RODEO: Making Refinements for Diverse Top-k Queries

Felix S. Campbell
Ben-Gurion University of the Negev
felixsal@post.bgu.ac.il

Julia Stoyanovich
New York University
stoyanovich@nyu.edu

Yuval Moskovitch
Ben-Gurion University of the Negev
yuvalmos@bgu.ac.il

ABSTRACT

Database queries are commonly used to select and rank items. With the increasing awareness of diversity, ensuring a diverse output (i.e., the representation of different groups in the top-k positions) becomes essential. To address this challenge, we present RODEO, a system that generates minimal modifications to queries to enhance the diversity of the ranking they produce based on constraints over groups' representation in the top-k for various k values.

PVLDB Reference Format:
Felix S. Campbell, Julia Stoyanovich, and Yuval Moskovitch. RODEO: Making Refinements for Diverse Top-k Queries. PVLDB, 17(12): 4341 - 4344, 2024.
doi:10.14778/3685800.3685870

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/fsalc/rodeo.

1 INTRODUCTION

Decision-making based on ranking is commonly used in important domains, such as hiring and school admission. It usually includes choosing qualifying candidates based on specific criteria (*e.g.,* for a job) and ranking them using a quantitative measure to identify the top candidates among those who qualify (*e.g.,* for a job interview). This process can be automated and represented using SQL queries, with the `WHERE` clause used to select candidates who fulfill specific criteria and the `ORDER BY` clause used to rank them. Further, it is often required to impose *cardinality constraints*, over *both* the number of candidates to return (*e.g.,* $k = 10$ job applicants to interview) and the presence of some demographic groups among the top-k candidates. Let us consider an example.

Example 1.1. Consider the hypothetical STEM-Stars foundation that wishes to grant university scholarships to high-performing students. They use a database of students seeking scholarships provided by their schools, such as the one in Table 1. Students have 6 attributes: a unique ID, gender, family income, grade point average (GPA), SAT score, and extracurricular activity (one of robotics (*RB*), Science Olympiad (*SO*), Math Olympiad (*MO*), game development (*GD*), and STEM tutoring (*TU*)).

STEM-Stars would like to award scholarships to strong students (GPA above some threshold) who also participate in extracurriculars. The selected students are then ranked by their SAT scores, and the foundation grants funding to the best six, with additional funding going to the top three students. Further, STEM-Stars wishes to promote female students by awarding a proportional number of scholarships by gender (*i.e.,* top-6 should include at least three women). Finally, to expand access to STEM education, they limit the number of scholarships to students from high-income families (*i.e.,* the top-3 can include at most one high-income student).

Table 1: `Students`

ID	Gender	Income	GPA	Activity	SAT ↓
t_1	M	Medium	3.7	SO	1590
t_2	F	Low	3.8	SO	1580
t_3	F	Low	3.6	GD	1570
t_4	M	High	3.8	RB	1560
t_5	F	Medium	3.6	MO	1550
t_6	F	Low	3.7	SO	1550
t_7	M	Low	3.7	RB	1540
t_8	F	High	3.9	RB	1530
t_9	M	High	3.7	RB	1520
t_{10}	F	Low	3.8	RB	1490
t_{11}	M	Medium	4.0	RB	1480
t_{12}	F	Low	3.7	RB	1410

To start, a STEM-Stars data analyst expresses the requirements using query Q that selects students who have participated in extracurricular robotics and have a GPA of at least 3.7:

```
Q: SELECT * FROM Students
   WHERE GPA >= 3.7 AND Activity = 'RB'
   ORDER BY SAT DESC
```

Evaluating this query over the datasets in Table 1 produces the ranking $[t_4, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}]$. Observe that this result does not satisfy the diversity constraints, because it includes only 2 women (t_8 and t_{10}) and more than 1 high-income student (t_4 and t_8).

To fulfill the desired constraints, STEM-Stars 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, modifying the results after they were computed may raise a procedural fairness concern. Moreover, it may explicitly use information about demographic or otherwise protected group membership, raising a disparate treatment concern. Alternatively, Q may be refined by adjusting the condition on `Activity` to include students involved in Science Olympiad (*SO*), resulting in the following query Q':

```
Q': SELECT * FROM Students
    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. In general, modifying the selection conditions may be desirable as they could themselves be the source of bias in the results. Furthermore, this approach is usually legally permissible, essentially because it applies the same evaluation process to all individuals.

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685870

Contributions. To facilitate such a process, we present Rodeo (Refinements fOr DivErse tOp-k queries), a system for refining queries to improve the diversity of the ranking they produce. Rodeo is an interactive system, allowing users to define queries and constraints over the top-k tuples for different values of k in their output using a dedicated interface.

Rodeo complements a recently published paper [3] by implementing the methods proposed in that paper in an interactive system. 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, Rodeo focuses on finding a refined query that is within a specified maximum distance from satisfying all of the constraints, if one exists. The system implements a solution based on a reduction to mixed-integer linear programming (MILP). Given an input, Rodeo formulates an MILP instance and utilizes a state-of-the-art solver to derive a solution. We will demonstrate the usefulness of Rodeo using three real-life datasets, allowing the audience to formulate their own queries and constraints and interact with Rodeo to generate refinements.

Related work. The notion of refining queries to satisfy a set of diversity constraints was recently presented in [10, 11, 13], however, this work focuses on constraints over the *entire* output, does not consider the order of tuples, and cannot be easily extend to ranking queries. The problem of ensuring diverse outputs in ranking queries has received much recent attention from the research community [1, 4, 5, 9, 16, 17]. These works can be categorized as post-processing methods (*e.g.,* [5, 16, 17]) that directly modify the output, or in-processing methods [1, 4, 9] 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 that the ranking algorithm and the scoring function are well-designed, and do not modify them.

2 TECHNICAL BACKGROUND

We next informally introduce the model and algorithms underlying Rodeo. Please see Campbell et al. [3] for more details.

2.1 Model

Queries and refinements. We support the class of conjunctive Select-Project-Join (SPJ) queries which have an `ORDER BY` s clause where s is some function mapping a single tuple t to a real-valued score. A query may have *numerical* or *categorical predicates* which filter its input to tuples matching these predicates. A numerical predicate has the form $A \diamond C$ where $\diamond \in \{<, \leq, =, >, \geq\}$, A is some attribute, and $C \in \mathbb{R}$. A categorical predicate has the form $\bigvee_{c \in C} A = c$ where A is some attribute and C is a subset of constants of the domain of A. We use the notion of query refinement defined in [12]. For a numerical predicate, refinements are changes to the value of the constant; for categorical predicates, a refinement is done by adding or removing values from the original constant list. We say that a query Q' is a *refinement* of query Q if Q' is obtained from Q by refining some predicates of Q.

Constraints. A *group* is a set of tuples all sharing the same value for a given categorical attribute, *e.g.,* `Gender` $= F$ is the group of females and `Gender` $= F \wedge$ `Income` $= Low$ is the group of females with a low-income level. Imposing constraints on the cardinality of tuples belonging to certain groups may be used to improve the diversity of the outputs of queries, and such methods have been studied in [10, 11, 13]. In the ranking setting, instead of specifying these constraints over the entire output, we may specify these constraints over the top-k tuples for multiple different values of k. 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 on the user's preferences, as we next demonstrate.

Example 2.1. Recall Q from Example 1.1. We may refine it by relaxing the GPA requirement to 3.6 and including students who participated in game development (GD), obtaining query Q'':

```
Q'': SELECT * FROM Students
     WHERE GPA >= 3.6 AND (Activity = 'RB' OR Activity = 'GD')
     ORDER BY SAT DESC
```

Similarly to the refined query Q', which adjust the condition on `Activity` to include students involved in Science Olympiad (SO), the top-6 students selected by Q'' (t_3, t_4, t_7, t_8, t_9, and t_{10}) include three women (t_3, t_8, and t_{10}), and there is only one high-income student (t_4) among the top-3. While the predicates of Q'' are intuitively more distant from the original query than Q' (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 alternative query refinement objectives, as illustrated above, Rodeo supports three distance measures to determine the minimality of a refinement query.

(i) *Queries predicate distance.* The first distance measure considers the difference between the predicates of the query Q and a refinement Q', thus preserving the essence of the original query Q. For each numerical predicate $n_Q = A \diamond C$ it computes the distance $|n_Q.C - n_{Q'}.C|$, where $n_Q.C$ is the value of C in n_Q and $n_{Q'}.C$ is the value of C in Q'. The distances between all numerical predicates are (normalized and) aggregated as $\sum_{n_Q} \frac{|n_Q.C - n_{Q'}.C|}{n_Q.C}$. The distance between categorical predicates is measured using Jaccard distance. These values are then summed to formulate a distance measure.

(ii) *Output set distance.* An alternative distance measure Rodeo supports considers the output of the query Q and a refinement Q' over the dataset D using Jaccard distance over the top-k tuples.

(iii) *Output order distance.* Finally, Rodeo can take into account the order of tuples in top-k item in the output to measure the distance between queries using (a variant of) Kendall's τ [8] that was proposed by Fagin et al. [7] to compare ranked lists.

Given a query Q over a dataset D, along with a set of cardinality constraints and a distance measure, the goal is to find a refinement with minimal distance from Q while satisfying the constraints. However, a refinement satisfying the constraints exactly may not always exist. We therefore relax the requirement for strict constraint satisfaction and instead focus on finding a refinement that comes close to satisfying the constraints, *i.e.,* within an acceptable distance. This 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, we leverage the notion of the *mean absolute percentage error*, as was done in [2], but modify its definition as to not penalize some cases that are acceptable in our setting.

2.2 MILP-based solution

Rodeo implements a mixed-integer linear program (MILP) solution to generate refinements. Mixed-integer linear programming is a model for optimizing a linear objective function subject to a set of expressions (equalities and inequalities) limiting the space of feasible assignments. At a high level, Rodeo utilizes data annotations to depict potential refinements. These annotations serve as variables in the MILP instance and enable the system to quantify the deviation from the constraint set without having to reevaluate refinements across the DBMS. A MILP instance solution is an assignment to the variables in the expressions that corresponds to a minimal refinement. While solving MILP instances can be expensive, we introduce a number of optimizations to make solving instances created from datasets of real-life scale efficient in practice. We next briefly explain the MILP instance construction using our running example, see [3] for details.

Given a query Q and a dataset D, for each categorical predicate over an attribute A in Q, we define a variable $A_v \in \{0, 1\}$ for each value v in the domain of A in the D. For instance, the variable $Activity_{SO}$ is generated in our running example. Intuitively, a solution to the MILP where $A_v = 1$ corresponds to a refinement that includes the value v in the categorical predicate over A. For each numerical predicate $A \diamond C$, we define a variable $C_{A,\diamond}$, and a set of variables $A_{v,\diamond}$ for each value v in the domain of A in D. 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 (*i.e.,* whether $v \diamond C_{A,\diamond}$). In our example, the variables $C_{GPA,\geq}$ and $GPA_{3.8,\geq}$ are generated. We then use these annotations to formulate expressions in the MILP instance.

Example 2.2. Tuples with `GPA = 3.8` satisfy a refinement's constraint $GPA \geq C$ if the value of C is at most 3.8. Namely, $GPA_{3.8,\geq} = 1 \Leftrightarrow 3.8 \geq C_{GPA,\geq}$ and 0 otherwise. We model this using the following two expression: $C_{GPA,\geq} + 5 \cdot GPA_{3.8,\geq} \geq 3.801$ and $C_{GPA,\geq} - 5 \cdot (1 - GPA_{3.8,\geq}) \leq 3.8$. We add a small constant to 3.8 in the first expression in order to relax the inequality as MILP expressions do not support strict inequalities. $GPA_{3.8,\geq}$ is multiplied by 5—a value larger than the possible values of $C_{GPA,\geq}$ (a constant known as "big M" in MILP modeling). Intuitively, the first expression ensures that $GPA_{3.8,\geq}$ is 1 if $3.8 \geq C_{GPA,\geq}$ and the second expression is used to ensure that $GPA_{3.8,\geq}$ is 0 if $3.8 \ngeq C_{GPA,\geq}$.

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. We utilize the data annotations to determine whether a tuple satisfies the selection conditions of the refinement. Consider for instance the tuple t_2 in Table 1. Intuitively, assignment of 1 to the variables $Activity_{SO}$ and $GPA_{3.8,\geq}$ corresponds to a refinement that includes $Activity = SO$ in the categorical predicates and that $3.8 \geq C_{GPA,\geq}$, namely, t_2 satisfies its selection conditions. We construct the expression $0 \leq Activity_{SO} + GPA_{3.8,\geq} - 2 \cdot r_{t_2} \leq 1$ to model this. The position of t_2 in the ranking depends on the existence of tuples ranked better than it in the output. This is modeled using the expression $1 + 12 \cdot (1 - r_{t_2}) + r_{t_1} = s_{t_2}$. In the case of t_2, its position depends on the existence of t_1 in the output (represented with r_{t_1} in the expression). If t_2 is not in the output, we set s_{t_2} to a value larger than a refinement's largest possible output size (12 in our example). *I.e.*, if $r_{t_2} = 0$ then $s_{t_2} = 13$.

For each constraint on the top-k over a group G, we use a variable $l_{t,k}$ to indicate whether a tuple t appears in the top-k ranked output of the corresponding refinement query, and $E_{G,k}$ to represent the number of tuples from G in the top-k that need to be added (or removed) to satisfy the constraint. In our example, $l_{t_2,6}$ is used to indicate whether t_2 is in the top-6 tuples, and we construct the expression $s_{t_2} + 25 \cdot l_{t_6,6} \geq 6.001$ and $s_{t_2} - 25 \cdot (1 - l_{t_6,6}) \leq 6$ to model $l_{t_2,6} = 1 \Leftrightarrow 6 \geq s_{t_2}$ and 0 otherwise, similarly to Example 2.2. $l_{t_2,6}$ is then used to determine the value of $E_{\texttt{Gender='Female'},6}$ which corresponds to how many additional women should appear in the top-6: $E_{Gender='Female',6} \geq 0$ and $E_{Gender='Female',6} \geq 3 - (l_{t_2,6} + l_{t_3,6} + l_{t_5,6} + l_{t_6,6} + l_{t_8,6} + l_{t_{10},6} + l_{t_{12},6})$. Note that $t_2, t_3, t_5, t_6, t_8, t_{10}$ and t_{12} are all the tuples of the group `Gender = 'Female'`. Finally, we combine the $E_{G,k}$ values in an expression that bounds their average by the maximum average deviation specified in the input.

3 SYSTEM OVERVIEW

Users interact with Rodeo using a dedicated web-based user interface implemented with React[1]. Rodeo's back-end is implemented in Python. It uses DuckDB [14] and the IBM CPLEX [2] MILP solver.

Input configuration. The input generation is done in three steps. The first step includes the data selection and query definition. Rodeo offers an easy-to-use interface to build queries by defining conditions over the attributes and specifying an expression that is used to rank the tuples. The generated query is serialized to SQL and displayed to the user. The next step includes the definition of cardinality constraints. The user can define groups by selecting attributes and choosing a value from within their domain in the database. For each constraint, the user needs to indicate a k value and whether it is a lower or upper bound constraint. In order to aid the user in choosing reasonable constraints, Rodeo provides a sketch of the distribution in the database for attributes where constraints are desirable. In the final step, the user configures the maximum average deviation from the set of cardinality constraints and sets the desired distance metric as shown in Figure 1. In order to understand the impact of the deviation, Rodeo displays the effect on the possible number of tuples in each group in the constraint list in the refined query.

Output overview. Upon submitting a request for refinement, the system communicates with the DBMS for query evaluation and the data annotation generation for the MILP instance. Rodeo constructs the MILP instance and utilizes the MILP solver to find a minimal assignment and returns the corresponding refinement. After the request for refinement has been processed by the back-end, the results are presented to the user as depicted in Figure 2. A number of features have been developed to assist the user in interpreting

[1] https://react.dev
[2] https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer

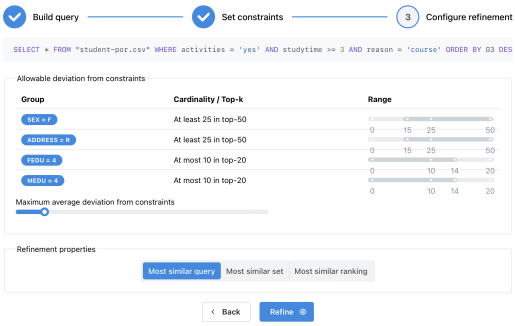

Figure 1: Input screen

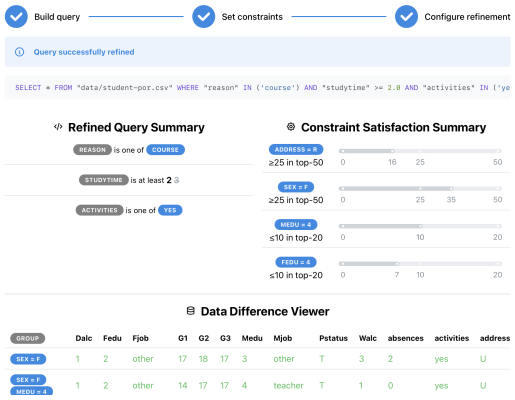

Figure 2: Displaying results of the refinement process

the result, including highlighting tuples added/removed, displaying the membership of tuples to relevant groups, and a depiction of how close each constraint is to being satisfied.

4 DEMONSTRATION

We will demonstrate the functionality of Rodeo using three real-life datasets: (1) **High-School Students** [6], consisting of 649 students from two Portuguese high schools with demographic information, grades, and social life data; (2) **Astronauts**[3], a dataset with background and careers information of 357 NASA astronauts; and (3) **Law Students** [15], containing demographic information and performance of 21,790 law students. We will let the audience specify queries, constraints, and deviation values and observe refinements produced by the system. The users will be able to interact with Rodeo to explore the results of the refinement and modify different steps of the pipeline to understand their effect on the output.

Demonstration Scenario. Let us consider the High-School Students dataset and play the role of STEM-Stars, selecting students to receive scholarships, as discussed in Example 1.1 in the Introduction. We define three requirements to be considered for the scholarship: *(i)* participation in extracurricular activities, *(ii)* at least 5 hours of weekly study time, and *(iii)* selecting their school for its course offerings (as opposed to, *e.g.,* location). We rank the students based on their final grades and grant scholarships to the top-50, and extended scholarships to the top-20. We then define constraints. To ensure equal access to higher educational opportunities, we aim to allocate half of the extended grants to students with a parent lacking higher education and half of the total scholarships to rural and/or female students (as shown in Figure 1).

The query derived from our original requirements does not satisfy the constraints and so we use Rodeo to refine it. We first consider the most-similar-query distance measure and try to refine the query. The system indicates that no refinement can perfectly satisfy the constraints. We may compromise on the constraint satisfaction by increasing the maximum average deviation to obtain refinements that are close to satisfying them. By doing so, Rodeo is able to produce a refinement which reduces the study time to at least 2 hours of weekly study time (the result shown in Figure 2).

While this query is close to the original query, we observe that ten students who were originally slated to receive scholarships no longer would. We can ask Rodeo for a refinement whose output is most similar to the original, and obtain one that differs by only one student compared with the original query. This refinement is similar to the original query, omitting the requirement to participate in extracurricular activities. Interacting with Rodeo helps the user find a refinement that best aligns with their preferences and evaluate the trade-off between query refinement and constraint satisfaction.

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

ACKNOWLEDGMENTS

This research was partially supported by the Israel Science Foundation 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 *ACM SIGMOD*.
[2] Matteo Brucato, Azza Abouzied, and Alexandra Meliou. 2014. Improving package recommendations through query relaxation. In *Data4U@VLDB*. ACM.
[3] Felix S. Campbell, Alon Silberstein, Julia Stoyanovich, and Yuval Moskovitch. 2024. Query Refinement for Diverse Top-k Selection. *Proc. ACM Manag. Data* 2, 3, Article 166 (May 2024), 27 pages. https://doi.org/10.1145/3654969
[4] L. Elisa Celis, Anay Mehrotra, and Nisheeth K. Vishnoi. 2020. Interventions for ranking in the presence of implicit bias. In *FAT* '20*. ACM.
[5] L. Elisa Celis, Damian Straszak, and Nisheeth K. Vishnoi. 2018. Ranking with Fairness Constraints. In *ICALP (LIPIcs)*, Vol. 107.
[6] Paulo Cortez. 2014. Student Performance. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5TG7T.
[7] Ronald Fagin, Ravi Kumar, and D. Sivakumar. 2003. Comparing Top k Lists. *SIAM J. Discret. Math.* 17, 1 (2003).
[8] M. G. Kendall. 1938. A New Measure of Rank Correlation. *Biometrika* 30, 1-2 (1938).
[9] Jon M. Kleinberg and Manish Raghavan. 2018. Selection Problems in the Presence of Implicit Bias. In *ITCS (LIPIcs)*, Vol. 94.
[10] Jinyang Li, Yuval Moskovitch, Julia Stoyanovich, and HV Jagadish. 2023. Query Refinement for Diversity Constraint Satisfaction. *PVLDB* 17, 2 (2023).
[11] Jinyang Li, Alon Silberstein, Yuval Moskovitch, Julia Stoyanovich, and H. V. Jagadish. 2023. ERICA: Query Refinement for Diversity Constraint Satisfaction. *PVLDB* 16, 12 (2023).
[12] Chaitanya Mishra and Nick Koudas. 2009. Interactive query refinement. In *EDBT*.
[13] Yuval Moskovitch, Jinyang Li, and H. V. Jagadish. 2022. Bias analysis and mitigation in data-driven tools using provenance. In *TaPP*. ACM.
[14] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *SIGMOD*. ACM.
[15] Linda F Wightman. 1998. LSAC National Longitudinal Bar Passage Study. LSAC Research Report Series. (1998).
[16] Ke Yang, Vasilis Gkatzelis, and Julia Stoyanovich. 2019. Balanced Ranking with Diversity Constraints. In *IJCAI*, Sarit Kraus (Ed.). ijcai.org.
[17] Ke Yang and Julia Stoyanovich. 2017. Measuring Fairness in Ranked Outputs. In *SSDBM*. ACM.