Query Refinement for Diversity Constraint Satisfaction

Jinyang Li
jinyli@umich.edu
University of Michigan

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

Julia Stoyanovich
stoyanovich@nyu.edu
New York University

H. V. Jagadish
jag@umich.edu
University of Michigan

ABSTRACT

Diversity, group representation, and similar needs often apply to query results, which in turn require constraints on the sizes of various subgroups in the result set. Traditional relational queries only specify conditions as part of the query predicate(s), and do not support such restrictions on the output. In this paper, we study the problem of modifying queries to have the result satisfy constraints on the sizes of multiple subgroups in it. This problem, in the worst case, cannot be solved in polynomial time. Yet, with the help of provenance annotation, we are able to develop a query refinement method that works quite efficiently, as we demonstrate through extensive experiments.

PVLDB Reference Format:
Jinyang Li, Yuval Moskovitch, Julia Stoyanovich, and H. V. Jagadish. Query Refinement for Diversity Constraint Satisfaction. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS.

1 INTRODUCTION

With increasing awareness of the need to improve representation of historically underrepresented population groups, many companies, government agencies, educational institutions, professional societies, and other organizations have adopted diversity initiatives as part of their selection processes for recruitment, award nomination, etc. For instance, fellowship programs frequently require a diverse nominee pool, such that at least one nominee is a member of each demographic group that is underrepresented in a particular field [1, 2]. Relational databases are often used to select and prioritize candidates in such settings.

Traditional relational queries specify conditions as part of the query predicates, and output tuples that satisfy these predicates. If a query is used as part of some high-stakes selection process, then it would be natural to state diversity requirements as cardinality constraints over the presence of some demographic groups in the query result. For example, a company may want to invite at least three women and at least one individual of each race for job interviews. Unfortunately, such constraints are currently not supported by relational systems: We can specify criteria on how to select candidates for a job interview, but cannot impose constraints on demographic group membership of the selected candidates. In this paper, we propose a method for augmenting relational queries with group cardinality constraints to satisfy diversity requirements.

ID	Gender	Race	Major	GPA	Q1	Q2	Q3
1	F	White	ME	3.65			
2	F	White	CS	3.95	✓	✓	✓
3	F	Black	CS	3.40			
4	F	White	ME	3.60			
5	F	White	EE	3.85		✓	
6	F	Black	EE	3.90		✓	✓
7	F	Asian	EE	3.85		✓	
8	M	White	CS	3.65			
9	M	White	CS	3.90	✓	✓	✓
10	M	Black	CS	3.85	✓	✓	
11	M	White	CS	3.40			
12	M	White	EE	3.85		✓	
13	M	Asian	EE	3.95		✓	✓
14	M	Black	ME	3.60			

Table 1: `Applicants` table with primary key ID. Job applicants selected by queries `Q1`, `Q2`, and `Q3` are marked with ✓.

ID	Type	Hours
2	r.l.	80
2	t.c.	90
3	t.c.	90
6	g.a.	120
7	t.c.	40
7	g.a.	60
9	r.l.	60
13	g.a.	100
14	t.c.	100

Table 2: `Internships` table, with primary key (ID, Type).

An important starting point for our work is the substantial literature on imposing constraints on the size of the overall query result with the help of *query refinement*—introducing slight modifications to the query so that the output satisfies the specified cardinality bounds [3–11]. However, cardinality constraints over specific groups in the query result are not supported by these methods. We further motivate the need for such constraints and give an overview of our proposed approach next, using an example.

Example 1.1. Table 1 shows a dataset `D` of 14 job candidates applying to a tech company, with five attributes: gender, race, (college) major, and GPA. Of these, gender and race are the *sensitive attributes* that denote membership in demographic groups. The employer may wish to state cardinality constraints w.r.t. such attributes, to counteract the effects of historical discrimination (in the US, this is based on the doctrine of disparate impact, and corresponds to an affirmative action-style intervention). Major and GPA are the *qualification attributes*, and the employer may wish to use them in selection conditions.

The company would like to interview applicants who graduated from college with a technical major and a high GPA. To start, the data analyst uses the following query to select candidates:

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

```
Q1: SELECT *
    FROM Applicants AS a
      WHERE a.Major = 'CS' AND a.GPA >= 3.85
```

Query Q1 selects candidates #2, #9, and #10, as marked in the corresponding column in Table 1. Note that, although half of the applicants are female, only one of them is among the selected set. Further, although White, Black and Asian individuals are present among the applicants, there are no Asians among the selected set. To increase diversity, the company would like to interview at least two female applicants, and at least one applicant of each race. This requirement can be expressed as a set of cardinality constraints on groups in the query result. We call these *group cardinality constraints*. This can be accomplished by slightly adjusting the selection criteria, for example, by expanding the set of majors to also include Electrical Engineering (EE), thus *relaxing* the query:

```
Q2: SELECT *
    FROM Applicants AS a
      WHERE (a.Major = 'CS' or a.Major = 'EE')
      AND a.GPA >= 3.85
```

Query Q2 selects applicants #5, #6, #7, #12 and #13 to be interviewed in addition to applicants #2, #9, and #10, as marked in Table 1. The selected set includes four women and at least one individual of each race, and so the refined query satisfies the diversity requirement on both gender and race.

Next, the data analyst observes that a total of eight candidates will be interviewed based on Q2. This will require a substantial time commitment on the part of the company's human resources department, who then impose an additional requirement — to reduce the number of selected candidates to no more than five, thus introducing an upper-bound constraint on the size of the overall result set. To satisfy this additional constraint, the query can, once again, be refined, this time *restricting* the GPA range:

```
Q3: SELECT *
    FROM Applicants AS a
      WHERE (a.Major = 'CS' or a.Major = 'EE')
      AND a.GPA >= 3.90
```

The result of Q3 consists of 4 applicants, #2, #6, #9, and #13, with 2 of each gender and at least 1 of each race, satisfying all constraints.

Example 1.2. Next, suppose that the employer decides to add another requirement: selecting applicants with some internship experience. Table 2 contains internship information for some applicants, and includes applicant's ID, the type of employer they interned with, and for how many hours. To select applicants with at least 80 hours of experience, the data analyst executes the following query:

```
Q4: SELECT *
    FROM Applicants, Internships AS a, i
      WHERE a.ID = i.ID
      WHERE (a.Major = 'CS' or a.Major = 'EE')
      AND a.GPA >= 3.85 AND i.Hours >= 80
```

To have a team with diverse professional backgrounds, the company may wish to interview at least one applicant for each internship type: research lab (r.l.), tech company (t.c.), and government agency (g.a.). This can, once again, be expressed as a cardinality constraint, using attributes from Internships (Table 2). Thus, together with predicates and constraints from Example 1.1, we now have multiple selection predicates and cardinality constraints involving attributes from both tables. Then we can continue to refine the predicates in this query, as already discussed.

Outline of our approach. In this paper, we study the problem of finding minimal refinements of a query so that its result satisfies the given group cardinality constraints. Our goal is to modify selection predicates *just enough* to meet the constraints, while preserving the intention of the original query as much as possible.

To find all minimal refinements, there are two main challenges. First, to test whether a candidate query modification has a result that satisfies the constraints, we need to execute this query, which can be time-consuming, especially when the dataset is large or stored in a remote database. Second, there are many ways to change a predicate in a query, and there can be many predicates in the query, so the number of different possible changes is combinatorially large, making exhaustive analysis computationally expensive. To address the first challenge, based on Green *et al.* [12], our provenance model annotates tuples in the source data with the necessary information with respect to the predicate, and translates cardinality constraints to algebraic expressions [5] that can be used to test whether the query satisfies the constraints. To address the second challenge and get a minimal refinement such that changing any predicate closer to the original one would fail to meet the constraints, we create a data structure called PVL which contains all possible values of predicates in the data, and we develop an efficient searching algorithm with pruning techniques to search for all minimal refinements.

Diversity is a compelling need when distributing access to resources and opportunities in a society, as we saw in our running example. Furthermore, it is also desirable in many other settings. For example, we usually want search and recommendation systems to produce diverse results [13]. While we use a running example about fairness (in the sense of representation), we underscore that the techniques we propose in this paper can be applied in many other contexts.

Alternative approaches. One way to satisfy group cardinality constraints is to modify the result set directly in a post-processing step, adding or removing tuples: Adding candidate #7, who is both female and Asian, to the result of Q1 in Example 1.1 would meet the diversity requirements. However, this method may be illegal in some jurisdictions and application contexts (e.g., it would be illegal in the US in the context of employment, housing, and lending), because it uses demographic group membership explicitly as part of decision making, effectively subjecting applicants from different demographic groups to different processes. Even where legal, this method may have undesirable side effects, such as tarring all members of a group, including its best-qualified members, as "weaker," on account of there being a different standard applied to the group.

Unlike post-processing modification to the result set, modifying the query predicates is usually legal, because all individuals are evaluated using the same process. One famous recent case involved the University of Texas, which was not permitted to give African Americans explicit (post-processing) preference in admissions; however, it was permitted to use rank in school as the basis for admission, thereby admitting top students from poor-performing segregated schools in preference to second-tier white students from top schools who may have better test scores.

Another alternative approach, with which our work shares motivation, was proposed by Shetiya *et al.* [14]. Their goal is also to satisfy group cardinality constraints over the query result. However, their work differs from ours in four important ways. The first three are technical: First, they only handle constraints over a single binary sensitive attribute (e.g., male vs. female gender *or* majority vs. minority ethnicity), while we handle multiple sensitive attributes and do not limit them to be binary. Example 1.1 illustrates this, and includes two sensitive attributes: binary gender and (non-binary) race. Second, we support both query relaxation (i.e., generating more result tuples, as illustrated by query Q2) and query contraction (i.e., generating fewer result tuples, as illustrated by Q3), while Shetiya *et al.* only support query relaxation. Third, Shetiya *et al.* only focus on queries over a single relation, while we support SPJ queries with predicates and constraints of attributes over multiple tables joined together, as we showed in Example 1.2.

The fourth difference between our work and Shetiya *et al.* is conceptual. Their diversity objective is phrased in terms of minimizing the distance between the *result sets* produced by the original query and the rewritten query, while we+ aim to minimize the distance between the *queries themselves*. This difference may appear to be subtle, but it corresponds to intrinsically different objectives in the rewriting: preserving the salient properties of the selection process while potentially changing the result substantially vs. preserving the result as much as possible while potentially making substantial changes to the selection process.

Recall two refinements of query Q1 presented in Example 1.1: Q2 modifies one of the predicates of Q1, adding EE to the list of majors, while Q3 modifies both the predicate on major and the predicate on GPA. Shetiya *et al.* [14] use Jaccard similarity to compare query results, computing: $sim_{Jacc}(\texttt{Q1(D)}, \texttt{Q2(D)}) = \frac{3}{8} = 0.375$ and $sim_{Jacc}(\texttt{Q1(D)}, \texttt{Q3(D)}) = \frac{2}{5} = 0.4$. Thus, they would consider Q3 to have higher similarity to Q1 over D , while we consider Q2 to have higher similarity to Q1 based on the query predicates.

Contributions and roadmap. In Section 2, we formally define the Query Refinement Problem, the problem of finding minimal refinements of a query sequence given a dataset and a set of constraints on the query results. We theoretically analyze the complexity of the problem, proving its hardness. In Section 3, we describe our provenance model based on [12]. Then, in Section 4, we present the Possible Value Lists (PVL), a data structure for representing the possible refinements based on provenance expressions, with the goal of improving the efficiency of search for minimal refinements. Our algorithm for generating minimal refinements utilizing the PVL is presented in Section 5, where we also propose optimization to the algorithm that can be applied in special cases. In Section 6, we experimentally evaluate our solution with multiple datasets, queries, and constraints. We give an overview of related work in Section 7 and conclude in Section 8.

2 PROBLEM DEFINITION

In this section, we first present the notion of query refinement. We then introduce cardinality constraints and formally define the Query Refinement Problem. Finally, we prove that the problem as defined cannot be solved in polynomial time, in the worst case. Figure 1 summarizes the notations used in this paper.

Notation	Description
D	dataset
Q	query or refinement query
$Q(D)$	result of executing Q over D
Q_n, Q_c	numerical predicates, categorical predicates of Q
$p.A, p.C, p.op$	attribute, constant, and operator of predicate p
$\mathcal{G}$	conjunction of conditions
$Q(D)_{\mathcal{G}}$	size of the group of tuples in $Q(D)$ satisfying $\mathcal{G}$
Cr	cardinality constraints
$\mathcal{V}_{Q(D)}$	set of variables in provenance expressions
$prov(t)$	annotation of tuple $t \in D$
$\mathcal{I}^{Cr}_{Q(D)}$	provenance expressions for constrains Cr over $Q(D)$
$val_{Q'}(A_v)$	valuation of $A_v \in \mathcal{V}_{Q(D)}$ with refinement Q'
$Tval_{Q'}(\mathcal{I}^{Cr}_{Q(D)})$	truth value of expression set $\mathcal{I}^{Cr}_{Q(D)}$
$L_{D,Q}$	PVL given D, Q
$Q.R$	list of indices representing Q in PVL $L_{D,Q}$
$Q\|_P$	partial query of Q containing predicates in set P

Figure 1: Notations Used.

2.1 Query Refinement

We consider in this paper the class of queries considered in [3], conjunctive Select-Project-Join (SPJ) queries with selection predicates. The selection predicates can be defined over *numerical* or *categorical* attributes. For numeric domains, selection predicates include range ($<, \leq, \geq, >$) and equality ($=$) predicates. Categorical domains permit only equality predicates. Namely, categorical predicates are of the form `attribute = constant_1 OR...OR attribute = constant_n`. For ease of presentation, in the rest of the paper, we assume numerical predicates are of the form `attribute <op> constant`[1] where `<op>` can be one of $\{<, \leq, \geq, >\}$. Equality predicates `attribute = constant` are translated into two predicates `attribute >= constant` and `attribute <= constant`.

For a query Q, we use Q_n and Q_c to denote the set of numerical and categorical selection predicates, respectively. Furthermore, we use $p.A$, $p.op$ and $p.C$ to denote the predicate's parts. Namely, for numerical predicates, p : `attribute >= constant`, $p.A$ denotes the `attribute`, $p.op$ denotes the operator `>=`, and $p.C$ to denotes the `constant`. Similarly, for categorical predicates `attribute = constant_1 OR...OR attribute = constant_n` we use $p.A$ to denote the attribute in the predicate, and $p.C$ and the list of constants in the p, i.e., the set {`constant_1,...,constant_n`}.

We use the notion of query refinement defined in [3] to formally state our problem. For a numerical predicate, refinements are changes to the value of the constant; while for categorical predicates, a refinement is done by adding and/or removing predicates from the original constant list.

Definition 2.1 (Predicate refinement (adopted from [3])). Let Q be an SPJ query, a refinement of a selection predicate $p \in Q$ is a selection predicate p' such that $p'.A = p_n.A$, and $p'_n.C \neq p_n.C$.

Example 2.2. Consider the predicates in query Q1 given in Example 1.1. For numeric predicate `a.GPA >= 3.85`, a refinement can

[1]Extending our solution to support other forms of numerical predicates, such as `attribute1 <op> constant·attribute2` is straightforward, and we do not discuss them in this paper.

be `a.GPA >= 3.90`. For categorical predicate `a.Major = 'CS'`, a refinement can be `a.Major = 'CS' or a.Major = 'EE'`.

The categorical refinement of [3] considers a hierarchy that also includes roll-up and drill-down. We can convert any hierarchical categorical refinement to Definition 2.1. For instance, rolling up `Country = 'US' and State = 'CA'` to get `Country = 'US'` is semantically equivalent to adding all the values in the domain of `State` to its conjunction. In the rest of the paper, for simplicity, we assume no hierarchy over categorical attributes.

Definition 2.3 (Query refinement). A query Q' is a refinement of query Q if Q' is obtained from Q by refining some predicates of Q.

Example 2.4. Query `Q3` is a refinement of `Q1` from Example 1.1 with respect to attributes Major and GPA.

2.2 Group Cardinality Constraints

Let D be a dataset, Q a query, and $Q(D)$ the result of executing the query over D. We define groups in $Q(D)$ using a conjunction of conditions $\mathcal{G} = \wedge_i(A_i = v_i)$ where A_i are distinct data attributes in $Q(D)$. For instance, in our running example, one of the conditions $\mathcal{G}$ is $\{Gender = F\}$. We use $Q(D)_{\mathcal{G}}$ to denote the group of tuples in $Q(D)$ that satisfy the condition $\mathcal{G}$. We can then define cardinality constraints over data groups as follows.

Definition 2.5 (Cardinality Constraints). Let D be a dataset, Q a query, and $\mathcal{G}$ a group definition condition set. A set of cardinality constraints Cr over $Q(D)_{\mathcal{G}}$ is a conjunction of expressions of the form $|Q(D)_{\mathcal{G}}|\ op\ x$, where $op \in \{\leq, <, =, >, \geq\}$ and x is a constant, or a function of the size of some other data group defined using $\mathcal{G}'$.We say that $Q(D)$ satisfies Cr if all the expressions hold. Namely, multiple cardinality constraints should be satisfied conjunctively.

Example 2.6. Continuing with Example 1.1, the cardinality constraint over the number of females can be formally expressed as $Q(D)_{Gender=F} \geq 2$. The other constraints include $Q(D)_{Race=White} \geq 1$, $Q(D)_{Race=Asian} \geq 1$, $Q(D)_{Race=Black} \geq 1$, $Q(D)_{\emptyset} \leq 5$.

Given a dataset D, a query Q, and a set of cardinality constraints Cr such that $Q(D)$ does not satisfy Cr, there can be multiple ways to refine Q so as to satisfy the constraints, as we demonstrate next.

Example 2.7. In Example 1.1, the results of the query `Q2` or `Q3`, which are both refinements of `Q1`, satisfy cardinality constraint $Q(D)_{Gender=F} \geq 2$ (at least 2 females). Another plausible refinement of `Q1` that qualifies at least two females is to adjust the GPA predicate to be `a.GPA >= 3.40`.

The refinements depicted in the above example suggest the company may have various ways to achieve its diversity goal. Each applies different modifications to the query. Intuitively, minimal modifications to the original query are preferred, e.g., a query that refines the predicate GPA to be `a.GPA >= 3.6` is preferred over one that refines the predicate GPA to be `a.GPA >= 3.55`, however, refinements that modify different attributes may be incomparable when no additional preference information is provided by the end-user (as long as they all satisfy the constraints). To capture this idea, we define the set of minimal refinements. We first define the dominance between predicate refinements as follows.

Definition 2.8 (Dominance in Refinement). Let Q be a query with a predicate p and let p', p'' be two refinements of p. For numerical predicate $p \in Q_n$, we say p' dominates p'', if $|p.C - p'.C| < |p.C - p''.C|$. For categorical predicate $p \in Q_c$, we say p' dominates p'', if $p.C \setminus p'.C \subsetneq p.C \setminus p''.C$ and $p'.C \setminus p.C \subsetneq p''.C \setminus p.C$. Let Q' and Q'' be two refinements of Q. We say that Q' dominates Q'' (with respect to Q) if $\forall p' \in Q'$, for $p'' \in Q''$ such that $p'.A = p''.A$, we have $p'' = p'$ or p' dominates p''.

Example 2.9. In Example 1.1, for GPA predicate `a.GPA = 3.85` in `Q1`, refinement `a.GPA = 3.90` dominates `a.GPA = 3.60`. For Major predicate `a.Major = 'CS'`, refinement `a.Major = 'EE'` dominates `a.Major = 'EE' or a.Major = 'ME'`).

Definition 2.10 (Minimal Refinement). Given a dataset D, a query Q, and a (set of) cardinality constraint(s) over data group(s) Cr, we say that Q' is a minimal refinement of Q with respect to Cr if (i) Q' is a refinement of Q, (ii) $Q'(D)$ satisfies Cr and (iii) $\nexists Q''$ such that Q'' satisfies conditions (i) and (ii), and Q'' dominates Q'.

Our goal is to find all minimal refinements of a query with respect to a set of cardinality constraints as we next define.

Problem 2.11 (Query Refinement Problem). *Given a dataset D, a SPJ query Q, and a set of cardinality constraints Cr, find all minimal refinements of Q with respect to Cr.*

Note that there may be a number of minimal refinements. The goal is to report all of them. It is also possible that no refinement satisfies the constraints. In this case, the result is an empty set of refinements. A naive solution to the Query Refinement Problem would be to traverse the set of all possible refinements. For each refinement, check whether it satisfies the constraints and if so, add it to a result set R. Finally, remove from R queries that are dominated by others in R. The time complexity of this naive approach is exponential, and as we show next, there are no polynomial time algorithm to solve the Query Refinement Problem.

2.3 Hardness Analysis

Theorem 2.12. *Given a dataset, an SPJ query, and a set of cardinality constraints, no polynomial time algorithm in the number of selection predicates can guarantee the enumeration of the set of all minimal refinements.*

	A_1	A_2	A_3	…	A_N	Gender
t_1	0	1	1	…	1	F
t_2	1	0	1	…	1	F
t_3	1	1	0	…	1	F
⋮	⋮	⋮	⋮	⋮	⋮	F
t_N	1	1	1	…	0	F
t_{N+1}	0	0	0	0	0	F
⋮	⋮	⋮	⋮	⋮	⋮	F
t_n	0	0	0	0	0	F

Table 3: Dataset for proof

Proof. We prove the theorem by constructing an example with an exponential number of minimal refinements. Consider a dataset with n tuples and query Q with N predicates over attributes $A_1 \cdots A_N$. Assume all N predicates are numeric predicates where the value of attribute $A_i (1 \leqslant I \leqslant N)$ can be either 0 or 1.

Without loss of generality, we assume $n > N$ and $N\%2 = 0$. Figure 3 shows the dataset, where the values of an attribute A_i for a tuple t is represented as either 1 or 0, meaning the value of $t.A_i$. For t_i where $1 \le i \le N$, $t_i.A_i = 0$, and $t_i.A_j = 1, j \ne i$. That is, for $t_1 - t_N$, only values on the diagonal are 0. For t_i where $i \ge N+1$, $t_i.A_j = 0, 1 \le j \le N$.

As in Figure 3, all n tuples have attribute gender as female, and we assume the cardinality constraint is to have at least $N/2$ females. With the original query Q, no female is selected, so we need to refine Q so that at least $N/2$ tuples satisfy the query. Since all values are binary, a refinement is to refine some predicates from $p.c = 1$ to $p.c = 0$. Any refinement that refines $N/2$ of the predicates to be $p.c = 0$ is a minimal refinement, for the following two reasons. First, such a refinement R makes $N/2$ of the tuples among $t_1 - t_N$ selected by the query, thus satisfying the cardinality constraint. Second, for any refinement R' dominated by R, R' has at most $N/2-1$ predicates refined with $p.c = 0$, making at most $N/2-1$ tuples among $t_1 - t_N$ selected by the query, and zero tuples among $t_{n+1} - t_n$ selected, thus not satisfying the cardinality constraint.

Therefore, any refinement refining half of the predicates is a minimal refinement, so the number of minimal refinements is $\binom{N}{N/2} = O(\sqrt{2}^N)$. □

In what follows, we will present a solution that largely increase the efficiency by using provenance model to avoid redundant database access, and a searching algorithm with pruning methods.

3 PROVENANCE MODEL

Given a query, the number of possible refinement queries can be extremely large. Furthermore, determining whether the output of a refinement query satisfies the cardinality constraints requires the evaluation of the query over the data. This process can be expensive, particularly for large or remote databases. However, this costly query evaluation process can be eliminated using data annotations developed in the context of provenance theory. In fact, we can generate provenance annotations within a linear pass over the dataset, and then propagate them throughout the query evaluation to generate provenance expression as we next explain.

Our provenance model is inspired by the idea of hypothetical reasoning using provenance presented in [5]. Intuitively, we generate provenance expressions, using the data, the given SPJ query, and the cardinality constraints. We leverage the idea of conditional tables (c-tables) [15], where tuples are associated with conditions. To capture the possible refinements, we annotate tuples in the data with the query selection conditions.

3.1 Provenance expressions

We first define a set of variables used in the provenance expressions. This set is determined by the given query Q and the data D.

Definition 3.1. Given a query Q over the data D, the set of variables $\mathcal{V}_Q$ is defined as

$$\mathcal{V}_{Q(D)} = \{A_{[t.A]} \mid \exists p \in Q_n \cup Q_c, p.A = A, t \in D\}$$

Where $[t.A]$ denotes the value of the attribute A in t.

Example 3.2. In Example 1.2, we have three attributes from two tables being joined in the query: Major, GPA and internship hours, with three, six, and six values, respectively. We use M, G and H as short for Major, GPA and internship hours, respectively.

$$\mathcal{V}_{Q(D)} = \{M_{CS}, M_{EE}, M_{ME}, G_{3.95}, G_{3.90}, G_{3.85}, G_{3.65}, G_{3.60}, G_{3.40}, H_{40}, H_{60}, H_{80}, H_{90}, H_{100}, H_{120}\}$$

We use the set of variables $\mathcal{V}_{Q(D)}$ to generate provenance annotations. In what follows, we use $\widetilde{Q}$ to denote the query obtained from Q when omitting the selection predicates. We generate provenance annotations for each tuple $t \in \widetilde{Q}(D)$. Intuitively, the annotated output $\widetilde{Q}(D)$ includes all the tuples after executing only join operations, and it represents the output of all possible refinements of Q, where the provenance annotation of each tuple can be used to determine whether it satisfies any given refinement.

Definition 3.3. Let Q be an SPJ query over D, the annotation of each tuple $t \in \widetilde{Q}(D)$ is $prov(t) = \prod_{i=1}^{i=k} A_{i[t.A_i]}$ where each A_i is an attribute that is involved in Q (i.e., $\exists p \in Q_n \cup Q_c, p.A = A_i$), and k is the number of such attributes.

Example 3.4. Continue with our running example.The annotations of applicants #3 and #6 are as follows.

$$prov(t_3) = M_{CS} \cdot G_{3.40} \cdot H_{90},\ prov(t_6) = M_{EE} \cdot G_{3.90} \cdot H_{120}$$

Finally, we define expressions that express the cardinality constraints using the resulting provenance annotations.

Definition 3.5 (Provenance expression of a constraint). Given a dataset D and a query Q, and a cardinality constraint $Q(D)_{\mathcal{G}}\ op\ x$, where $op \in \{>, \ge, <, \le\}$, we define the provenance expression of the constraint as $\left(\sum_{t \in Q(D)_{\mathcal{G}}} prov(t)\right)\ op\ x$. For a given set of cardinality constrains Cr over $Q(D)$ we use $\mathcal{I}^{Cr}_{Q(D)}$ to denote the corresponding set of provenance expressions.

Example 3.6. Consider again Example 1.1 and 1.2 with the data in Table 1 and Table 2. We first join Table 1 and Table 2 by attribute ID, so the "intermediate" table only contains applicants #2, #3, #6, #7, #9, #10, #13, #14, with their ID, gender, race, major, GPA, internship hours and former employer. The set of provenance expressions $\mathcal{I}^{Cr}_{Q(D)}$ consists of 8 provenance expressions corresponding to the set of constraints. For example, the following expressions correspond to the constraint that at least two females and at least one white are selected.

$$M_{CS} \cdot G_{3.95} \cdot H_{170} + M_{CS} \cdot G_{3.40} \cdot H_{90} + M_{EE} \cdot G_{3.90} \cdot H_{120} + M_{EE} \cdot G_{3.85} \cdot H_{100} \ge 2 \quad (1)$$

$$M_{CS} \cdot G_{3.95} \cdot H_{170} + M_{CS} \cdot G_{3.90} \cdot H_{60} \ge 1 \quad (2)$$

3.2 Refinements through valuation

For a given query Q over the data D, each possible refinement query Q' corresponds to an assignment to the variables set $\mathcal{V}_{Q(D)}$ in the corresponding provenance expression. The latter is called a *valuation* in the provenance literature, and we formally define the valuation of our variables as follows.

Definition 3.7 (Valuation). Let $A_v \in \mathcal{V}_{Q(D)}$ and Q' be a refinement query of Q.

$$val_{Q'}(A_v) = \begin{cases} 1 & \text{if } p \in Q'_n \text{ s.t. } p.A = A, v\ op\ p.c \\ 1 & \text{if } p \in Q'_c \text{ s.t. } p.A = A, v \in p.C \\ 0 & \text{otherwise} \end{cases}$$

By assigning the value $val_{Q'}(A_{iv})$ to each variable in the provenance expression, the above value assignment can then be used for the valuation of the provenance expressions associated with each tuple in the data and in turn, the truth values of the provenance expressions. Given a set of provenance expressions $\mathcal{I}^{Cr}_{Q(D)}$ and the value assignment $val_{Q'}$, we use $Tval_{Q'}(I)$ to denote the truth value of the expression $I \in \mathcal{I}^{Cr}_{Q(D)}$, obtained by applying $val_{Q'}(A_{iv})$ on each variable A_{iv} in I. Overloading notation we use $Tval_{Q'}(\mathcal{I}^{Cr}_{Q(D)})$ to denote the truth value of the expression set $\mathcal{I}^{Cr}_{Q(D)}$, namely $Tval_{Q'}(\mathcal{I}^{Cr}_{Q(D)}) = \wedge_{I \in \mathcal{I}^{Cr}_{Q(D)}} Tval_{Q'}(I)$

Example 3.8. Consider the provenance expression Equation 1 (denoted as I_1) and Equation 2 (denoted as I_2) from Example 3.6. With query Q4, only variables $G_{3.40}$ and H_{60} are 0, and others are 1. Thus the truth values of I_1 and I_2 are both true.

Note that the valuation function in Definition 3.7 resembles the use of the logical expression: intuitively, a valuation of 1 (true) or 0 (false) indicates whether a single tuple beongs to the (refined) query output. These values are then aggregated to capture the cardinality constraints, which cannot be expressed using logical expression.

Proposition 3.9. *Let D be a dataset and Q a query. Q satisfies a set of cardinality constraints Cr if and only if $Tval_Q(\mathcal{I}^{Cr}_{Q(D)}) = True$.*

Namely, given the provenance expressions of the cardinality constraints, we can efficiently examine the effect of refinements on constraint satisfaction without the need to access the data and execute the potential refinements. Furthermore, the provenance expressions may guide the refinement search as we next explain.

4 POSSIBLE VALUE LISTS (PVL)

With a provenance model available for testing potential refinements without accessing the data itself and repetitively evaluating refinement queries, the next question is how to find all the minimal refinements efficiently. A straightforward method is to traverse all possible refinements one by one, but this can be computationally prohibitive given the combinatorial number of refinements. Our solution uses a set of lists depicting the possible refinements based on the variables in the provenance expression, called the Possible Value Lists (PVL), as we next describe.

The PVL is used to enumerate the possible refinement queries. Each such query can be defined by the modifications, to the original query, in the predicates' values. Recall that a query can be refined by disjunctively adding and/or removing values from the original categorical predicate lists in the query, or by varying the constants in the conditions over numerical attributes. Moreover, for numerical predicates, minimal refinements can only consist of numerical values from the tuples in the data that are involved in cardinality constraints. E.g., if we only consider satisfying the constraint over the number of females in our running example, a refinement query that refines the predicate GPA to be `a.GPA >= 3.45` can not be minimal because it results in the same output as refining the predicate GPA to be `a.GPA >= 3.60`, since no female applicant has a GPA such that $3.45 \leq GPA < 3.60$, but the latter dominates the former.

Given a dataset D and a query Q and cardinality constraints Cr, a PVL denoted as $L_{D,Q,Cr}$, depicts the possible modifications

	$l_{M_{EE}}$	$l_{M_{ME}}$	$l_{M_{CS}}$	l_G
1	0	0	1	3.85
2	1	1	0	3.90
3				3.95
4				4.00
5				3.65
6				3.60
7				3.40

(a) The PVL $L_{D,Q,Cr}$ of the running example (Example 1.1)

	$l_{M_{ME}}$	$l_{M_{CS}}$	l_G
1	0	1	3.85
2	1	0	3.90
3			3.65
4			3.60
5			3.40

(b) The PVL of Example 5.3

	$l_{M_{EE}}$	$l_{M_{ME}}$	l_G
1	0	0	3.85
2	1	1	3.65
3			3.60
4			3.40

(c) The PVL of Example 5.6

Figure 2: Possible value lists (PVL).

using lists of possible values. For each numerical predicate $p_n \in Q_n$, $L_{D,Q,Cr}$ consists of a list containing $p_n.C$ and all values of $p_n.A$ appearing in tuples of D that are involved in data groups specified by Cr. We denote this list as $l_{p_n.A}$. For each categorical predicate $p_c \in Q_c$, $L_{D,Q,Cr}$ consists of a set of lists, one for each possible value v in the domain of $p_c.A$, denoted as $l_{p_c.A_v}$. Each of these lists contains two values, 1 and 0, which represents the existence (1) and absence (0) of v in $p_n.C$, respectively. Intuitively, this set of lists corresponding to a categorical predicate p_c can be used to describe the set of all possible values' subgroups of attribute $p_c.A$. The values in the lists are sorted based on their distance from the values in the original query Q. For $p_n \in Q_n$, values in list $l_{p_n.A}$ are sorted by the distance from $p_n.C$, such that values closer to C_i are placed higher. For $p_c \in Q_c$, similarly, $l_{p_c.A_v} = [1, 0]$ if $v \in p_c.C$, and $l_{p_c.A_v} = [0, 1]$, otherwise. In other words, 1 (existence) is placed higher than 0 (absence) if v is already part of the original predicate p_c; and vice versa.

Example 4.1. Figure 2a depicts the PVL $L_{D,Q1,Cr}$ of Example 1.1 with dataset D, query Q1, and Cr over the number of females, people of each race, and the overall size of the result. The Major predicate has CS but not EE or ME, so 1 is placed above 0 in the list $l_{M_{CS}}$, and 0 is placed above 1 in the lists $l_{M_{EE}}, l_{M_{ME}}$. Note that values 4.00, 3.95, and 3.90 in l_G correspond to values 3.95, 3.90 and 3.85 in D, respectively. The meaning of colors highlighting some values will be explained later.

Note that depending on $p_n.op$ (operators in predicates) and values in D, the values in list $l_{p_n.A}$ may be $v \pm precision(p_n.A)$, where v is a value of numerical attribute $p_n.A$ in D and $precision(p_n.A)$ is its precision, e.g., in our running example $precision(GPA) = 0.05$. Specifically, when $p_n.op$ is $\geq$ ($<$), for values $v \geq p_n.C$, $l_{p_n.A}$ should contain $v + precision(p_n.A)$ instead of v; when $p_n.op$ is $\leq$ ($>$), for values $v \leq p_n.C$, $l_{p_n.A}$ should contain $v - precision(p_n.A)$ instead of v. In the running example, to contract GPA predicate (`a.GPA >= 3.85`) so that it disqualifies a 3.85 (3.90, 3.95) GPA value, it needs to be refined to `a.GPA >= 3.90 (3.95, 4.00)`.

Intuitively, each value in a list of PVL $L_{D,Q,Cr}$ demonstrates a possible refinement of a specific predicate. A combination of one value from each list corresponds to a refinement of the query Q. In this manner, PVL encompasses all minimal refinements.The relationship between PVL and refinements of query Q is summarized in the following proposition.

Proposition 4.2. *Given a query Q over the dataset D and cardinality constraint Cr, let $\mathbb{Q}$ be the set of all possible refinement queries Q that can be generated using values from D.*

(1) Every minimal refinement query Q must be in $\mathbb{Q}$.
(2) $\forall Q' \in \mathbb{Q}$, Q' can be represented using a set of indexes $Q'.R$ in the PVL $L_{D,Q,Cr}$ such that $Q'.R[l]$ is the index in the list $l \in L_{D,Q,Cr}$. $\forall p_n \in Q'_n, \exists l_{p_n.A} \in L_{D,Q,Cr}, p_n.C = l_{p_n.A}[Q'.R[l_{p_n.A}]]$. $\forall p_c \in Q'_c, \exists l_{p_c.A_v} \in L_{D,Q,Cr}$ such that $l_{p_c.A_v}$ consists of 0 and 1. If $v \in p_c.C$ then $Q'.R[l_{p_c.A_v}]$ is the index of 1, and 0 otherwise.
(3) Every possible index set $Q'.R$ corresponds to a query $Q' \in \mathbb{Q}$.

Example 4.3. Revisiting Example 1.1 with PVL in Figure 2a, we can represent refinement query Q2, which adds EE as a recognized major, as $Q2.R = [2, 1, 1, 2]$. This corresponds to values $l_{M_{EE}}[2] = 1, l_{M_{ME}}[1] = 0, l_{M_{CS}}[1] = 1, l_G[1] = 3.85$ The refinement query Q3 from Example 1.1 can be represented as $Q3.R = [2, 1, 1, 2]$, as highlighted in blue.

Our algorithm for generating minimal refinements utilizes the PVL to traverse the possible refinements. As stated in Proposition 4.2, all minimal refinements can be represented from the PVL. This representation holds the following properties that serve as the foundation for the correctness of our algorithm.

Proposition 4.4. *Let D be a dataset, Q be a query, and Cr be constraints, and $L_{D,Q,Cr}$ be the corresponding PVL which consists of n_l lists. Additionally, let $Q_1, Q_2 \in \mathbb{Q}$ be two different refinement queries.*

(1) Q_1 dominates $Q_2 \iff \forall j, 1 \le j \le n_l, Q_1.R[j] \le Q_2.R[j]$.
(2) If Q_1 and Q_2 are minimal refinements then $\exists j, 1 \le j \le n_l$, such that $Q_1.R[j] \le Q_2.R[j]$ and $\exists j', 1 \le j' \le n_l$, such that $Q_1.R[j'] > Q_2.R[j']$

In the next section, we will explain how to search for minimal refinements with PVL.

5 GENERATING MINIMAL REFINEMENTS

We now introduce the algorithm for searching minimal query refinements using the PVL. We will first describe the algorithm, followed by an explanation of optimizations that can be applied in specific scenarios.

5.1 The algorithm

The algorithm uses the notion of partial queries we next define.

Definition 5.1 (Partial queries). Given a query Q with the set of predicates $Q_n \cup Q_c$ and a subset $P \subseteq Q_n \cup Q_c$, the query $Q|_P$ is similar to Q but consists only of the predicates in the set P. We say that $Q|_P$ is a *partial query* of Q.

Example 5.2. Consider the original query Q1 in Example 1.1 with predicates `a.Major = 'CS' AND a.GPA >= 3.85`. For $P = \{$`a.GPA >= 3.85`$\}$, $Q1|_P$ is a partial query of Q1.

The high-level idea of the algorithm is to generate refinement queries that satisfy the constraints by completing the predicate set of partial queries to obtain refinements of the given query Q. This process is done for different partial queries using a recursive function. We next provide details on the different parts of the algorithm.

Algorithm 1: Search for all minimal refinements

input : PVL L, query Q, and cardinality constraints $\mathcal{I}$.
output: All minimal relaxations

Function `Search`$(L, Res, Q|_P, \mathcal{I})$:
1 $Q_b \leftarrow$ `findRefinement` $(L, \mathcal{I}, Q|_P)$
2 **if** $Q_b.R \neq \emptyset$ **then**
3 `update` (Res, Q_b)
4 **if** *there are more than one list in L* **then**
5 **foreach** $l \in L$ **do**
6 $k \leftarrow Q_b.R[l]$
7 **for** *i from $k-1$ to 1* **do**
8 $Q'|_{P'} \leftarrow$`addPredicate` $(Q|_P, l[i])$
9 $\mathcal{I}_r \leftarrow$ inequalities of $\mathcal{I}$ requiring relaxations
10 **if** $Tval_{Q'|_{P'}}(\mathcal{I}_r) = True$ *and there are other lists l' s.t. $Q_b.R[l]$ is not the last index in l'* **then**
11 $L' \leftarrow$ remove l from L
12 $Res \leftarrow$ `Search`$(L', Res, Q'|_{P'}, \mathcal{I})$
13 remove values above index $Q_b.R[l]$ in list l
14 **return** *Res*

Finding a minimal refinement (from a partial query). Given a partial query $Q'|_P$ (where Q' is a refinement of Q), the algorithm generates minimal refinement queries by completing the predicates of P to obtain a refinement of the input query Q, which is procedure `findRefinement` (line 1). This is done by traversing the PVL in a top-down manner, attempting all possible refinements until a suitable refinement is found. This approach starts with the values closest to the original predicate, ensuring that the first refinement satisfying the cardinality constraints must be a minimal refinement (since no refinements located above it satisfies them).

Recursive search. The core of the algorithm is a recursive search function depicted in Algorithm 1. Given a partial query $Q|_P$ with a fixed predicate set P (as part of a minimal refinement), a temporary result set *Res*, a PVL L, and a set of provenance inequalities, `Search` function first uses the `findRefinement` procedure to identify a minimal refinement Q_b (line 1). If no such refinement exists, the current result set *Res* is returned; otherwise, *Res* is updated with Q_b (line 3), and the algorithm generates new refinements by adding a predicate with a value from a list $l \in L$ to P, removing l from L, and recursively searching for new refinements (lines 4–13). Specifically, for the list $l \in L$, the algorithm considers every possible predicate that can be generated using values located above $Q_b.R[l]$ in l, effectively adding a new predicate to P and removing one list from L in each iteration. This process ends when there are no refinements found in the PVL or when the PVL has only one list.

Reducing the search space. When we traverse the possible refinements, the search space can be reduced using the notion of *partial dissatisfaction* as we next explain. For a given set of provenance inequalities $\mathcal{I}$, we use $\mathcal{I}_r \subseteq \mathcal{I}$ to denote the subset of inequalities that are of the form $Q(D)_{\mathcal{G}}\ op\ x$, where $op \in \{>, \ge\}$ and x is a constant (we call such constraints *relaxation constraints*). For any given (refinement) query Q', if there exists a predicate set $P \subset Q'_n \cup Q'_c$ such that $Tval_{Q'|_P}(\mathcal{I}_r) = False$ then $Tval_{Q'}(\mathcal{I}^{Cr}_{Q(D)}) = False$. Namely, if a partial query of Q' does not satisfy some relaxation constraints,

then Q' does not satisfy the whole constraint set. For example, if a partial query $Q|_P$ disqualifies 6 out of 7 females, then any query Q would not satisfy the constraint that at least 2 females are selected.

Putting it all together. Given a query Q over D and a set of provenance inequalities $\mathcal{I}^{Cr}_{Q(D)}$ generated form the cardinality constraints Cr, we find the set of all minimal refinements by calling the `Search` function with the initial parameters $L_{D,Q,Cr}$, an empty result set *Res*, a partial query $Q|_P$ with $P = \emptyset$, and $\mathcal{I}^{Cr}_{Q(D)}$. `Search` calls `findRefinement` as explained above and use it to generate new refinements. The search is optimized by avoiding refinements that can not satisfy the constraints based on partial dissatisfaction (lines 9 – 10). Finally, to avoid repetitive checking, at the end of the recursive call with values in list l, we remove all these values, i.e., remove values above index $Q_b.R[l]$ in this list (line 13). This is because refinements with these values are already checked.

Example 5.3. Consider Example 1.1 with data in Figure 1, original query `Q1` and constraints. Provenance expressions of the constraints are generated using the provenance model, and PVL is built as shown in Figure 2a. To reduce the search space, 3.95 and 4.00 in l_G (highlighted in grey) are removed since partial queries with `a.GPA >= 3.95` (or `a.GPA >= 4.00`) do not satisfy the requirement of selecting at least two females. We search in PVL for base minimal refinement by a top-down traversal, and obtain Q_b s.t. $Q_b.R = [2, 1, 1, 2]$, as highlighted in blue, refining Major predicates to `a.Major = 'EE' or a.Major = 'CS'` and GPA predicate to `a.GPA >= 3.90`. Based on Proposition 4.4, any other minimal refinement must have some values above these blue values in PVL, so we check the value in $l_{M_{EE}}[1]$ and $l_G[1]$. For the value in $l_{M_{EE}}[1]$, we add EE to the Major predicate, get `a.Major = 'CS'`, generate a new PVL by removing list $l_{M_{EE}}$ from PVL, and recursively search in the same way, but no base minimal refinements are found in Figure 2b.. Similarly, we fix GPA predicate with value $l_G[1] = 3.85$, generate a new PVL by removing l_G from PVL in Figure 2a, and recursively search in the same way, but do not find any more minimal refinements. The final result contains only one minimal refinement.

We prove that our algorithm can find all minimal refinements without reporting any non-minimal refinements, as in the following theorem.

Theorem 5.4. *Given a dataset D, a SPJ query Q, and a set of provenance inequalities $\mathcal{I}^{Cr}_{Q(D)}$, our algorithm can find all minimal refinements, and all of those reported are minimal refinements.*

Proof. The second half of the theorem is trivial because we always check a refinement against the current result set before adding it, anything reported by the algorithm must be a minimal refinement. We prove the first half of the theorem by induction. Based on Algorithm 1, the lists in PVL are predicates that remain to be refined, and Q'_P is a partial query with predicates P that are already fixed. We aim to prove that each iteration of Search(L, Res, Q'_P) can find all minimal refinements.

Base case: When there is only one list in PVL, there is only one predicate p remaining to refine with at most one minimal refinement. If there is a refinement of p with which the updated query $Q'|_P$ have $Tval_{Q'|_P}(\mathcal{I}^{Cr}_{Q(D)}) = True$, it can be found out by the search in Algorithm 1, line 1.

Induction step: Assume that with an x-list PVL, Algorithm 1 can find all minimal refinements. We need to prove it still can with an $(x+1)$-list PVL. Suppose there are y minimal refinements with an $(x+1)$-list PVL, where $y \geq 1$. One of these y minimal refinements will be found as our base minimal refinement Q_b by traversal. For each of the other $(y-1)$ minimal refinements Q, compared with Q_b, Q is refined less w.r.t some lists, and more w.r.t. some other lists, based on Proposition 4.4. So there is at least one list l in PVL such that $Q.R[l] < Q_b.R[l]$. Let l be the left-most list satisfying this condition. If l is a numerical list corresponding to predicate p, according to Algorithm 1, we will get updated partial query $Q'|_{P'}$ by adding predicate p' s.t. $p'.A = p.A, p'.op = p.op, p'.c = l[Q.R[l]]$. If l is a categorical list corresponding to A_v of predicate p, similarly, we update partial query $Q|_P$ by adding or not adding v to $p.C$ depending on whether the value to fix is 1 or 0. And then we do iteration Search($L'_{D,Q,Cr}, Res, Q'|_{P'}$), where $L'_{D,Q,Cr}$ is the PVL without list l. Based on our induction hypothesis, Search($L'_{D,Q,Cr}, Res, Q'|_{P'}$) can find all minimal refinements since $L'_{D,Q,Cr}$ has x lists. Thus, Q can be found by Search($L'_{D,Q,Cr}, Res, Q'|_{P'}$). Removing values above the base minimal refinement Q_b (line 13 in Algorithm 1) does not affect generating the query Q, because they are removed after being checked already. □

Complexity analysis. Generating the provenance inequalities is polynomial in the data size, as it requires a single linear pass over the dataset to annotate the data in addition to the query evaluation. For the searching algorithm, the worst case is to traverse all possible refinements, so the time complexity is the number of all possible refinements is the product of the length of all lists in PVL is

$$O\left(\left(\prod_{i=1}^{N_n}(|Dom(A_i)|+1)\right)\cdot 2^{\sum_{j=1}^{N_c}|Dom(A_j)|}\right)$$

where N_n, N_c denote the number of numeric predicates and categorical predicates, respectively; A_i, A_j denote numeric attributes and categorical attributes, respectively, and $Dom(A)$ denotes the domain of attribute A. In the worst case, $|Dom(A_i)| = |D| = n$, thus we have $O\left(\left(n^{N_n}\right)\cdot 2^{\sum_{j=1}^{N_c}|Dom(A_j)|}\right)$

While the time complexity is exponential to the number of numerical predicates and the size of domains of categorical predicates in the worst case, we show in our experiments that in practice, the running time of our algorithm is typically much better, due to our search space pruning.

5.2 Optimizations

We next describe some optimizations that can be applied when 1) different predicates are over different attributes without equality (=) operator; and 2) all the constraints are relaxation constraints or all are contraction constraints, i.e., all inequalities are of the form $C(D)_{\mathcal{G}}\ op\ x$, where x is a constant and $op \in \{>, \geq\}$ (or $op \in \{<, \leq\}$) for *all* the constraints. For simplicity of presentation, in what follows, we present the case of relaxation. The case of contraction is symmetric.

First of all, note that tuples in D that already satisfy the cardinality constraints do not impact the refinement process and can therefore be excluded from both the provenance inequalities and the PVL. Furthermore, when all constraints are relaxation constraints,

the process of finding a minimal refinement from a partial query and the subsequent recursive searches can be optimized, owing to the monotonicity property of relaxation queries in the following proposition.

Proposition 5.5. *Let D be a dataset, Q be a query, and $L_{D,Q,Cr}$ be the corresponding PVL which consists of n_c lists. Let $\mathcal{I}^{Cr}_{Q(D)}$ be a set of provenance inequalities such that Cr are all relaxation constraints. Let Q_1, Q_2 be two different relaxations such that $Tval_{Q_1}(\mathcal{I}^{Cr}_{Q(D)}) = True$ and Q_1 dominates Q_2. Then $Tval_{Q_2}(\mathcal{I}^{Cr}_{Q(D)}) = True$.*

Proof. Without loss of generosity, assume for all numerical predicates p, $p.op$ is $\geq$, i.e., they are all of the form $p.A \geq p.c$. To prove the proposition, we only need to show that $\forall A_v \in \mathcal{V}_{Q(D)}$ s.t. $val_{Q_1}(A_v) = 1$, we have $val_{Q_2}(A_v) = 1$. Let p_1, p_2 be the predicate corresponding to A_v in Q_1, Q_2, respectively. First, when A is a numerical attribute, based on Proposition 4.4, with list l_A, we have $Q_1.R[l_A] \leq Q_2.R[l_A]$, thus $p_1.c \geq p_2.c$. Based on Definition 3.7, $val_{Q_1}(A_v) = 1$ means $v \geq p_1.c$, so $v \geq p_2.c$, thus $val_{Q_2}(A_v) = 1$. Second,when A is a categorical attribute, with list l_A, based on Proposition 4.4, we assume $Q_2.R[l_A] = 2, Q_1.R[l_A] = 1$ (because other cases are trivial), so $p_2.C \subset p_1.C$. Based on Definition 3.7, $val_{Q_2}(A_v) = 1$ means $v \in p_1.C$, so $v \in p_2.C$, thus $val_{Q_2}(A_v) = 1$. □

Based on the above proposition, minimal refinements can be efficiently identified. Rather than employing a top-down search that traverses all potential refinements, we optimize this procedure by implementing a binary search to locate a refinement that satisfies the cardinality constraints. Following that, we "tighten" the refinement to make it minimal. This process is depicted in Algorithm 2. We first use a binary search (line 2) to obtain an initial query Q_b such that $\forall l \in L, Q_b.R[l] = k$ and $val_{Q_b}(\mathcal{I}^{Cr}_{Q(D)}) = True$. For lists l with length smaller than k, we set $Q.R[l]$ to be the length of l. Next, we tighten Q_b – make it minimal by moving up the index values in each list as much as possible (lines 4 – 10). The refinement we get at the end of the process is a minimal relaxation.

Proposition 4.4 also provides us with another optimization so that some recursions can be avoided. When we recursively search for other relaxations by fixing a predicate with values above $Q_b.R$, (lines 7 – 12 in Algorithm 2), if there is a value $l[i]$ that fixing it ends up adding no new minimal relaxation to the result set, we do not need to check values above index i in list l. In other words, from line 7 to line 12 in Algorithm 2, if one of the iterations in the for loop does not update the result set *Res* with any new minimal relaxations, we execute a "break" after line 12 to end the for loop in advance, because the following iterations would check values that relax the query less, and based on monotonicity, will definitely not satisfy the cardinality constraints.

Example 5.6. Consider again Example 1.1 with the original query `Q1`. To show a simple example with relaxation constraints, we only consider the constraint that there should be at least two females selected. As mentioned before, when all constraints are relaxation constraints, provenance inequalities can be simplified by removing tuples satisfying query already, so the provenance expression is

$$M_{ME} \cdot G_{3.65} + M_{CS} \cdot G_{3.40} + M_{ME} \cdot G_{3.60} + M_{EE} \cdot G_{3.85} + M_{EE} \cdot G_{3.90} + M_{EE} \cdot G_{3.85} \geq 1 \quad (3)$$

Algorithm 2: Find a minimal relaxation

input : A PVL L and cardinality constraints $\mathcal{I}$.
output: A minimal relaxation $\mathcal{R}$

```
1  Procedure findRelaxation(L, I, Q|_P)
2      Q_b ← binarySearch(L, I, Q|_P)
3      if Q_b.R ≠ ∅ then
4          foreach l ∈ L do
5              foundMin ← False
6              while Q_b.R[l] ≥ 1 or foundMin do
7                  Q_b.R[l] ← Q_b.R[l] − 1
8                  if Tval_{Q_b}(I) = False then
9                      Q_b.R[l] ← Q_b.R[l] + 1
10                     foundMin ← True
11     return Q_b
```

Q_1^H	Q_2^H
SELECT * FROM Healthcare WHERE	
income >= **100K** and **num-children** >= **3** and **county in ("county2", "county3")**	**income** >= **150K** and **num-children** <= **4** and **complications** <= **8** and **county in ("county2", "county4")**
Q_1^A	Q_2^A
SELECT * FROM ACSIncome WHERE	
working_hours >= **40** and **Educational_attainment** >= **19** and **Class_of_worker in ("local_gov", "state_gov", "federal_gov")**	**working_hours** <= **40** and **Educational_attainment** <= **19** and **income in ("<20K", "20K-40K")**
Q_3^T	
SELECT * FROM customer, order, lineitem WHERE c_custkey = o_custkey and l_orderkey = o_orderkey and **c_mktsegment = 'BUILDING'** and **o_orderdate < date 1995-03-28** and **l_shipdate > date 1995-03-28**	
Q_{12}^T	
SELECT * FROM order, lineitem WHERE l_orderkey = o_orderkey and l_commitdate < l_receiptdate and l_shipdate < l_commitdate **l_shipmode in ('RAIL', 'AIR')** and **l_receiptdate >= date 1995-01-01** and **l_shipdate < date 1996-01-01**	

Figure 3: Queries used in experiments

Compared to Figure 2a, the PVL in Figure 2c does not contain list $l_{M_{CS}}$ and values greater than 3.85 in list l_G since they already satisfy `Q1`. To get an initial relaxation Q_b, we do a binary search and find that 2 is the smallest index k such that $\forall l \in L, Q_b.R[l] = k$ and $val_{Q_b}(\mathcal{I}^{Cr}_{Q1(D)}) = True$. Then we tighten Q_b by moving up values in each list as much as possible, and get $Q'_b.R = [2, 1, 1]$, highlighted in blue color, which is a minimal relaxation that adds EE to be a recognized major. Then, we fix the value in $l_{M_{EE}}[1]$ and search for other relaxations recursively, and find another two minimal relaxations: (1) modifying GPA predicate to be `a.GPA >= 3.40`; and (2) adding ME to the major list, and modifying GPA predicate to be `a.GPA >= 3.65`.

6 EXPERIMENTS

We experimentally evaluate the proposed solutions, showing the usefulness and effectiveness of our approach. We start with our experiment setup and then present a quantitative experimental study whose goal is to assess the efficiency of our algorithm. In particular, we examine our algorithm's performance with different datasets, queries, and cardinality constraints, and compare it to

C_1^H	C_2^H	C_3^H
$\{race = race2\}$ $\{race = race1, age = group1\}$	$\{race = race2\}$ $\{age = group2\}$	$\{race = race2\}$ $\{age = group2\}$
C_1^A	C_2^A	C_3^A
$\{sex = F,$ $marital = Married\}$ $\{race = Black\}$	$\{sex = M, relationship$ $= husband/wife\}$ $\{age = 30 - 60,$ $marital = divorced\}$	$\{sex = M, race = Asian\}$ $\{sex = F, marital = Married\}$
$C_1^{T,3}$	$C_2^{T,3}$	$C_3^{T,3}$
$\{c\_nationkey = 23\}$ $\{l\_shipmode = MAIL\}$	$\{c\_nationkey = 2\}$ $\{o\_orderstatus = P,$ $l\_returnflag = N\}$ $\{l\_shipmode = MAIL\}$ $\{l\_shipinstruct =$ $COLLECT\ COD\}$ $\{l\_linenumber = 5\}$	$\{l\_linenumber = 6,$ $l\_shipmode = SHIP,$ $o\_orderstatus = F,$ $o\_orderpriority = 1 - URGENT\},$ $\{l\_shipinstruct =$ $DELIVER\ IN\ PERSON,$ $o\_orderstatus = P,$ $o\_orderpriority = 2 - HIGH\}$
$C_1^{T,12}$	$C_2^{T,12}$	$C_3^{T,12}$
$\{o\_orderpriority = 5 - LOW,$ $l\_linenumber = 3\},$ $\{l\_shipinstruct =$ $COLLECT\ COD\},$ $\{l\_returnflag = A\}$	$\{o\_orderpriority =$ $1 - URGENT\},$ $\{l\_linenumber = 1\}$ $\{l\_returnflag = A,$ $o\_orderstatus = F\};$	$\{l\_linenumber = 7,$ $l\_shipinstruct = NONE,$ $o\_orderpriority = 5 - LOW,$ $o\_orderstatus = F\},$ $\{o\_orderpriority =$ $4 - NOT\ SPECIFIED,$ $l\_shipinstruct =$ $TAKE\ BACK\ RETURN,$ $l\_linenumber = 7,$ $l\_returnflag = A\}$

Figure 4: Constraints used in experiments

a naive solution. We further study the effect of the data size, the selectivity of the given query, and the constraint properties on the algorithm's running time. We show the effect of the optimization presented in Section 5.2 and conclude with a comparison to [14] using a case study. The results show our solution can perform well across a wide range of datasets, queries, and cardinality constraints. Our PVL-based search algorithm runs up to 100 times faster than a naive traversal over the possible refinements, and the optimization can further achieve a gain of up to 99.87%.

6.1 Experiment Setup

Datasets: We use three different datasets to do experiments: Healthcare, ACSIncome, and TPC-H benchmark.

- **The Healthcare dataset**[2] is a dataset used in [16] to train a classifier identifying patients at risk for serious complications. It contains demographic and clinical history information of 887 patients, with attributes like number of children, income, number of complications, county, race, age, etc..
- **ACSIncome dataset** is one of five datasets created by [17] as an improved alternative to the popular UCI Adult dataset. The data was compiled from the American Community Survey (ACS) Public Use Microdata Sample (PUMS). It covers all 50 states and Puerto Rico, with 1,664,500 rows and 11 features.
- **TPC-H benchmark**[3] (Transaction Processing Performance Council Benchmark H) is a standard benchmark for measuring the performance of relational database management systems (RDBMS) and data warehousing systems. The benchmark consists of a suite of 22 SQL queries and a database schema with 8 relational tables, and is designed to simulate the decision support systems of a typical data warehousing environment. We use TPC-H 3.0.1 to generate datasets and queries 100M data size.

Queries: We generated two queries each for Healthcare and ACSIncome dataset (denoted as $Q_1^H, Q_2^H, Q_1^A, Q_2^A$) without JOIN operations as each dataset has a single relation. From TPC-H benchmark queries, we randomly chose #3 and #12 (Q_3^T, Q_{12}^T), which both include JOIN operations. The TPC-H query was simplified by removing GROUP BY, ORDER BY, and LIMIT clauses which are not considered in our setting and only affect tuple display. This modification doesn't affect the experiment's validity. Additionally, we replaced attribute sets in the query projections with `SELECT *` for simplicity and to ensure all attributes in the cardinality constraints were included. One predicate over l_receiptdate in query #12 is replaced with attribute l_shipdate, which enables us to explore different scenarios and apply optimizations. An overview of all queries used in the experiments is presented in Figure 3.

Cardinality constraints: Figure 4 summarizes the constraints used in experiments. We form three sets of cardinality constraints for each of the Healthcare, ACSIncome, and TPC-H datasets, by using representative sensitive attributes such as race and gender for the first two demographic datasets, and attributes from multiple tables that are not used in queries for non-demographic TPC-H dataset. Each set contains 2 to 5 constraints that must be fulfilled simultaneously. The first set of constraints for each dataset contains relaxation constraints, the second contains contraction constraints, and the third contains a combination of both: first group being relaxed to 105% and second being contracted to 95%. Similar results were observed for other queries and constraints (see [18]).

Compared algorithms: We compare our optimized solution to the baseline approach, where both operate on in-memory data and utilize provenance expressions to evaluate potential refinements. This comparison is made because we recognize that managing data on an external database would require multiple I/O accesses using the baseline approach, which makes the baseline algorithm even less efficient. Our algorithm requires only a single access to generate the provenance inequalities.

- **PVL-based Search (PS).** The generalized algorithm searches for minimal refinements using the PVL and provenance inequalities described in Section 5.
- **Baseline Algorithm (Baseline).** The baseline (naive) algorithm for minimal refinements searching is described at the beginning of Section 4. It uses provenance model to avoid multiple database accesses, but searches for minimal refinements by traversing all possible refinements.

Platform: All experiments were performed on a macOS Ventura 13.2 machine with a Apple M2 Max chip, 64 GB memory. The algorithms were implemented using Python3.

6.2 Experiment Results

We first conduct experiments to compare the running time of our algorithm with that of the baseline. Subsequently, we investigate the impact of various parameters on our algorithm's running time by systematically altering them, enabling us to identify the scalability of our algorithm.

[2]https://github.com/stefan-grafberger/mlinspect/tree/master/example_pipelines/healthcare

[3]https://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp

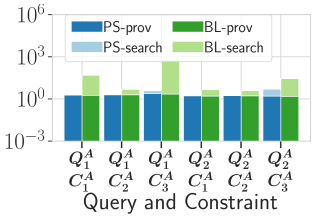


Figure 5: ACSIncome

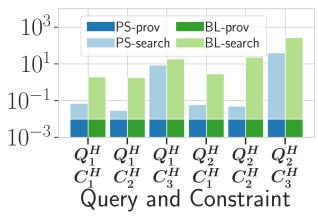
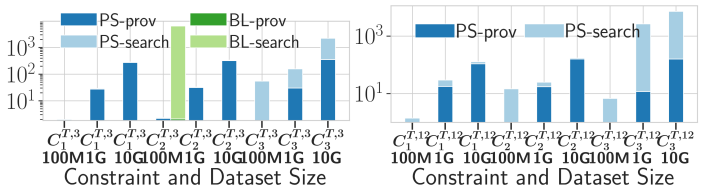
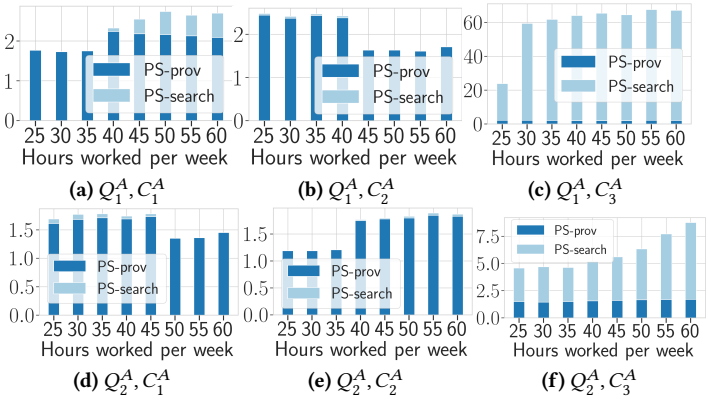


Figure 6: Healthcare



(a) Q_3^T **(b)** Q_{12}^T

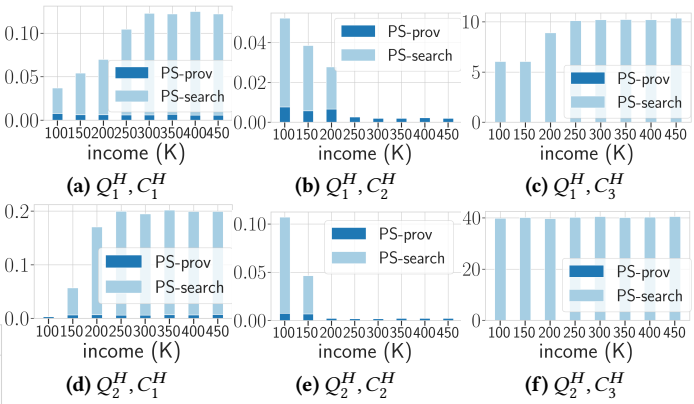
Figure 7: TPC-H benchmark



(a) Q_1^A, C_1^A **(b)** Q_1^A, C_2^A **(c)** Q_1^A, C_3^A

(d) Q_2^A, C_1^A **(e)** Q_2^A, C_2^A **(f)** Q_2^A, C_3^A

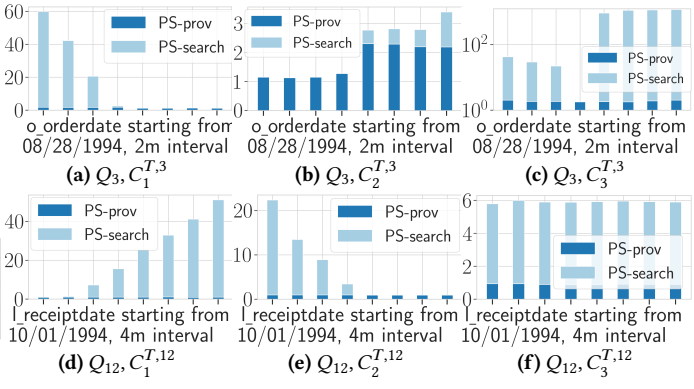
Figure 8: Effect of query selectivity on the running time, ACSIncome data, x-axis shows different constants in a predicate, y-axis is running time (s)



(a) Q_1^H, C_1^H **(b)** Q_1^H, C_2^H **(c)** Q_1^H, C_3^H

(d) Q_2^H, C_1^H **(e)** Q_2^H, C_2^H **(f)** Q_2^H, C_3^H

Figure 9: Effect of query selectivity on the running time, Healthcare data, x-axis shows different constants in a predicate, y-axis is running time (s)



(a) $Q_3, C_1^{T,3}$ **(b)** $Q_3, C_2^{T,3}$ **(c)** $Q_3, C_3^{T,3}$

(d) $Q_{12}, C_1^{T,12}$ **(e)** $Q_{12}, C_2^{T,12}$ **(f)** $Q_{12}, C_3^{T,12}$

Figure 10: Effect of query selectivity on the running time, TPC-H benchmark, 100M data sizes, x-axis shows different constants in a predicate, y-axis is running time (s)

Running time. The first set of experiments assesses the running time for different scenarios by executing our algorithm and comparing its running with the baseline naive algorithm. In all tested scenarios, our algorithm outperforms a baseline naive algorithm by more than 100 times, as shown in Figures 5, 6, and 7. The blue bar depicts the running time of our solution (provenance search), and the green bar shows the running time for the baseline algorithm. The dark section represents the provenance generation time, and the light blue section shows the searching time for minimal refinements. The x-axis displays the query and constraint used, while the y-axis shows the running time in seconds. In some experiments the green bar is omitted because the program of baseline approach cannot terminate within a 3-hour time-out. Our solution performs well with various flavors of constraint sets, including relaxation, contraction, and a mix of both, because our solution searches through fewer refinements using the PVL. The running time for different query-constraint combinations varies significantly, because the running time is affected by many factors such as the search space and the distance between the original query and queries satisfying cardinality constraints, as we demonstrate next.

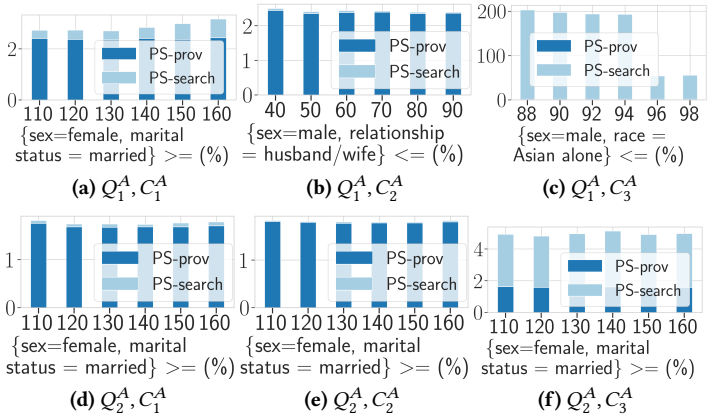
Data size. We examined the effect of data size on running time using the TPC-H dataset, and present our findings in Figure 7, which shows the running time for data sizes of 100M, 1G, and 10G. Data size affects running time in two primary ways: the time to generate provenance expressions and the search space size. The time needed to generate provenance expressions and assign values to them scales linearly with the data size, while the size of the Provenance Value List (PVL) generally with the number of attribute values in larger datasets. Due to these factors, for each query and constraints pair, we observed a clear trend of increasing running time as the data size increased.

Query Selectivity. To examine the effect of query selectivity, we varied the value of a single constant in the predicate of each query. The results are shown in Figures 8, 9, 10, where the x-axis is the value of the constant in the predicate being varied, and the y-axis is the running time in seconds. Intuitively, the selectivity of the query (with respect to the constraints) determines the search space the algorithm traverses to find refinement queries. When increasing the selectivity, we expect the search space to increase for relaxation,

as it starts from refinements close to the original query. Thus we expect to see an increase in the running time, as in Figures 8a, 8e, 9a, 9d, 10b, 10d. For contraction constraints, we expect to see an opposite trend, as in Figures 8b, 8d, 9b, 9e, 10a, 10d. In the general case (constraint set $C_3^A, C_3^H, C_3^{T,3}, C_3^{T,12}$), increasing the constant in a predicate does not affect the size of the provenance expression which contains the entire data, but it does impact the relaxation and contraction constraints in opposite ways, resulting in a trade-off between the constraints. The running time is primarily influenced by the contraction constraint for lower selectivity, while for higher selectivity, it is mainly devoted to meeting the relaxation constraint, so there can be a turning point. A similar trend is observed in Figure 10c . However, some figures, such as Figure 8c, 8f, 9c, 9f, 10f, display a monotonic increase or decrease without a turning point, which is due to the satisfaction of either the contraction or relaxation constraints dominating the overall running time of the algorithm.

Constraints satisfaction properties. To analyze the effect of the cardinality constraint on the running time, we varied the ratio to which one group is relaxed or contracted in the constraint while keeping the ratio of the other groups unchanged and observed its impact. Specifically, for relaxation constraints, we relax one of the groups to 110%, 120%, 130%, 140%, 150%, 160%; for contraction constraints, we contract one of the groups to 40%, 50%, 60%, 70%, 80%, 90%; and for refinement constraints, we either relax one group to 102%, 104%, 106%, 108%, 110%, 112%, or contract one group to 88%, 90%, 92%, 94%, 96%, 98%. Altering the constant in the cardinality constraint does not affect the size of the Provenance Value List (PVL), as it depends on the dataset, query, and groups definition in constraints rather than constants in constraints. We intuitively expect that as constraints are tightened or relaxed, the running time of algorithms will increase or decrease respectively, as queries satisfying the constraints move further or closer to the original query. Figures 11, 12, and 13 display the results of these experiments for three datasets. We observed an increase in the running time when constraints were more difficult to satisfy, as in Figure 11a, 12a, 12c, 12d, 12f, 13a, 13d. Opposite trends are observed in other figures when the queries becomes more and more easy to satisfy. But some figures, like Figure 11b, 11d, 11e, 11f, 13b, 13e, 13f. exhibit negligible changes in running time when a constraint changes, indicating that the impact of these changes is minor. This can be attributed to the uneven distribution of data groups in the constraints, where adjusting a predicate to a particular value can qualify or disqualify multiple tuples simultaneously, satisfying various relaxation/contraction ratios. This effect is particularly noticeable when there are many tuples with the same value in an attribute, resulting in minimal refinements that are almost the same for different relaxation/contraction ratios.

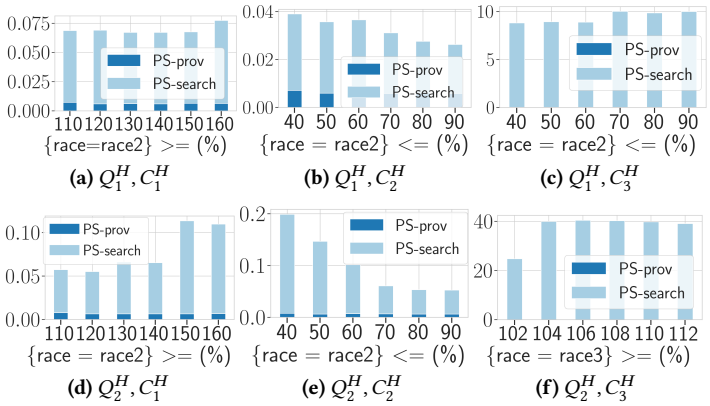
Effect of optimizations. To assess the impact of the optimizations for the searching algorithm described in Section 5.2, we conducted experiments using algorithms both with and without optimizations and compared their search times. These optimizations incorporate the use of binary search instead of a top-down traversal when searching for a refinement, exploiting the monotonicity property of queries when all constraints are relaxation constraints or all are contraction constraints. We used queries $(Q_1^H, Q_2^H, Q_1^A, Q_2^A, Q_3^T, Q_{12}^T)$ and the first two sets of constraints



(a) Q_1^A, C_1^A (b) Q_1^A, C_2^A (c) Q_1^A, C_3^A

(d) Q_2^A, C_1^A (e) Q_2^A, C_2^A (f) Q_2^A, C_3^A

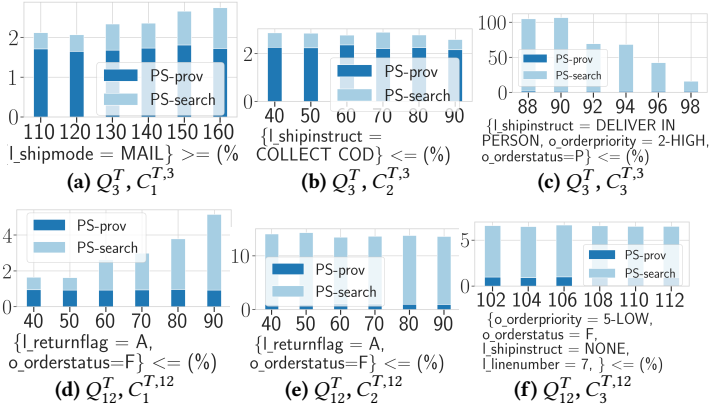
Figure 11: Effect of constraints on the running time, ACSIncome dataset, y-axis is running time (s)



(a) Q_1^H, C_1^H (b) Q_1^H, C_2^H (c) Q_1^H, C_3^H

(d) Q_2^H, C_1^H (e) Q_2^H, C_2^H (f) Q_2^H, C_3^H

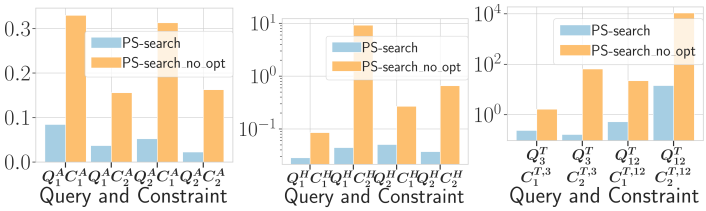
Figure 12: Effect of constraints on the running time, Healthcare dataset, y-axis is running time (s)



(a) $Q_3^T, C_1^{T,3}$ (b) $Q_3^T, C_2^{T,3}$ (c) $Q_3^T, C_3^{T,3}$

(d) $Q_{12}^T, C_1^{T,12}$ (e) $Q_{12}^T, C_2^{T,12}$ (f) $Q_{12}^T, C_3^{T,12}$

Figure 13: Effect of constraints on the running time, datasize, x-axis shows how much groups are relaxed/contracted, y-axis is running time (s)



(a) ACSIncome dataset **(b) Healthcare dataset** **(c) TPC-H 100M dataset**

Figure 14: Effect of optimizations on the searching time, y-axis is searching time (s)

$(C_1^H, C_2^H, C_1^A, C_2^A, C_1^{T,3}, C_2^{T,3}, C_1^{T,12}, C_2^{T,12})$ for the three datasets, and the results are shown in Figure 14. The light blue bar represents the searching time with optimization, and the light orange bar represents the time without optimization. Our results indicate that the optimizations significantly reduced execution times across all tested scenarios and enhance the efficiency of the algorithm with gains of up to 99.87%.

6.3 Comparison with Related Work

We compared our algorithm with the one by Shetiya *et al.* [14], which also addresses group cardinality constraints but with different objectives and settings. To demonstrate the differences between the two algorithms, we conducted experiments with numerical predicates and a single binary sensitive attribute, which both algorithms can accommodate. We use TexasTribune dataset (used by [14]), containing salary information for Texas state employees. To be consistent with [14] and allow for easy comparison, we imposed both upper bound and lower bound for salary attribute as the query and used gender as a sensitive attribute to form our cardinality constraint. Specifically, we use query Q with two predicates `a.salary >= 10000` and `a.salary <= 20000` (for [14], the query selects tuples with salaries within the range [10000, 20000]), resulting in a gender disparity of 69, with 100 males and 169 females in the result set. As for cardinality constraints, since [14] focuses on the disparity between tuples determined by a single binary attribute, we require the gender disparity to be no larger than 65 ($\epsilon = 65$ for [14]), resulting in constraint: $|Q(D)_{gender=M} - Q(D)_{gender=F}| \leq 65$.

The refinement returned by [14] is a query with salary range $r_1 = [10135.7, 19759.9]$, (Jaccard similarity of 0.96). Our algorithm returns additional four refinements $r_2 = [11138.4, 20000.0]$, $r_3 = [10000.0, 19723.8]$, $r_4 = [10764.0, 19999.2]$, and $r_5 = [11036.2, 19999.9]$ (with Jaccard similarity of 0.91, 0.96, 0.95, 0.93, respectively). Note that all the returned refinements are minimal according to Definition 2.10. The differences in the results are due to the minimality definitions: distance between queries versus distance in the result sets. Our definition allows the user to select between multiple refinements according to their preferences. For example, if the user prefers to minimize change to the lower bound of the salary, r_3 would be the best refinement.

7 RELATED WORK

We next overview multiple lines of related work.

Query refinement. The problem of query refinement was studied, e.g., in [3, 4, 14, 19–22], with methods for slightly modifying the query so that output size satisfies cardinality constraints. Typically, the focus is on the size of the whole output rather than the size of some data groups in the output. For instance, [4, 19] focus on relaxing queries with an empty result set in order to yield some answers. Other works [3, 20] studied the too many/few answers problems, where the goal is to refine the query to satisfy some cardinality constraint on the result's size. Recent work by Shetiya et al. [14] is an exception in that it aims to refine queries to satisfy constraints on the size of some data groups in the result rather than the cardinality of the entire result. We discuss and demonstrate the differences in Section 6.3. The work of [23, 24] also aims to satisfy cardinality constraints through minimal query refinement, but it does not consider categorical attributes and provides only cardinality estimations without exact result.

The work of [21, 22] studied the problem of explaining missing tuples in the query result through query refinement. A key difference is that [21, 22] aim to include given user-specified missing tuples in the result set by the refinement rather than imposing constraints over groups, where including or removing different (not specified by the user) tuples may apply.

Query result diversification. Query result diversification is another related topic aiming to increase the diversity in the query result while maintaining the relevance of results to the original query [25–28]. In some settings, relevance and diversity are tradeoffs [29], and in others, user information is used to select the result set. However, two main differences exist between this line of work and ours. First, there is a ranking in query result diversification: items are ranked by their relevance to the original query. The goal is to select items that are ranked as high as possible and, at the same time, as diverse as possible. On the contrary, there is no ranking in our setting: an item is either selected or not by the query. Second, while query result diversification studies how to select items based on the given query without changing the query, we achieve a diverse result set by refining the query itself.

Provenance. Data provenance was studied extensively in recent years. An early approach to user provenance for RDBMS [30, 31] finds a set of tuples that contribute to a certain output tuple. The model proposed in [32] collects contributing input tuples separately for each step in the derivation of the output tuple so that it shows how the output is obtained. Then a general framework is proposed in [12] that annotates tuples with elements from a semiring K so that queries can be evaluated over annotated relations. The Caravan system proposed in [5] enables what-if analysis by creating provisioned representations to efficiently compute hypothetical reasoning so that what-if scenarios can be evaluated without accessing the original data or executing complex queries. Our provenance model is inspired by [5], with tuple annotations propagated through query valuation in [12].

Fairness constraints in other settings. While our work does not only apply to fairness context, fairness is an important topic in data selection and there are many prior works focusing on satisfying fairness constraints in some settings. Some works aim to satisfy fairness constraints under a certain diversity model [33–35], and others focus on fairness constraints in the optimization of an additive utility [36]. There is also some work on fairness in set selection

and ranking [37], fairness in classification [38], and diversity in top-k results [39–41]. Other works consider constraint query languages [42], presenting a declarative language with extensions to SQL to support fairness constraints in queries [43].

8 CONCLUSION

In this paper, we have studied the problem of finding all minimal refinements of a given query so that the result set satisfies given cardinality constraints that are on the size of some data groups in the result. Our proposed solution uses a provenance model to annotate tuples in the source data with the necessary information with respect to the predicate and translates cardinality constraints to algebraic expressions, so that whether or not constraints are satisfied can be checked without checking the original data. We also propose a search algorithm that efficiently explores the search space and finds potential refinements, which are then verified using our proposed provenance-based approach. Our experimental results demonstrate the effectiveness and efficiency of our solution on a variety of datasets and constraints.

REFERENCES

[1] https://www.microsoft.com/en-us/research/academic-program/phd-fellowship/canada-us/.

[2] https://research.google/outreach/faq/?category=phd.

[3] Chaitanya Mishra and Nick Koudas. Interactive query refinement. In *EDBT*, 2009.

[4] Nick Koudas, Chen Li, Anthony K. H. Tung, and Rares Vernica. Relaxing join and selection queries. In *VLDB*, 2006.

[5] Daniel Deutch, Zachary G. Ives, Tova Milo, and Val Tannen. Caravan: Provisioning for what-if analysis. In *CIDR*. www.cidrdb.org, 2013.

[6] Babak Salimi, Johannes Gehrke, and Dan Suciu. Bias in OLAP queries: Detection, explanation, and removal. In *SIGMOD*, 2018.

[7] Bienvenido Vélez, Ron Weiss, Mark A Sheldon, and David K Gifford. Fast and effective query refinement. In *ACM SIGIR Forum*, volume 31, pages 6–15. ACM New York, NY, USA, 1997.

[8] Jessie Ooi, Xiuqin Ma, Hongwu Qin, and Siau Chuin Liew. A survey of query expansion, query suggestion and query refinement techniques. In *2015 4th International Conference on Software Engineering and Computer Systems (ICSECS)*, pages 112–117. IEEE, 2015.

[9] Michael Ortega-Binderberger, Kaushik Chakrabarti, and Sharad Mehrotra. An approach to integrating query refinement in sql. In *International Conference on Extending Database Technology*, pages 15–33. Springer, 2002.

[10] Kaushik Chakrabarti, Kriengkrai Porkaew, and Sharad Mehrotra. Efficient query refinement in multimedia databases. In *ICDE Conference*, 2000.

[11] Yannis Papakonstantinou and Vasilis Vassalos. Query rewriting for semistructured data. *ACM SIGMOD Record*, 28(2):455–466, 1999.

[12] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*. ACM, 2007.

[13] Marina Drosou, HV Jagadish, Evaggelia Pitoura, and Julia Stoyanovich. Diversity in big data: A review. *Big data*, 5(2):73–84, 2017.

[14] Suraj Shetiya, Ian P Swift, Abolfazl Asudeh, and Gautam Das. Fairness-aware range queries for selecting unbiased data. In *Proc. of the Int. Conf. on Data Engineering, ICDE*, 2022.

[15] Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.

[16] Stefan Grafberger, Shubha Guha, Julia Stoyanovich, and Sebastian Schelter. Mlinspect: A data distribution debugger for machine learning pipelines. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2736–2739, 2021.

[17] Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. Retiring adult: New datasets for fair machine learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[18] https://github.com/JinyangLi01/Query_refinement/blob/master/FullPaper/Query_Refinement.pdf.

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

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

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

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

[23] Chiara Accinelli, Barbara Catania, Giovanna Guerrini, and Simone Minisi. covrew: a python toolkit for pre-processing pipeline rewriting ensuring coverage constraint satisfaction. In *EDBT*, pages 698–701, 2021.

[24] Chiara Accinelli, Simone Minisi, and Barbara Catania. Coverage-based rewriting for data preparation. In *EDBT/ICDT Workshops*, 2020.

[25] Marcos R Vieira, Humberto L Razente, Maria CN Barioni, Marios Hadjieleftheriou, Divesh Srivastava, Caetano Traina, and Vassilis J Tsotras. On query result diversification. In *2011 IEEE 27th International Conference on Data Engineering*, pages 1163–1174. IEEE, 2011.

[26] Kaiping Zheng, Hongzhi Wang, Zhixin Qi, Jianzhong Li, and Hong Gao. A survey of query result diversification. *Knowledge and Information Systems*, 51(1):1–36, 2017.

[27] Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*, pages 784–791, 2008.

[28] Reinier H Van Leuken, Lluis Garcia, Ximena Olivares, and Roelof van Zwol. Visual diversification of image search results. In *Proceedings of the 18th international conference on World wide web*, pages 341–350, 2009.

[29] Ting Deng and Wenfei Fan. On the complexity of query result diversification. *Proceedings of the VLDB Endowment*, 6(8):577–588, 2013.

[30] Yingwei Cui and Jennifer Widom. Practical lineage tracing in data warehouses. In *Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)*, pages 367–378. IEEE, 2000.

[31] Yingwei Cui, Jennifer Widom, and Janet L Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Transactions on Database Systems (TODS)*, 25(2):179–227, 2000.

[32] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and where: A characterization of data provenance. In *International conference on database theory*, pages 316–330. Springer, 2001.

[33] Zafeiria Moumoulidou, Andrew McGregor, and Alexandra Meliou. Diverse data selection under fairness constraints. *arXiv preprint arXiv:2010.09141*, 2020.

[34] Elisa Celis, Vijay Keswani, Damian Straszak, Amit Deshpande, Tarun Kathuria, and Nisheeth Vishnoi. Fair and diverse dpp-based data summarization. In *International Conference on Machine Learning*, pages 716–725. PMLR, 2018.

[35] Ke Yang, Vasilis Gkatzelis, and Julia Stoyanovich. Balanced ranking with diversity constraints. *arXiv preprint arXiv:1906.01747*, 2019.

[36] Julia Stoyanovich, Ke Yang, and HV Jagadish. Online set selection with fairness and diversity constraints. In *Proceedings of the EDBT Conference*, 2018.

[37] Meike Zehlike, Ke Yang, and Julia Stoyanovich. Fairness in ranking, Part I: Score-based ranking. *ACM Comput. Surv.*, apr 2022.

[38] Alexandra Chouldechova and Aaron Roth. A snapshot of the frontiers of fairness in machine learning. *Commun. ACM*, 63(5):82–89, 2020.

[39] Albert Angel and Nick Koudas. Efficient diversity-aware search. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 781–792, 2011.

[40] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. Diversifying top-k results. *arXiv preprint arXiv:1208.0076*, 2012.

[41] Julia Stoyanovich, Sihem Amer-Yahia, and Tova Milo. Making interval-based clustering rank-aware. In Anastasia Ailamaki, Sihem Amer-Yahia, Jignesh M. Patel, Tore Risch, Pierre Senellart, and Julia Stoyanovich, editors, *EDBT 2011, 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21-24, 2011, Proceedings*, pages 437–448. ACM, 2011.

[42] Paris C Kanellakis, Gabriel M Kuper, and Peter Z Revesz. Constraint query languages. *Journal of computer and system sciences*, 51(1):26–52, 1995.

[43] Matteo Brucato, Azza Abouzied, and Alexandra Meliou. Package queries: efficient and scalable computation of high-order constraints. *The VLDB Journal*, 27(5):693–718, 2018.