Erica due to the additional constraint requiring the output size to be exactly 100.

6 RELATED WORK

Query refinements. The problem of query refinement has been addressed in previous studies such as [13, 24, 30, 34, 40, 41]. They focus on modifying queries to satisfy cardinality constraints, mostly emphasizing the overall output size rather than specific data groups within the output, and does not consider ranking of the results. For example, [24, 34] aim to relax queries with an empty result set to produce some answers. Other works like [13, 30] address the issues of too many or too few answers by refining queries to meet specific cardinality constraints on the result's size. A recent line of work has studied the use of refinement to satisfy diversity constraints [27, 32, 38]. The work of [38] aims to refine queries to satisfy constraints on the size of specific data groups in the result, however, they consider only numerical predicates with a single binary sensitive attribute. Closer to our work, Erica [26, 27, 32] utilizes provenance annotations to efficiently find minimal refinements. While our proposed solution is inspired by these works, their focus is on selection queries and can not be easily extended to ranking queries. Particularly, the provenance model used in these works is insufficiently expressive to capture the semantics of ranking, motivating our need to devise a new way to annotate and use these annotations to find the best approximation refinement. We discuss and demonstrate the differences in Section 5.3.

Constrained query answering. More generally, our problem answers queries that are subject to some set of constraints over the results. Systems like those proposed in [4, 6] allow querying groups of tuples that optimize some objective function while satisfying some constraints on the output, including cardinality constraints. However, they do not support top-k queries and therefore do not extend to the ranking setting. The work in [4] specifically relaxes the constraints of the problem to achieve partial satisfaction of the set of constraints, however it does so by removing constraints and not by modifying them as in our work. In [29], the authors develop a system to answer how-to queries. How-to queries answer how to modify the database in order to satisfy some constraints while optimizing for an objective. However, their system also lacks support for ranking, making it unsuitable to use for intervening on the top-k for various k values as in our framework.

Fairness in ranking. The problem we consider in this paper has implications in the context of fairness. Fairness in ranking has been the subject of much recent attention [1, 9, 10, 12, 20, 23, 45, 46, 48, 49]. These works can be categorized as post-processing methods (e.g., [10, 45, 46]) that directly modify the output rankings, or in-processing solutions [1, 9, 12, 20, 23] that adjust the ranking algorithm or modify items to produce a different score. Our solution can be considered as an in-processing method, however unlike existing solutions, we assume ranking algorithms and scores of different items are well-designed, and do not modify them.

Query result diversification. Query result diversification aims to increase result diversity while maintaining relevance of results to the original query by including or excluding tuples from the set of tuples in the result of the query output. [14, 19, 42]. Unlike our solution, the diversification is achieved by modifying the set of the tuples directly rather than the query, and does not consider tuples absent from the original query.

MILP & databases. Mixed-integer linear programming has been used in data management in order to solve relevant NP-hard optimization problems. However, as pointed out in [5, 28, 29, 43], scaling MILP problems to database-size problems is difficult. In order to scale, these works make several optimizations. In particular, the relevancy-based optimization we proposed resembles optimizations presented in [29, 43].

7 CONCLUSION

We identified a novel intervention to diversify (according to user-input constraints) the output of top-k queries by refining the selection predicates of the input query. Furthermore, we recognized the importance of maintaining the user's intent as best as possible when searching for such a refinement. Towards this end, we developed a framework that can find the closest refinement for various distance measures that satisfy the user's desired constraints. We introduced optimizations in order to make our framework practical for datasets of real-life scale. We demonstrated this with a suite of experiments, showing our framework's scaling capability and the usefulness of our optimizations. In the future, this problem could be extended to find refinements that remain diverse even after adding new data. This way, the refinement may explain some underlying bias of the query instead of fitting to the original data. Extending our model to richer classes of queries presents further interesting directions as we discussed in Section 3.

ACKNOWLEDGMENTS

This research was partially supported by ISF grant 2121/22, the Frankel Center for Computer Science at BGU, and NSF Awards No. 2326193 and 2312930.

REFERENCES

[1] 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 Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1259–1276. https://doi.org/10.1145/3299869.3300079

[2] Ricardo Baeza-Yates. 2018. Bias on the web. *Commun. ACM* 61, 6 (2018), 54–61. https://doi.org/10.1145/3209581

[3] Pierre Bourhis, Daniel Deutch, and Yuval Moskovitch. 2016. Analyzing data-centric applications: Why, what-if, and how-to. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*. IEEE Computer Society, 779–790. https://doi.org/10.1109/ICDE.2016.7498289

[4] Matteo Brucato, Azza Abouzied, and Alexandra Meliou. 2014. Improving package recommendations through query relaxation. In *Proceedings of the First International Workshop on Bringing the Value of "Big Data" to Users, Data4U@VLDB*

2014, Hangzhou, China, September 1, 2014, Rada Chirkova and Jun Yang (Eds.). ACM, 13. https://doi.org/10.1145/2658840.2658843

[5] Matteo Brucato, Azza Abouzied, and Alexandra Meliou. 2018. Package queries: efficient and scalable computation of high-order constraints. *VLDB J.* 27, 5 (2018), 693–718. https://doi.org/10.1007/s00778-017-0483-4

[6] Matteo Brucato, Rahul Ramakrishna, Azza Abouzied, and Alexandra Meliou. 2015. PackageBuilder: From Tuples to Packages. *CoRR* abs/1507.00942 (2015). arXiv:1507.00942 http://arxiv.org/abs/1507.00942

[7] Felix S. Campbell, Alon Silberstein, Julia Stoyanovich, and Yuval Moskovitch. 2024. Query Refinement for Diverse Top-k Selection (Implementation). https://github.com/fsalc/diverse-top-k

[8] Felix S. Campbell, Alon Silberstein, Julia Stoyanovich, and Yuval Moskovitch. 2024. Query Refinement for Diverse Top-k Selection (Tech Report). arXiv:2403.17786 [cs.DB]

[9] L. Elisa Celis, Anay Mehrotra, and Nisheeth K. Vishnoi. 2020. Interventions for ranking in the presence of implicit bias. In *FAT* '20: Conference on Fairness, Accountability, and Transparency, Barcelona, Spain, January 27-30, 2020*, Mireille Hildebrandt, Carlos Castillo, L. Elisa Celis, Salvatore Ruggieri, Linnet Taylor, and Gabriela Zanfir-Fortuna (Eds.). ACM, 369–380. https://doi.org/10.1145/3351095.3372858

[10] L. Elisa Celis, Damian Straszak, and Nisheeth K. Vishnoi. 2018. Ranking with Fairness Constraints. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic (LIPIcs, Vol. 107)*, Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 28:1–28:15. https://doi.org/10.4230/LIPIcs.ICALP.2018.28

[11] Abraham Charnes and William W Cooper. 1962. Programming with linear fractional functionals. *Naval Research Logistics Quarterly* 9, 3-4 (1962), 181–186.

[12] Zixuan Chen, Panagiotis Manolios, and Mirek Riedewald. 2023. Why Not Yet: Fixing a Top-k Ranking that Is Not Fair to Individuals. *Proc. VLDB Endow.* 16, 9 (2023), 2377–2390. https://www.vldb.org/pvldb/vol16/p2377-chen.pdf

[13] Wesley W. Chu and Qiming Chen. 1994. A structured approach for cooperative query answering. *IEEE Transactions on Knowledge and Data Engineering* 6, 5 (1994), 738–749.

[14] Ting Deng and Wenfei Fan. 2014. On the Complexity of Query Result Diversification. *ACM Trans. Database Syst.* 39, 2 (2014), 15:1–15:46. https://doi.org/10.1145/2602136

[15] Daniel Deutch, Zachary G. Ives, Tova Milo, and Val Tannen. 2013. Caravan: Provisioning for What-If Analysis. In *Sixth Biennial Conference on Innovative Data Systems Research, CIDR 2013, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2013/Papers/CIDR13_Paper100.pdf

[16] Daniel Deutch, Yuval Moskovitch, and Val Tannen. 2014. A Provenance Framework for Data-Dependent Process Analysis. *Proc. VLDB Endow.* 7, 6 (2014), 457–468. https://doi.org/10.14778/2732279.2732283

[17] Ronald Fagin, Ravi Kumar, and D. Sivakumar. 2003. Comparing Top k Lists. *SIAM J. Discret. Math.* 17, 1 (2003), 134–160. https://doi.org/10.1137/S0895480102412856

[18] Sahin Cem Geyik, Stuart Ambler, and Krishnaram Kenthapadi. 2019. Fairness-Aware Ranking in Search & Recommendation Systems with Application to LinkedIn Talent Search. In *SIGKDD*. ACM.

[19] Sreenivas Gollapudi and Aneesh Sharma. 2009. An axiomatic approach for result diversification. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl (Eds.). ACM, 381–390. https://doi.org/10.1145/1526709.1526761

[20] Md Mouinul Islam, Dong Wei, Baruch Schieber, and Senjuti Basu Roy. 2022. Satisfying Complex Top-k Fairness Constraints by Preference Substitutions. *Proc. VLDB Endow.* 16, 2 (2022), 317–329. https://www.vldb.org/pvldb/vol16/p317-roy.pdf

[21] Richard M. Karp. 1972. Reducibility Among Combinatorial Problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA (The IBM Research Symposia Series)*, Raymond E. Miller and James W. Thatcher (Eds.). Plenum Press, New York, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9

[22] M. G. Kendall. 1938. A New Measure of Rank Correlation. *Biometrika* 30, 1-2 (06 1938), 81–93. https://doi.org/10.1093/biomet/30.1-2.81 arXiv:https://academic.oup.com/biomet/article-pdf/30/1-2/81/423380/30-1-2-81.pdf

[23] Jon M. Kleinberg and Manish Raghavan. 2018. Selection Problems in the Presence of Implicit Bias. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA (LIPIcs, Vol. 94)*, Anna R. Karlin (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 33:1–33:17. https://doi.org/10.4230/LIPIcs.ITCS.2018.33

[24] Nick Koudas, Chen Li, Anthony K. H. Tung, and Rares Vernica. 2006. Relaxing Join and Selection Queries. In *VLDB*.

[25] Matt J. Kusner, Joshua R. Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual Fairness. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 4066–4076. https://proceedings.neurips.cc/paper/2017/hash/a486cd07e4ac3d270571622f4f316ec5-Abstract.html

[26] Jinyang Li, Yuval Moskovitch, Julia Stoyanovich, and HV Jagadish. 2023. Query Refinement for Diversity Constraint Satisfaction. *Proceedings of the VLDB Endowment* 17, 2 (2023), 106–118.

[27] Jinyang Li, Alon Silberstein, Yuval Moskovitch, Julia Stoyanovich, and H. V. Jagadish. 2023. Erica: Query Refinement for Diversity Constraint Satisfaction. *Proc. VLDB Endow.* 16, 12 (2023), 4070–4073. https://doi.org/10.14778/3611540.3611623

[28] Anh L. Mai, Pengyu Wang, Azza Abouzied, Matteo Brucato, Peter J. Haas, and Alexandra Meliou. 2023. Scaling Package Queries to a Billion Tuples via Hierarchical Partitioning and Customized Optimization. *CoRR* abs/2307.02860 (2023). https://doi.org/10.48550/arXiv.2307.02860 arXiv:2307.02860

[29] 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, Scottsdale, AZ, USA, May 20-24, 2012*, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman (Eds.). ACM, 337–348. https://doi.org/10.1145/2213836.2213875

[30] Chaitanya Mishra and Nick Koudas. 2009. Interactive query refinement. In *EDBT 2009, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings (ACM International Conference Proceeding Series, Vol. 360)*, Martin L. Kersten, Boris Novikov, Jens Teubner, Vladimir Polutin, and Stefan Manegold (Eds.). ACM, 862–873. https://doi.org/10.1145/1516360.1516459

[31] Yuval Moskovitch, Jinyang Li, and H. V. Jagadish. 2022. Bias analysis and mitigation in data-driven tools using provenance. In *Proceedings of the 14th International Workshop on the Theory and Practice of Provenance, TaPP 2022, Philadelphia, Pennsylvania, 17 June 2022.* ACM, 1:1–1:4. https://doi.org/10.1145/3530800.3534528

[32] Yuval Moskovitch, Jinyang Li, and H. V. Jagadish. 2022. Bias analysis and mitigation in data-driven tools using provenance. In *Proceedings of the 14th International Workshop on the Theory and Practice of Provenance, TaPP 2022, Philadelphia, Pennsylvania, 17 June 2022*, Adriane Chapman, Daniel Deutch, and Tanu Malik (Eds.). ACM, 1:1–1:4. https://doi.org/10.1145/3530800.3534528

[33] Yuval Moskovitch, Jinyang Li, and H. V. Jagadish. 2023. Detection of Groups with Biased Representation in Ranking. *CoRR* abs/2301.00719 (2023). https://doi.org/10.48550/arXiv.2301.00719 arXiv:2301.00719

[34] Ion Muslea and Thomas J Lee. 2005. Online query relaxation via bayesian causal structures discovery. In *AAAI*. 831–836.

[35] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. 2016. The Synthetic data vault. In *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. 399–410. https://doi.org/10.1109/DSAA.2016.49

[36] Christopher Peskun, Allan Detsky, and Maureen Shandling. 2007. Effectiveness of medical school admissions criteria in predicting residency ranking four years later. *Medical education* 41, 1 (2007).

[37] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1981–1984. https://doi.org/10.1145/3299869.3320212

[38] Suraj Shetiya, Ian P. Swift, Abolfazl Asudeh, and Gautam Das. 2022. Fairness-Aware Range Queries for Selecting Unbiased Data. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022.* IEEE, 1423–1436. https://doi.org/10.1109/ICDE53745.2022.00111

[39] Julia Stoyanovich, Ke Yang, and H. V. Jagadish. 2018. Online Set Selection with Fairness and Diversity Constraints. In *Proceedings of the 21st International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018*, Michael H. Böhlen, Reinhard Pichler, Norman May, Erhard Rahm, Shan-Hung Wu, and Katja Hose (Eds.). OpenProceedings.org, 241–252. https://doi.org/10.5441/002/edbt.2018.22

[40] Quoc Trung Tran and Chee-Yong Chan. 2010. How to conquer why-not questions. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data.* 15–26.

[41] Quoc Trung Tran, Chee-Yong Chan, and Srinivasan Parthasarathy. 2009. Query by output. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data.* 535–548.

[42] Marcos R. Vieira, Humberto Luiz Razente, Maria Camila Nardini Barioni, Marios Hadjieleftheriou, Divesh Srivastava, Caetano Traina Jr., and Vassilis J. Tsotras. 2011. On query result diversification. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan (Eds.). IEEE Computer Society, 1163–1174. https://doi.org/10.1109/ICDE.2011.5767846

[43] Xiaolan Wang, Alexandra Meliou, and Eugene Wu. 2017. QFix: Diagnosing Errors through Query Histories. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 1369–1384. https://doi.org/10.1145/3035918.3035925

[44] Linda F Wightman. 1998. LSAC National Longitudinal Bar Passage Study. LSAC Research Report Series. (1998).

[45] Ke Yang, Vasilis Gkatzelis, and Julia Stoyanovich. 2019. Balanced Ranking with Diversity Constraints. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*,

Sarit Kraus (Ed.). ijcai.org, 6035–6042. https://doi.org/10.24963/ijcai.2019/836

[46] Ke Yang and Julia Stoyanovich. 2017. Measuring Fairness in Ranked Outputs. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Chicago, IL, USA, June 27-29, 2017*. ACM, 22:1–22:6. https://doi.org/10.1145/3085504.3085526

[47] Meike Zehlike, Philipp Hacker, and Emil Wiedemann. 2020. Matching code and law: achieving algorithmic fairness with optimal transport. *Data Min. Knowl. Discov.* 34, 1 (2020), 163–200. https://doi.org/10.1007/s10618-019-00658-8

[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

Received October 2023; revised January 2024; accepted February 2024