Query by Output

Quoc Trung Tran Chee-Yong Chan Srinivasan Parthasarathy*
National University of Singapore The Ohio State University
{tqtrung, chancy}@comp.nus.edu.sg srini@cse.ohio-state.edu

ABSTRACT

It has recently been asserted that the usability of a database is as important as its capability. Understanding the database schema, the hidden relationships among attributes in the data all play an important role in this context. Subscribing to this viewpoint, in this paper, we present a novel data-driven approach, called *Query By Output (QBO)*, which can enhance the usability of database systems. The central goal of QBO is as follows: given the output of some query Q on a database D, denoted by $Q(D)$, we wish to construct an alternative query Q' such that $Q(D)$ and $Q'(D)$ are instance-equivalent. To generate instance-equivalent queries from $Q(D)$, we devise a novel data classification-based technique that can handle the *at-least-one semantics* that is inherent in the query derivation. In addition to the basic framework, we design several optimization techniques to reduce processing overhead and introduce a set of criteria to rank order output queries by various notions of utility. Our framework is evaluated comprehensively on three real data sets and the results show that the instance-equivalent queries we obtain are interesting and that the approach is scalable and robust to queries of different selectivities.

Categories and Subject Descriptors

H.2.4 [**System**]: Query processing; H.2.8 [**Database Applications**]: Data mining

General Terms

Algorithms, Design

Keywords

query by output, instance-equivalent queries, at-least-one semantics

*This work was done when the author was on sabbatical at the National University of Singapore (NUS) in 2008.

SIGMOD'09, June 29–July 2, 2009, Providence, Rhode Island, USA.

1. INTRODUCTION

A perennial challenge faced by many enterprises is the management of their increasingly large and complex databases which can contain hundreds and even thousands of tables [10, 23]. The problem is exacerbated by the fact that the metadata and documentation for these databases are often incomplete or missing [11]. Several different approaches have been proposed to address this important practical issue. One example here is *database structure mining* where the goal is to discover structural relationships among the database tables [1, 11, 23]. Another example is *intensional query answering* where the goal is to augment query answers with additional information to help users understand the results as well as relevant database content [15].

In this paper, we present a novel data-driven approach called *Query by Output* (`QBO`). `QBO` aims to derive interesting query-based characterizations of an input database table which can be the result of some query or materialized view. In contrast to conventional querying which takes an input query Q and computes its output, denoted by $Q(D)$, w.r.t. an input database D, the basic idea of `QBO` is to take as input the output $Q(D)$ of some query Q and compute a set of queries $Q'_1, \cdots, Q'_n$ such that each $Q'_i(D)$ is (approximately) equivalent to $Q(D)$. We say that two queries Q and Q' are *instance-equivalent* w.r.t. a database D (denoted by $Q \equiv_D Q'$), if $Q(D)$ and $Q'(D)$ are equivalent.

Before we discuss the specific contributions of this paper, we briefly highlight some of the use-case scenarios of our proposal.

`QBO` Applications. The most obvious application of `QBO` is in conventional database querying where Q is known. Consider the scenario when a user submits a query Q to be evaluated on a database D. Instead of simply returning the query result $Q(D)$ to the user, the database system can also apply `QBO` to compute additional useful information about Q and D in the form of instance-equivalent queries. Besides providing alternative characterizations (potential simplifications) of $Q(D)$, IEQs can also help users to better understand the database schema. Specifically, since many enterprise data schema are very complex and large, potentially involving hundreds and even thousands of relations [23], the part of the database schema that is referenced by the user's query may be quite different from that referenced by an IEQ. The discovery of this alternative "path" in the schema to generate an instance-equivalent query can aid the user's understanding of the database schema or potentially help refine the user's initial query.

Another obvious application of `QBO` is that it can help

the user better understand the actual data housed within the database. Unusual or surprising IEQs can be useful for uncovering hidden relationships among the data. In several instances simpler or easier to understand relationships may be uncovered which can again aid in the understanding of the data contained within the complex database and help the user refine future queries posed to the database.

QBO may also have interesting applications in database security where attackers who have some prior domain knowledge of the data may attempt to derive sensitive information. For example, if an attacker is aware of the existing correlation structure in the data, they can easily use this information to formulate two or more separate queries which on paper look very different (e.g. using different selection criteria) but in reality may be targeting the same set of tuples in the database. Such sets or groups of queries can potentially be used to reverse-engineer the privacy preserving protocol in use. Subsequently, sensitive information can be gleaned. As a specific example, consider a protocol such as ϵ-diversity [24] which relies on detecting how similar the current query is with a previous set of queries (history) answered by the database, to determine if the current query can be answered without violating the privacy constraints. The notion of similarity used by such methods relies primarily on the selection attributes and thus such protocols will fail to recognize IEQs that use different selection attributes. Privacy in such protocols will then be breached. Automatically recognizing such IEQs via the methods proposed in this paper and subsequently leveraging this information to enhance such protocols may provide more stringent protection against such kinds of attacks.

Another important class of QBO applications is in scenarios where the input consists of $Q(D)$ but not Q itself. Such scenarios are common in data analysis and exploration applications where the information provided is often incomplete: the data set $Q(D)$, which is produced by some query/view Q, is available but not the query/view [11]. The ability to reverse engineer Q from $Q(D)$ then becomes important. Annotating data with relevant metadata is essential in curated databases [6]. Such reverse engineering is also useful for generating concise query-based summaries of groups of tuples of interest to the user (e.g., dominant tuples selected by skyline queries [4]).

Contributions. In this paper, we introduce the novel problem of QBO and propose a solution, TALOS, that models the QBO problem as a data classification task with a unique property that we term *at-least-one semantics*. To handle data classification with this new semantics, we develop a new dynamic class labeling technique and also propose effective optimization techniques to efficiently compute IEQs. Our experimental evaluation of TALOS demonstrates its efficiency and effectiveness in generating interesting IEQs.

2. OVERVIEW OF OUR APPROACH

The QBO problem takes as inputs a database D, an optional query Q, and the query's output $Q(D)$ (w.r.t. D) and computes one or more IEQs Q', where Q and Q' are IEQs if $Q(D) \equiv_D Q'(D)$. We refer to Q as the *input query*, $Q(D)$ as the *input result*, and Q' as the *output query*.

First, let us state the following theoretical results that we have established for variants of the QBO problem.

THEOREM 1. *Given an input query Q, we define QBO_S to be the problem to find the output query Q' where Q' is a conjunctive query that involves only projection and selection (with predicates in the form "A_i op c", A_i is an attribute, c is constant and op $\in \{<, \leq, =, \neq, >, \geq\}$) such that (1): $Q'(D) \equiv_D Q(D)$ and (2) the number of operators (AND, OR and NOT) used in the selection condition is minimized. Then QBO_S is unlikely to be in P.*

Proof Sketch: We prove Theorem 1 by reducing the Minimization Circuit Size Problem to QBO_S. Details are given in [20]. □

THEOREM 2. *Given an input query Q, we define QBO_G to be the problem to find an output query Q' such that $Q'(D) \equiv_D Q(D)$ and Q' can contain arbitrary arithmetic expressions in the select-clause. Then QBO_G is PSPACE-hard.*

Proof Sketch: We prove Theorem 2 by reducing the Integer Circuit Evaluation Problem to QBO_G. Details are given in [20]. □

Given the above results, in this paper, we consider relational queries Q where the select-clause refer to only attributes (and not to constants or arithmetic/aggregation/string expressions) to ensure that Q' can be derived efficiently from $Q(D)$. We also require that $Q(D) \neq \emptyset$ for the problem to be interesting.

For simplicity, our approach considers only select-project-join (SPJ) queries for Q' where all the join predicates in Q' are foreign-key joins. Thus, our approach requires only very basic database integrity constraint information (i.e., primary and foreign key constraints). Based on the knowledge of the primary and foreign key constraints in the database, the database schema can be modeled as a *schema graph*, denoted by $\mathcal{SG}$, where each node in $\mathcal{SG}$ represents a relation, and each edge between two nodes represents a foreign-key join between the relations corresponding to the nodes.

For ease of presentation and without loss of generality, we express each Q' as a relational algebra expression. To keep our definitions and notations simple and without loss of generality, we shall assume that there are no multiple instances of a relation in Q and Q'.

Running Example. In this paper, we use a database housing baseball statistics[1] for our running example as well as in our experiments. Part of the schema is illustrated in Figure 1, where the key attribute names are shown in bold. The Master relation describes information about each player (identified by *pID*): the attributes *name*, *country*, *weight*, *bats*, and *throws* refer to his name, birth country, weight (in pounds), batting hand (left, right, or both), and throwing hand (left or right), respectively. The Batting relation provides the number of home runs (HR) of a player when he was playing for a team in a specific year and season (stint). The Team relation specifies the rank obtained by a team for a specified year.

Notations. Given a query Q, we use *rel(Q)* to denote the collection of relations involved in Q (i.e., relations in SQL's from-clause); *proj(Q)* to denote the set of projected attributes in Q (i.e., attributes in SQL's select-clause); and *sel(Q)* to denote the set of selection predicates in Q (i.e., conditions in SQL's where-clause).

[1] http://baseball1.com/statistics/

pID	name	country	weight	bats	throws
P1	A	USA	85	L	R
P2	B	USA	72	R	R
P3	C	USA	80	R	L
P4	D	Germany	72	L	R
P5	E	Japan	72	R	R

(a) Master

pID	**year**	**stint**	team	HR
P1	2001	2	PIT	40
P1	2003	2	ML1	50
P2	2001	1	PIT	73
P2	2002	1	PIT	40
P3	2004	2	CHA	35
P4	2001	3	PIT	30
P5	2004	3	CHA	60

(b) Batting

team	**year**	rank
PIT	2001	7
PIT	2002	4
CHA	2004	3

(c) Team

Figure 1: Running Example: Baseball Data Set D

2.1 Instance-Equivalent Queries (IEQs)

Our basic definition of *instance-equivalent queries (IEQs)* requires that the IEQs Q and Q' produce the same output (w.r.t. some database D); i.e., $Q(D) \equiv_D Q'(D)$. The advantage of this simple definition is that it does not require the knowledge of Q to derive Q', which is particularly useful for `QBO` applications where Q is either missing or not provided. However, there is a potential "accuracy" tradeoff that arises from the simplicity of this weak form of equivalence: an IEQ may be "semantically" quite different from the input query that produced $Q(D)$ as the following example illustrates.

EXAMPLE 1. *Consider the following three queries on the baseball database D in Figure 1:*
$Q_1 = \pi_{country}(\sigma_{bats="R" \wedge throws="R"}(Master))$,
$Q_2 = \pi_{country}(\sigma_{bats="R" \wedge weight \leq 72}(Master))$, *and*
$Q_3 = \pi_{country}(\sigma_{bats="R"}(Master))$.
Observe that although all three queries produce the same output after projection ($\{USA, Japan\}$), only Q_1 and Q_2 select the same set of tuples $\{P2, P5\}$ from R. Specifically, if we modify the queries by replacing the projection attribute "country" with the key attribute "pID", we have $Q_1(D) = \{P2, P5\}$, $Q_2(D) = \{P2, P5\}$ and $Q_3(D) = \{P2, P3, P5\}$. Thus, while all three queries are IEQs, we see that the equivalence between Q_1 and Q_2 is actually "stronger" (compared to that between Q_1 and Q_3) in that both queries actually select the same set of relation tuples. □

However, if Q is provided as part of the input, then we can define a stronger form of instance equivalence as suggested by the above example. Intuitively, the stricter form of instance equivalence not only ensures that the instance-equivalent queries produce the same output (w.r.t. some database D), but it also requires that their outputs be projected from the same set of "core" tuples. We now formally characterize weak and strong IEQs based on the concepts of *core relations* and *core queries*.

Core relations. Given a query Q, we say that $S \subseteq rel(Q)$ is a set of *core relations* of Q if S is a minimal set of relations such that for every attribute $R_i.A \in proj(Q)$, (1) $R_i \in S$ or (2) Q contains a chain of equality join predicates "$R_i.A = \cdots = R_j.B$" such that $R_j \in S$.

Intuitively, a set of core relations of Q is a minimal set of relations in Q that "cover" all the projected attributes in Q. As an example, if $Q = \pi_{R_1.X}\sigma_p(R_1 \times R_2 \times R_3)$ where $p = (R_1.X = R_3.Y) \wedge (R_2.Z = R_3.Z)$, then Q has two sets of core relations, $\{R_1\}$ and $\{R_3\}$.

Core queries. Given a query Q where $S \subseteq rel(Q)$, we use Q_S to denote the query that is derived from Q by replacing $proj(Q)$ with the key attribute(s) of each relation in S. If S is a set of core relations of Q, we refer to Q_S as a *core query* of Q.

Strong & weak IEQs. Consider two IEQs Q and Q' (w.r.t. a database D); i.e., $Q(D) \equiv_D Q'(D)$. We say that Q and Q' are *strong IEQs* if Q has a set of core relations S such that (1) ${Q'}_S$ is a core query of Q', and (2) $Q_S(D)$ and ${Q'}_S(D)$ are equivalent. IEQs that are not strong are classified as *weak IEQs*.

The strong IEQ definition essentially requires that both Q and Q' share a set of core relations such that $Q(D)$ and $Q'(D)$ are projected from the same set of selected tuples from these core relations. Thus, in Example 1, Q_1 and Q_2 are strong IEQs whereas Q_1 and Q_3 are weak IEQs.

Note that in our definition of strong IEQ, we only impose moderate restrictions on Q and Q' (relative to the weak IEQ definition) so that the space of strong IEQs is not overly constrained and that the strong IEQs generated are hopefully both interesting as well as meaningful.

As in the case with weak IEQs, two strong IEQs can involve different sets of relations. As an example, suppose query Q selects pairs of records from two core relations, *Supplier* and *Part*, that are related via joining with a (non-core) *Supply* relation. Then it is possible for a strong IEQ Q' to relate the same pair of core relations via a different relationship (e.g., by joining with a different non-core *Manufacture* relation).

We believe that each of the various notions of query equivalence has useful applications in different contexts depending on the available type of information about the input query and database. At one extreme, if both Q and the database integrity constraints are available, we can compute semantically equivalent queries. At the other extreme, if only $Q(D)$ and the database D are available, we can only compute weak IEQs. Finally, if both Q and the database D are available, we can compute both weak and strong IEQs.

Precise & approximate IEQs. It is also useful to permit some perturbation so as to include IEQs that are "close enough" to the original. Perturbations could be in the form of extra records or missing records or a combination thereof. Such generalizations are necessary in situations where there are no precise IEQs and useful for cases where the computational cost for finding precise IEQs is considered unacceptably high. Moreover, a precise IEQ Q' might not always provide insightful characterizations of $Q(D)$ as Q' could be too "detailed" with many join relations and/or selection predicates.

The imprecision of a weak IEQ Q' of Q (w.r.t. D) can be quantified by $|Q(D) - Q'(D)| + |Q'(D) - Q(D)|$; the imprecision of a strong IEQ can be quantified similarly. Thus, Q' is considered an approximate (strong/weak) IEQ of Q if its imprecision is positive; otherwise, Q' is a precise (strong/weak) IEQ.

As the search space for IEQs can be very large, particularly with large complex database schema where each rela-

tion has foreign-key joins with other relations, users should be able to restrict the search space by specifying hints/ preferences in the form of control parameters. Some examples include: (1) restricting Q' to be conjunctive queries, (2) setting an upper bound on the number of selection predicates in Q', (3) setting an upper bound on the number of relations in Q', (4) specifying a specific set of relations to be included (excluded) in (from) Q', and (5) specifying a specific set of attributes to be included (excluded) in (from) the selection predicates in Q'. In addition to these query-specific controls, some method-specific controls can also be applied on the IEQs search space; we discuss some of these in Section 4. We note although all the above user hints can be easily incorporated into our proposed algorithms, we do not delve on these control knobs any further in the paper but instead focus on the core problem of computing IEQs.

2.2 TALOS: Conceptual Approach

In this section, we give a conceptual overview of our approach, named `TALOS` (for **T**ree-based classifier with **A**t **L**east **O**ne **S**emantics), for the `QBO` problem.

Given an input result $Q(D)$, to generate a SPJ Q' that is an IEQ of Q, we need to basically determine the three components of Q': $rel(Q')$, $sel(Q')$, and $proj(Q')$. Clearly, if $rel(Q')$ contains a set of core relations of Q, then $proj(Q')$ can be trivially derived from these core relations[2]. Thus, the possibilities for Q' depends mainly on the options for both $rel(Q')$ and $sel(Q')$. Between these two components, enumerating different $rel(Q')$ is the easier task as $rel(Q')$ can be obtained by choosing a subgraph G of the schema graph $\mathcal{SG}$ such that G contains a set of core relations of Q: $rel(Q')$ is then given by all the relations represented in G. Note that it is not necessary for $rel(Q) \subseteq rel(Q')$ as Q may contain some relations that are not core relations. The reason for exploring different possibilities for $rel(Q')$ is to find interesting alternative characterizations of $Q(D)$ that involve different join paths or selection conditions from those in Q. `TALOS` enumerates different schema subgraphs by starting out with minimal subgraphs that contain a set of core relations of Q and then incrementally expanding the minimal subgraphs to generate larger, more complex subgraphs.

We now come to most critical and challenging part of our solution which is how to generate "good" $sel(Q')$'s such that each $sel(Q')$ is not only succinct (without too many conditions) and insightful but also minimizes the imprecision between $Q(D)$ and $Q'(D)$ if Q' is an approximate IEQ. We propose to formulate this problem as a *data classification* task as follows.

Consider the relation J that is computed by joining all the relations in $rel(Q')$ based on the foreign-key joins represented in G. Without loss of generality, let us suppose that we are looking for weak IEQs Q'. Let L denote the ordered listing of the attributes in $proj(Q')$ such that that the schema of $\pi_L(J)$ and $Q(D)$ are equivalent[3]. J can be partitioned into two disjoint subsets, $J = J_0 \cup J_1$, such that $\pi_L(J_1) \subseteq Q(D)$ and $\pi_L(J_0) \cap Q(D) = \emptyset$. For the purpose of deriving $sel(Q')$, one simple approach to classify the tuples in J is to label the tuples in J_0, which do not contribute to the query's result $Q(D)$, as *negative tuples*, and label the tuples in J_1 as *positive tuples*.

Given the labeled tuples in J, the problem of finding a $sel(Q')$ can now be viewed as a data classification task to separate the positive and negative tuples in J: $sel(Q')$ is given by the selection conditions that specify the positive tuples. A natural solution is to examine if off-the-shelf data classifier can give us what we need. To determine what kind of classifier to use, we must consider what we need to generate our desired IEQ Q'. Clearly, the classifier should be efficient to construct and the output should be easy to interpret and express using SQL; i.e., the output should be expressible in axis parallel cuts of the data space. These criteria rule out a number of classifier systems such as neural networks, k-nearest neighbor classification, Bayesian classifiers, and support vector machines [16]. Rule based classifiers or decision trees (a form of rule-based classifier) are a natural solution in this context. `TALOS` uses decision tree classifier for generating $sel(Q')$.

We now briefly describe how a simple binary decision tree is constructed to classify a set of data records D. For expository simplicity, assume that all the attributes in D have numerical domains. A decision tree DT is constructed in a top-down manner. Each leaf node N in the tree is associated with a subset of the data records, denoted by D_N, such that D is partitioned among all the leaf nodes. Initially, DT has only a single leaf node (i.e., its root node) which is associated with all the records in D. Leaf nodes are classified into pure and non-pure nodes depending on a given goodness criterion. Common goodness criteria include entropy, classification error and the Gini index [16]. At each iteration of the algorithm, the algorithm examines each non-pure leaf node N and computes the best split for N that creates two child nodes, N_1 and N_2, for N. Each split is computed as a function of an attribute A and a split value v associated with the attribute. Whenever a node N is split (w.r.t. attribute A and split value v), the records in D_N are partitioned between D_{N_1} and D_{N_2} such that a tuple $t \in D_N$ is distributed into D_{N_1} if $t.A \leq v$; and D_{N_2}, otherwise.

A popular goodness criterion for splitting, the Gini index, is computed as follows. For a data set S with k distinct classes, its Gini index is $Gini(S) = 1 - \sum_{j=1}^{k}(f_j^2)$ where f_j denote the fraction of records in S belonging to class j. Thus, if S is split into two subsets S_1, S_2, then the Gini index of the split is given by

$$Gini(S_1, S_2) = \frac{|S_1|\ Gini(S_1) + |S_2|\ Gini(S_2)}{|S_1| + |S_2|},$$

where $|S_i|$ denote the number of records in S_i. The general objective is to pick the splitting attribute whose best splitting value reduces the Gini index the most (the goal is to reduce Gini to 0 resulting in all pure leaf nodes).

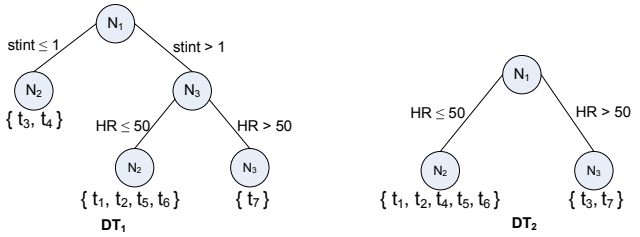
EXAMPLE 2. *To illustrate how decision tree classifier can be applied to derive IEQs, consider the following query on the baseball database D:* $Q_4 = \pi_{name}\ (\sigma_{bats=\text{“}R\text{”}\wedge throws=\text{“}R\text{”}}\ Master)$. *Note that* $Q_4(D) = \{B, E\}$. *Suppose that the schema subgraph G considered contains both* `Master` *and* `Batting`*; i.e.,* $rel(Q'_4) = \{Master, Batting\}$. *The output of* $J = Master \bowtie_{pID} Batting$ *is shown in Figure 2(a). Using* t_i *to denote the* i^{th} *tuple in J, J is partitioned into*

[2]Note that even though the definition of a weak IEQ Q' of Q does not require the queries to share a set of core relations, we find this restriction to be a reasonable and effective way to obtain "good" IEQs.

[3]If the search is for strong IEQs, then the discussion remains the same except that L is the ordered listing of the key attributes of a set of core relations S of Q, and we replace $Q(D)$ by $Q_S(D)$.

pID	name	country	weight	bats	throws	year	team	stint	HR
P1	A	USA	85	L	R	2001	PIT	2	40
P1	A	USA	85	L	R	2003	ML1	2	50
P2	B	USA	72	R	R	2001	PIT	1	73
P2	B	USA	72	R	R	2002	PIT	1	40
P3	C	USA	80	R	L	2004	CHA	2	35
P4	D	Germany	72	L	R	2001	PIT	3	30
P5	E	Japan	72	R	R	2004	CHA	3	60

(a) $J = \texttt{Master} \bowtie_{pID} \texttt{Batting}$



(b) Decision trees DT_1 and DT_2

Figure 2: Example of deriving IEQs for $Q_4 = \pi_{name}\ (\sigma_{bats=\text{“R”} \wedge throws=\text{“R”}}\ Master)$ on D

$J_0 = \{t_1, t_2, t_5, t_6\}$ and $J_1 = \{t_3, t_4, t_7\}$. Figure 2(b) shows two example decision trees, DT_1 and DT_2, constructed from J. Each decision tree partitions the tuples in J into different subsets (represented by the leaf nodes) by applying different sequences of attribute selection conditions. By labeling all tuples in J_1 as positive, the IEQ derived from DT_1 is given by $Q'_4 = \pi_{name}(\sigma_{stint\leq 1 \vee (stint>1 \wedge HR>50)}\ (Master \bowtie Batting))$. More details are described in Section 4.1. □

2.3 TALOS: Challenges

There are two key challenges in adapting decision tree classifier for the QBO problem.

At Least One Semantics. The first challenge concerns the issue of how to assign class labels in a flexible manner without over constraining the classification problem and limiting its effectiveness. Contrary to the impression given by the above simple class labeling scheme, the task of assigning class labels to J is actually a rather intricate problem due to the fact that multiple tuples in J_1 can be projected to the same tuple in $\pi_L(J_1)$. Recall that in the simple class labeling scheme described, a tuple t is labeled positive iff $t \in J_1$. However, note that it is possible to label only a subset of tuples $J'_1 \subseteq J_1$ as positive (with tuples in $J - J'_1$ labeled as negative) and yet achieve $\pi_L(J'_1) = \pi_L(J_1)$ (without affecting the imprecision of Q'). In other words, the simple scheme of labeling all tuples in J_1 as positive is just one (extreme) option out of many other possibilities.

We now discuss more precisely the various possibilities of labeling positive tuples in J to derive different $sel(Q')$. Let $\pi_L(J_1) = \{t_1, \cdots, t_k\}$. Then J_1 can be partitioned into k subsets, $J_1 = P_1 \cup \cdots \cup P_k$, where each $P_i = \{t \in J_1 \mid \text{the projection of t on L is } t_i\}$. Thus, each P_i represents the subset of tuples in J_1 that project to the same tuple in $\pi_L(J_1)$. Define J'_1 to be a subset of tuples of J_1 such that it consists of at least one tuple from each subset P_i. Clearly, $\pi_L(J'_1) = \pi_L(J_1)$, and there is a total of $\prod_{i=1}^{k}(2^{|P_i|} - 1)$ possibilities for J'_1. For a given J'_1, we can derive $sel(Q')$ using a data classifier based on labeling the tuples in J'_1 as positive and the remaining tuples in $J_1 - J'_1$ as negative.

Based on the above discussion on labeling tuples, each tuple in J can be classified as either a *bound tuple* or *free tuple* depending on whether there is any freedom to label the tuple. A tuple $t \in J$ is a *bound tuple* if either (1) $t \in J_0$ in which case t must be labeled negative, or (2) t is the only tuple in some subset P_i, in which case t must certainly be included in J'_1 and be labeled positive; otherwise, t is a *free tuple* (i.e., t is in some subset P_i that contains more than one tuple).

In contrast to conventional classification problem where each record in the input data comes with a well defined class label, the classification problem formulated for QBO has the unique characteristic where there is some flexibility in the class label assignment. We refer to this property as *at-least-one semantics*. To the best of our knowledge, we are not aware of any work that has addressed this variant of the classification problem.

An obvious approach to solve the at-least-one semantics variant is to map the problem into the traditional variant by first applying some arbitrary class label assignment that is consistent with the at-least-one semantics. In our experimental study, we compare against two such static labeling schemes, namely, NI, which labels all free tuples as positive, and RD, which labels a random non-empty subset of free tuples in each P_i as positive[4]. However, such static labeling schemes do not exploit the flexible class labeling opportunities to optimize the classification task. To avoid the limitations of the static scheme, TALOS employs a novel *dynamic class labeling scheme* to compute optimal node splits for decision tree construction without having to enumerate an exponential number of combinations of class labeling schemes for the free tuples.

EXAMPLE 3. *Continuing with Example 2, J_1 is partitioned into two subsets: $P_1 = \{t_3, t_4\}$ and $P_2 = \{t_7\}$, where P_1 and P_2 contribute to the outputs “B” and “E”, respectively. The tuples in J_0 and P_2 are bound tuples, while the tuples in P_1 are free tuples. To derive an IEQ, at least one of the free tuples in P_1 must be labeled positive. If t_3 is labeled positive and t_4 is labeled negative, DT_2 in Figure 2(b) is a simpler*

[4]We also experimented with a scheme that randomly labels only one free tuple for each subset as positive, but the results are worse than NI and RD.

decision tree constructed by partitioning J based on a selection predicate on attribute HR. The IEQ derived from DT_2 is $Q_4'' = \pi_{name}\ \sigma_{HR>50}\ (Master \bowtie Batting)$. □

Performance Issues. The second challenge concerns the performance issue of how to efficiently generate candidates for $rel(Q')$ and optimize the computation of the single input table J required for the classification task. To improve performance, TALOS exploits join indices to avoid a costly explicit computation of J and constructs mapping tables to optimize decision tree construction.

3. HANDLING AT-LEAST-ONE SEMANTICS

In this section, we address the first challenge of TALOS and present a novel approach for classifying data with the at-least-one semantics.

3.1 Computing Optimal Node Splits

The main challenge for classification with the at-least-one semantics is how to optimize the node splits given the presence of free tuples which offer flexibility in the class label assignment. We present a novel approach that computes the optimal node split *without* having to explicitly enumerate all possible class label assignments to the free tuples. The idea is based on exploiting the flexibility offered by the at-least-one semantics.

Let us consider an initial set of tuples S that has been split into two subsets, S_1 and S_2, based on a value v of a numeric attribute A (the same principle applies to categorical attributes as well); i.e., a tuple $t \in S$ belongs to S_1 iff $t.A \leq v$. The key question is how to compute the optimal Gini index of this split without having to enumerate all possible class label assignments for the free tuples in S such that the at-least-one semantics is satisfied. Without loss of generality, suppose that the set of free tuples in S is partitioned (as described in Section 2.3) into m subsets, $P_1, \cdots, P_m$, where each $|P_i| > 1$.

Let $n_{i,j}$ denote the number of tuples in $P_i \cap S_j$, and f_j denote the number of free tuples in S_j to be labeled positive to minimize $Gini(S_1, S_2)$, where $i \in [1, m], j \in \{1, 2\}$. We classify P_i, $i \in [1, m]$, as a SP_1*-set* (resp. SP_2*-set*) if P_i is completely contained in S_1 (resp. S_2); otherwise, P_i is a SP_{12}*-set* (i.e., $n_{i,1} > 0$ and $n_{i,2} > 0$).

To satisfy the at-least-one semantics, we need to ensure that at least one free tuple in each P_i, $i \in [1, m]$, is labeled positive. Let T_j, $j \in \{1, 2\}$, denote the minimum number of free tuples in S_j that must be labeled positive to ensure this. Observe that for a specific P_i, $i \in [1, m]$, if P_i a SP_1-set (resp. SP_2-set), then we must have $T_1 \geq 1$ (resp. $T_2 \geq 1$). Thus, T_j is equal to the number of SP_j-sets. More precisely, $T_j = \sum_{i=1}^{m} \max\{0, 1 - n_{i,3-j}\}$, $j \in \{1, 2\}$.

Thus, f_1 and f_2 must satisfy the following two conditions:

(A1) $T_j \leq f_j \leq \sum_{i=1}^{m} n_{i,j}$, $j \in \{1, 2\}$; and

(A2) $f_1 + f_2 \geq m$.

Condition (A1) specifies the possible number of free tuples to be labeled positive for each S_j, while condition (A2) specifies the minimum combined number of tuples in S to be labeled positive in order that the at-least-one semantics is satisfied for each P_i.

Based on conditions (A1) and (A2), it can be shown that the optimal value of $Gini(S_1, S_2)$ can be determined by considering only five combinations of f_1 and f_2 values as indicated by the second and third columns in Table 1. The proof of this result is given elsewhere [20]. These five cases correspond to different combinations of whether the number of positive or negative tuples is being maximized in each of S_1 and S_2: case C1 maximizes the number of positive tuples in both S_1 and S_2; case C2 maximizes the number of positive tuples in S_1 and maximizes the number of negative tuples in S_2; case C3 maximizes the number of negative tuples in S_1 and maximizes the number of positive tuples in S_2; and cases C4 and C5 maximize the number of negative tuples in both S_1 and S_2. The optimal value of $Gini(S_1, S_2)$ is given by the minimum of the Gini index values derived from the above five cases.

3.2 Updating Labels & Propagating Constraints

Once the optimal $Gini(S_1, S_2)$ index is determined for a given node split, we need to update the split of S by converting the free tuples in S_1 and S_2 to bound tuples with either positive/negative class labels. The details of this updating depends on which of the five cases the optimal Gini value was derived from, and is summarized by the last four columns in Table 1.

For case C1, which is the simplest case, all the free tuples in S_1 and S_2 will be converted to positive tuples. However, for the remaining cases, which involve maximizing the number of negative tuples in S_1 or S_2, some of the free tuples may not be converted to bound tuples. Instead, the maximization of negative tuples in S_1 or S_2 is achieved by propagating another type of constraints, referred to as "exactly-one" constraints, to some subsets of tuples in S_1 or S_2. Similar to the principle of at-least-one constraints, the idea here is to make use of constraints to optimize the Gini index values for subsequent node splits without having to explicitly enumerate all possible class label assignments. Thus, in Table 1, the fourth and fifth columns specify which free tuples are to be converted to bound tuples with positive and negative labels, respectively; where an '-' entry means that no free tuples are to be converted to bound tuples. The sixth and seventh columns specify what subsets of tuples in S_1 and S_2, respectively, are required to satisfy the exactly-one constraint; where an '-' entry column means that no constraints are propagated to S_1 or S_2.

We now define the exactly-one constraint and explain why it is necessary. An *exactly-one constraint* on a set of free tuples S' requires that exactly one free tuple in S' must become labeled as positive with the remaining free tuples in S' labeled as negative. Consider case C2, which is to maximize the number of positive (resp. negative) tuples in S_1 (resp. S_2). The maximization of the number of positive tuples in S_1 is easy to achieve since by converting all the free tuples in S_1 to positive, the at-least-one constraints on the SP_1-sets and SP_{12}-sets are also satisfied. Consequently, for each SP_{12}-set P_i, all the free tuples in $P_i \cap S_2$ can be converted to negative tuples (to maximize the number of negative tuples in S_2) without violating the at-least-one constraint on P_i. However, for a SP_2-set P_i, to maximize the number of negative tuples in P_i while satisfying the at-least-one semantics translates to an exactly-one constraint on P_i. Thus, for case C2, an exactly-one constraint is propagated to each SP_2-set in S_2, and no constraints are propagated to S_1. A

Case	Number of free tuples to be labeled positive		Labeling of free tuples		Exactly-One Constraint Propagation	
	f_1	f_2	positive	negative	S_1	S_2
$C1$	$\sum_{i=1}^{m} n_{i,1}$	$\sum_{i=1}^{m} n_{i,2}$	$S_1 \cup S_2$	-	-	-
$C2$	$\sum_{i=1}^{m} n_{i,1}$	T_2	S_1	SP_{12}-sets in S_2	-	SP_2-sets
$C3$	T_1	$\sum_{i=1}^{m} n_{i,2}$	S_2	SP_{12}-sets in S_1	SP_1-sets	-
$C4$	T_1	$m - T_1$	-	SP_{12}-sets in S_1	SP_1-sets	All subsets
$C5$	$m - T_2$	T_2	-	SP_{12}-sets in S_2	All subsets	SP_2-sets

Table 1: Optimizing Node Splits

similar reasoning applies to cases C3 to C5. Thus, while the at-least-one constraint is applied to each subset of free tuples P_i in the initial node split, the exactly-one constraint is applied to each P_i for subsequent node splits. This second variant of the node split problem can be optimized by techniques similar to what we have explained so far for the first variant. In particular, the first condition (A1) for f_1 and f_2 remains unchanged, but the second condition (A2) becomes $f_1 + f_2 = m$. Consequently, the optimization of the Gini index value becomes simpler and only needs to consider cases C4 and C5.

EXAMPLE 4. *To illustrate how class labels are updated and how constraints are propagated during a node split, consider the following query on the baseball database D:* $Q_5 = \pi_{stint}\ (\sigma_{country="USA"}\ Master \bowtie_{pID} Batting)$. *Suppose that the weak-IEQ* Q'_5 *being considered has* $rel(Q'_5) = \{Master, Batting\}$. *Let* $J = Master \bowtie_{pID} Batting$ *(shown in Figure 2(a)). Since* $Q_5(D) = \{1, 2\}$, *we have* $J_0 = \{t_6, t_7\}$, $P_1 = \{t_1, t_2, t_5\}$ *(corresponding to stint = 2) and* $P_2 = \{t_3, t_4\}$ *(corresponding to stint = 1). The tuples in* J_0 *are labeled negative, while the tuples in* P_1 *and* P_2 *are all free tuples.*

Suppose that the splitting attribute considered is "weight", and the optimal splitting value for "weight" is 72. *The* $Gini(S_1, S_2)$ *values computed (w.r.t. "weight = 72") for the five cases, C1 to C5, are* 0.29, 0.48, 0.21, 0.4 *and* 0.4, *respectively. Thus, the optimal value of* $Gini(S_1, S_2)$ *is* 0.21 *(due to case C3). We then split tuples with weight* ≤ 72 *(i.e.,* $\{t_3, t_4, t_6, t_7\}$*) into* S_1 *and tuples with weight* > 72 *(i.e.,* $\{t_1, t_2, t_5\}$*) into* S_2. *Thus,* P_1 *is a* SP_2*-set while* P_2 *is a* SP_1*-set. Since the optimal Gini index computed is due to case C3 (i.e., maximizing negative tuples in* S_1 *and maximizing positive tuples in* S_2*), all the free tuples in* S_2 *(i.e.,* t_1, t_2 *and* t_5*) are labeled positive, and an exactly-one constraint is propagated to the set of tuples* $P_2 \cap S_1$ *(i.e.,* $\{t_3, t_4\}$*).* □

In summary, TALOS is able to efficiently compute the optimal Gini index value for each attribute split value considered without enumerating an exponential number of class label assignments for the free tuples.

4. OPTIMIZING PERFORMANCE

In this section, we first explain how TALOS adapts a well-known decision tree classifier for performing data classification in the presence of free tuples where their class labels are not fixed. We then explain the performance challenges of deriving Q' when $rel(Q')$ involves multiple relations and present optimization techniques to address these issues. For ease of presentation and without loss of generality, the discussion here assumes weak IEQs.

4.1 Classifying Data in TALOS

We first give an overview of SLIQ [13], a well-known decision tree classifier, that we have chosen to adapt for TALOS. We then describe the extensions required by TALOS to handle data classfication in the presence of free tuples. Finally, we present a non-optimized, naive variant of TALOS. It is important to emphasize that our approach is orthogonal to the choice of the decision tree technique.

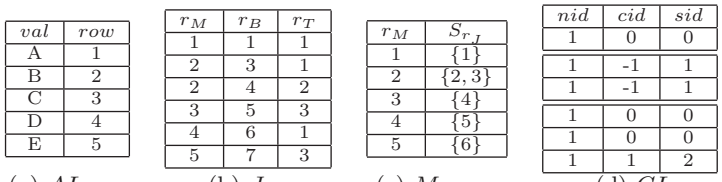
Overview of SLIQ. To optimize the decision tree construction on a set of data records D, SLIQ uses two key data structures. First, a sorted attribute list, denoted by AL_i, is pre-computed for each attribute A_i in D. Each AL_i can be thought of as a two-column table (val, row), of the same cardinality as D, that is sorted in non-descending order of val. Each record $r = (v, i)$ in AL_i corresponds to the i^{th} tuple t in D, and $v = t.A_i$. The sorted attribute lists are used to speed up the computation of optimal node splits. To determine the optimal node split w.r.t. A_i requires a single sequential scan of AL_i.

Second, a main-memory array called *class list*, denoted by CL, is maintained for D. This is a two-column table (nid, cid) with one record per tuple in D. The i^{th} entry in CL, denoted by $CL[i]$, corresponds to the i^{th} tuple t in D, where $CL[i].nid$ is the identifier of leaf node N, $t \in D_N$, and $CL[i].cid$ refers to the class label of t. CL is used to keep track of the tuples location (i.e., in which leaf nodes) as leaf nodes are split.

Class List Extension. In order to support data classification with free tuples, where their class labels are assigned dynamically, we need to extend SLIQ with the following modification. The class list table $CL(nid, cid, sid)$ is extended with an additional column "sid", which represents a subset identifier, to indicate which subset (i.e., P_i) a tuple belongs to. This additional information is needed to determine the optimal Gini index values as discussed in the previous section. Consider a tuple t which is the i^{th} tuple in D. The cid and sid values in CL are maintained as follows: if t belongs to J_0, then $CL[i].cid = 0$ and $CL[i].sid = 0$; if t is a bound tuple in P_j, then $CL[i].cid = 1$ and $CL[i].sid = j$; otherwise, if t is a free tuple in P_j, then $CL[i].cid = -1$ and $CL[i].sid = j$.

EXAMPLE 5. *Figure 3 shows some data structures created for computing IEQs for* $Q_4(D)$. *Figure 3(a) shows the attribute list created for attribute* Master.*name; and Figure 3(d) shows the initial class list created for* J_{hub}, *where all the records are in a single leaf node (with nid value of 1).* □

Naive TALOS. Before presenting the optimizations for TA-



val	row
A	1
B	2
C	3
D	4
E	5

(a) AL_{name}

r_M	r_B	r_T
1	1	1
2	3	1
2	4	2
3	5	3
4	6	1
5	7	3

(b) J_{hub}

r_M	S_{r_J}
1	{1}
2	{2, 3}
3	{4}
4	{5}
5	{6}

(c) M_{Master}

nid	cid	sid
1	0	0
1	-1	1
1	-1	1
1	0	0
1	0	0
1	1	2

(d) CL

Figure 3: Example data structures for $Q_4(D)$

LOS in the next section, let us first describe a non-optimized, naive variant of TALOS (denoted by TALOS-). Suppose that we are considering an IEQ Q' where $rel(Q') = \{R_1, \cdots, R_n\}$, $n > 1$, that is derived from some schema subgraph G. First, TALOS- joins all the relations in $rel(Q')$ (based on the foreign-key joins represented in G) to obtain a single relation J. Next, TALOS- computes attribute lists for the attributes in J and a class list for J. TALOS- is now ready to construct a decision tree DT to derive the IEQ Q' with these structures. DT is initialized with a single leaf node consisting of the records in J, which is then refined iteratively by splitting the leaf nodes in DT. TALOS- terminates the splitting of a leaf node when (1) its tuples are either all labeled positive or all labeled negative; or (2) its tuples have the same attribute values w.r.t. all the splitting attributes. Finally, TALOS- classifies each leaf node in DT as positive or negative as follows: a leaf node is classified as positive if and only if (1) all its tuples are labeled positive, or (2) the ratio of the number of its positive tuples to the number of its negative tuples is no smaller than a threshold value given by τ. In our experiments, we set $\tau = 1$. $sel(Q')$ is then derived from the collection of positive leaf nodes in DT as follows. Each set of tuples in a positive leaf node is specified by a selection predicate that is a conjunction of the predicates along the path from the root node to that leaf node, and the set of tuples in a collection of positive leaf nodes is specified by a selection predicate that is a disjunction of the selection predicate for each selected leaf node. In the event that all the leaf nodes in DT are classified as negative, the computation of Q' is not successful (i.e., there is no IEQ for $rel(Q')$) and we refer to Q' as a *pruned IEQ*.

4.2 Optimizations

The naive TALOS described in the previous section suffers from two drawbacks. First, the overhead of computing J can be high especially if there are many large relations in $rel(Q')$. Second, since the cardinality of J can be much larger than the cardinality of each of the relations in $rel(Q')$, building decision trees directly using J entails the computation and scanning of correspondingly large attribute lists which further increases the computation cost. In the rest of this section, we present the optimization techniques used by TALOS to address the above performance issues.

Join Indices & Hub Table. To avoid the overhead of computing J from $rel(Q')$, TALOS exploits pre-computed join indices [21] which is a well-known technique for optimizing joins. For each pair of relations, R and R', in the database schema that are related by a foreign-key join, its join index, denoted by $I_{R,R'}$, is a set of pairs of row identifiers referring to a record in each of R and R' that are related by the foreign-key join.

Based on the foreign-key join relationships represented in the schema subgraph G, TALOS computes the join of all the appropriate join indices for $rel(Q')$ to derive a relation, called the *hub table*, denoted by J_{hub}. Computing J_{hub} is much more efficient than computing J since there are fewer number of join operations (i.e., number of relevant join indices) and each join attribute is a single integer-valued column.

EXAMPLE 6. *Consider again query Q_4 introduced in Example 2. Suppose that we are computing IEQ Q'_4 with $rel(Q'_4)$ = {*Master*, *Batting*, *Team*}. Figure 3(b) shows the hub table, J_{hub}, produced by joining two join indices: one for* Master $\bowtie_{pID}$ Batting *and the other for* Batting $\bowtie_{team,year}$ Team*. Here, r_M, r_B, and r_T refer to the row identifiers for Master, Batting, and Team relations, respectively.* □

Mapping Tables. Instead of computing and operating on large attribute lists (each with cardinality equal to $|J|$) as in the naive approach, TALOS operates over the smaller pre-computed attribute lists AL_i for the base relations in $rel(Q')$ together with small *mapping tables* to link the pre-computed attribute lists to the hub table. In this way, TALOS only needs to pre-compute once the attribute lists for all the base relations, thereby avoiding the overhead of computing many large attribute lists for different $rel(Q')$ considered.

Each mapping table, denoted by M_i, is created for each $R_i \in rel(Q')$ that links each record r in R_i to the set of records in J_{hub} that are related to r. Specifically, for each record r in R_i, there is one record in M_i of the form (j, S), where j is the row identifier of r, and S is a set of row identifiers representing the set of records in J_{hub} that are created from r.

EXAMPLE 7. *Figure 3(c) shows the mapping table M_{Master} that links the* Master *relation in Figure 1 and J_{hub} in Figure 3(b). The record $(2, \{2, 3\})$ in M_{Master} indicates that the second tuple in* Master *relation (with pID of P2), contributed two tuples, located in the second and third rows, in J_{hub}.* □

Computing Class List. We now explain how TALOS can efficiently compute the class list CL for J (without having explicitly computed J) by using the attribute lists, hub table, and mapping tables. The key task in computing CL is to partition the records in J into subsets (J_0, P_1, P_2, etc.), as described in the previous section. For simplicity and without loss of generality, assume that the schema of $Q(D)$ has n attributes $A_1, \cdots, A_n$, where each A_i is an attribute of relation R_i. TALOS first initializes CL with one entry for each record in J_{hub} with the following default values: $nid = 1$, $cid = 0$, and $sid = 0$. For each record r_k that is accessed

by a sequential scan of $Q(D)$, TALOS examines the value v_i of each attribute A_i of r_k. For each v_i, TALOS first retrieves the set of row identifiers RI_{v_i} of records in R_i that have a value of v_i for attribute $R_i.A_i$ by performing a binary search on the attribute list for $R_i.A_i$. With this set of row identifiers RI_{v_i}, TALOS probes the mapping table M_i to retrieve the set of row identifiers JI_{v_i} of the records in J_{hub} that are related to the records referenced by RI_{v_i}. The intersection of the JI_{v_i}'s for all the attribute values of r_k, denoted by P_k, represents the set of records in J that can generate r_k. TALOS updates the entries in CL corresponding to the row identifiers in P_k as follows: (1) the *sid* value of each entry is set to k (i.e., all the entries belong to the same subset corresponding to record r_k); and (2) the *cid* value of each entry is set to 1 (i.e., tuple is labeled positive) if $|P_k| = 1$; otherwise, it is set to -1 (i.e., it is a free tuple).

EXAMPLE 8. *We illustrate how* TALOS *creates CL for query Q_4, which is shown in Figure 3(d). Initially, each row in CL is initialized with $sid = 0$ and $cid = 0$.* TALOS *then access each record of $Q_4(D)$ sequentially. For the first record (with name = "B'),* TALOS *searches AL_{name} and obtains $RI_B = \{2\}$. It then probes M_{Master} with the row identifer in RI_B and obtains $JI_B = \{2, 3\}$. Since $Q_4(D)$ contains only one attribute, we have $P_1 = \{2, 3\}$. The second and the third rows in CL are then updated with $sid = 1$ and $cid = -1$. Similarly, for the second record in $Q_4(D)$ (with name = "E"),* TALOS *searches AL_{name} and obtains $RI_E = \{5\}$, and derives $JI_E = \{6\}$ and $P_2 = \{6\}$. The sixth row in CL is then updated with $sid = 2$ and $cid = 1$.* □

5. RANKING IEQS

In this section, we describe the ranking criteria we adopt to prioritize results presented to the user. Specifically, we consider a metric based on the Minimum Description Length (MDL) principle [18], and two metrics based on the F-measure [22].

Minimum Description Length (MDL). The MDL principle argues that all else being equal, the best model is the one that minimizes the sum of the cost of describing the data given the model and the cost of describing the model itself. If M is a model that encodes the data D, then the total cost of the encoding, $cost(M, D)$, is defined as: $cost(M, D) = cost(D|M) + cost(M)$, where $cost(M)$ is the cost to encode the model (i.e., the decision tree in our case) and $cost(D|M)$ is the cost to encode the data given the model. We can rely on succinct tree-based representations to compute $cost(M)$. The data encoding cost, $cost(D|M)$, is calculated as the sum of classification errors. The details of the encoding computations are given elsewhere [13].

F-measure. We now present two useful metrics based on the popular F-measure. The first variant follows the standard definition of F-measure: the F-measure for two IEQs Q and Q' is defined as $F_m = \frac{2\times|p_a|}{2\times|p_a|+|p_b|+|p_c|}$, where $p_a = Q(D) \cap Q'(D)$, $p_b = Q'(D) - Q(D)$, and $p_c = Q(D) - Q'(D)$. We denote this variant as *F-measure* in our experimental study.

Observe that the first variant is useful only for approximate IEQs, and is not able to distinguish among precise IEQs as this metric gives identical values for precise IEQs since p_b and p_c are empty. To rank precise IEQs, we introduce a second variant, denoted by F_m^{est}, which relies on estimating p_a, p_b, and p_c using existing data probabilistic models (as opposed to using the actual values from the data set). F_m^{est} captures how the equivalence of queries is affected by database updates, and the IEQ with high F_m^{est} is preferable to another IEQ with low F_m^{est}. For simplicity, we use a simple independent model [20] to estimate F_m^{est}; other techniques such as the Bayesian model by Getoor and others [8] can be applied too. The second variant has the benefit that estimates which are computed from a global distribution model may more accurately reflect the true relevance of the IEQs than one computed directly from the data. This of course pre-supposes that future updates follow the existing data distribution.

6. EXPERIMENTAL STUDY

In this section, we evaluate the performance of the proposed approaches for computing IEQs and study the relevance of the results returned. The algorithms being compared include our proposed TALOS approach, which is based on a dynamic assignment of class labels for free tuples, and two static class labeling techniques: NI labels all the free tuples as positive, and RD labels a random number of at least one free tuple in each subset as positive. We also examined the effectiveness of our proposed optimizations by comparing against a non-optimized naive variant of TALOS (denoted by TALOS-) described in Section 4.1.

The database system used for the experiments is MySQL Server 5.0.51; and all algorithms are coded using C++ and compiled and optimized with gcc. Our experiments are conducted on dual core 2.33GHz machine with 3.25GB RAM, running Linux. The experimental result timings reported are averaged over 5 runs with caching effects removed.

6.1 Data sets & Queries

We use three real data sets: one small size (Adult), one medium size (Baseball) and one large data set (TPC-H). All the test queries are given in Table 2, where sf refers to the selectivity factor of a query.

Adult. The Adult data set, from the UCI Machine Learning Repository[5], is a single-relation data set that has been used in many classification works. We use this data set to illustrate the utility of the IEQs for the simple case when both the input query Q as well as the output IEQ Q' involve only one relation. The four test queries for this data set are A_1, A_2, A_3 and A_4[6]. The first three queries have different selectivities: low (A_1), medium (A_2) and very high (A_3). Query A_4 is used to illustrate how TALOS handles skyline queries [4].

In addition, we also run three sets of workload queries with varying selectivities (low, medium, high) shown in Table 3. Each workload set W_i consists of five queries denoted by W_{i1} to W_{i5}. The average selectivity factor of the queries in W_1, W_2, and W_3 are, respectively, 0.85, 0.43, and 0.05.

Baseball. The baseball data set is a more complex, multi-relation database that contains Batting, Pitching, and Fielding statistics for Major League Baseball from 1871 through

[5] http://archive.ics.uci.edu/ml/datasets/Adult

[6] We use *gain*, *ms*, *edu*, *loss*, *nc*, *hpw*, and *rs*, respectively, as abbreviations for capital-gain, marital-status, education, capital-loss, native-country, hours-per-week, and relationship.

	Query	sf
A_1	π_{nc} ($\sigma_{occ=\text{“Armed-Force”}}$ $adult$)	0.91
A_2	π_{edu} ($\sigma_{ms=\text{“Married-AF”}\wedge race=\text{“Asian”}}$ $adult$)	0.17
A_3	$\pi_{edu,occ}$ ($\sigma_{ms=\text{“Never-married”}\wedge 64\leq age\leq 68}$ $\sigma_{race=\text{“White”}\wedge gain>500\wedge sex=\text{“F”}}$ $adult$)	0.06
A_4	π_{id} ($\sigma_{SKY\text{-}LINE(gain\ MAX, age\ MIN)}$ $adult$)	0.06
B_1	$\pi_{name}(\sigma_{team=\text{“ARI”}\wedge year=2006\wedge HR>10}$ $(Master \bowtie Batting))$	0.0004
B_2	$\pi_{name}(\sigma_{sum(HR)>600})$ $(Master \bowtie Batting)$	0.001
B_3	π_{name} ($\sigma_{SKY\text{-}LINE(HR\ MAX, SO\ MIN)}$ $(Master \bowtie Batting))$	0.002
B_4	$\pi_{name,year,rank}$ ($\sigma_{team=\text{“CIN”}\wedge 1982<year<1988}$ $(Manager \bowtie Team))$	0.0004
T_1	$\pi_{mfgr}(\sigma_{brand=\text{“Brand\#32”}}(part))$	0.2
T_2	$\pi_{name}(\sigma_{price>500{,}000\wedge priority=\text{“Urgent”}}$ $(customer \bowtie order))$	0.0004

Table 2: Test queries for experiments

	Query	sf
W_{11}	π_{ms} ($\sigma_{19\leq age\leq 22\wedge edu=\text{“Bachelors”}}$ $adult$)	0.79
W_{12}	π_{nc} ($\sigma_{occ=\text{“Armed-Force”}}$ $adult$)	0.91
W_{13}	$\pi_{occ,ms}$ ($\sigma_{nc=\text{“Phillipines”}\wedge 30\leq age\leq 40}$ $adult$)	0.83
W_{14}	$\pi_{edu,age}$ ($\sigma_{wc=\text{“Private”}\wedge race=\text{“Asian”}}$ $adult$)	0.82
W_{15}	$\pi_{occ,edu}$ ($\sigma_{gain>9999}$ $adult$)	0.89
W_{21}	π_{edu} ($\sigma_{23\leq age\leq 24\wedge nc=\text{“Germany”}}$ $adult$)	0.53
W_{22}	$\pi_{age,wc,edu}$ ($\sigma_{hpw\leq 19\wedge race=\text{“White”}}$ $adult$)	0.64
W_{23}	$\pi_{edu,age,ms}$ ($\sigma_{wc=\text{“Private”}\wedge race=\text{“Asian”}}$ $adult$)	0.61
W_{24}	$\pi_{edu,age}$ ($\sigma_{ms=\text{“Separated”}\wedge wc=\text{“State-gov”}}$ $\sigma_{race=\text{“White”}}$ $adult$)	0.2
W_{25}	π_{edu} ($\sigma_{ms=\text{“Married-AF”}\wedge race=\text{“Asian”}}$ $adult$)	0.17
W_{31}	π_{age} ($\sigma_{ms=\text{“Divorced”}\wedge wc=\text{“State”}\wedge age>70}$ $adult$)	0.002
W_{32}	$\pi_{occ,edu}$ ($\sigma_{ms=\text{“NM”}\wedge 64\leq age\leq 68\wedge race=\text{“White”}}$ $\sigma_{gain>2000\wedge sex=\text{“F”}}$ $adult$)	0.06
W_{33}	$\pi_{age,wc,edu}$ ($\sigma_{hpw\leq 19\wedge race=\text{“White”}\wedge nc=\text{“England”}}$ $adult$)	0.01
W_{34}	$\pi_{edu,age,gain}$ ($\sigma_{ms=\text{“Married-civ”}\wedge race\text{“Asian”}}$ $\sigma_{30\leq age\leq 37}$ $adult$)	0.17
W_{35}	$\pi_{edu,gain}$ ($\sigma_{gain>5000\wedge nc=\text{“Vietnam”}}$ $adult$)	0.0008

Table 3: Workload query sets (W_1, W_2, W_3) for Adult

2006 created by Sean Lahman. There are 16,639 records of baseball players, 88,686 records in *Batting*, 37,598 records in *Pitching*, 128,426 records *Fielding* and other auxiliary relations (AwardsPlayer, Allstar, Team, Managers, etc.). The queries used for this data set (B_1, B_2, B_3, B_4) are common queries that mainly relate to baseball players' performance.

TPC-H. To evaluate the scalability of our approach, we use the TPC-H data set (with a scaling factor of 1) and two test queries, T_1 and T_2.

6.2 Comparing TALOS, NI, and RD

In this section, we compare TALOS against the two static class-labeling schemes, NI and RD, in terms of their efficiency as well as the quality of the generated IEQs.

Figures 4(a) and (b) compare the performance of the three algorithms in terms of the number of weak IEQs generated and their running times, respectively, using the queries A_1 to A_4. Note that Figure 4 only compares the performance for weak IEQs because as the Adult data set is a single-relation database, all the tuples are necessarily bound when computing strong IEQs. Thus, the performance results for strong IEQs are the same for all algorithms and are therefore omitted. Similarly, the results for query A_4 are also omitted from the graphs because it happens that all the tuples are bound for query A_4; hence, the performance results are again the same for all three algorithms.

The results in Figures 4(a) and (b) clearly show that TALOS outperforms NI and RD in terms of both the total number of (precise and approximate) IEQs computed[7] as well as the running time. In particular, observe that the number of precise IEQs from TALOS is consistently larger than that from NI and RD. This is due to the flexibility of TALOS's dynamic assignment of class labels for free tuples which increase its opportunities to derive precise IEQs. In contrast, the static class label assignment schemes of NI and RD are too restrictive and are not effective for generating precise IEQs.

Query	Average height			Average size		
	NI	RD	TALOS	NI	RD	TALOS
A_1	14.9	19.8	2.1	5304	9360	4.7
A_2	13.4	20.1	3.2	4769	4966	6.9
A_3	16.1	21.8	6.5	3224	2970	19.2

Table 4: Comparison of decision trees for NI, RD, and TALOS

In addition, TALOS is also more efficient than NI and RD in terms of the running time. The reason for this is due to the flexibility of TALOS's dynamic labeling scheme for free tuples which results in decision trees that are smaller than those constructed by NI and RD. Table 4 compares the decision trees constructed by TALOS, NI, and RD in terms of their average height and average size (i.e., number of nodes). Observe that the decision trees constructed by TALOS are significantly more compact than that by NI and RD.

Figures 4(c) and (d) compare the quality of the IEQs generated by the three algorithms using the MDL and F-measure metrics, respectively. The results show that TALOS produces much better quality IEQs than both NI and RD: while the average value of the MDL metric for TALOS is extremely low (under 60), the corresponding values of both NI and RD are in the range of [4000, 22000]. For the F-measure metric, the average value for TALOS is nearly 1 (larger than 0.8), whereas the values for NI and RD are only around 0.3 and 0.4, respectively.

Figure 5 compares the three algorithms for the three sets of workload queries, W_1, W_2, and W_3, on the Adult data set. As the results in Figure 5(a) show, TALOS again outperforms both NI and RD in terms of running time. For both low and medium selectivity query workload (i.e., W_1 and W_2), the results in Figures 5(b) and (c) show TALOS is able to find many more precise IEQs for all queries compared to NI and RD. The reason for this is because such queries have a larger number of free tuples which gives TALOS more flexibility to derive precise IEQs. Figure 5(d) shows the comparison for the high selectivity query workload (i.e., W_3). As the number of free tuples is smaller for highly selective queries, the flexibility for TALOS becomes reduced, but TALOS still obtains about 1.5 to 9 times larger number of precise IEQs compared to NI and RD.

Our comparison results for the Baseball data set (not shown) also demonstrate similar trends with TALOS outperforming NI and RD in both the running time as well as the number and quality of IEQs generated.

[7]For clarity, we have also indicated in Figure 4(a) the number of pruned IEQs (defined in Section 4.1) computed by each algorithm. Since the number of decision trees considered by all three algorithms are the same, the sum of the number of precise, approximate, and pruned IEQs generated by all the algorithms are the same.

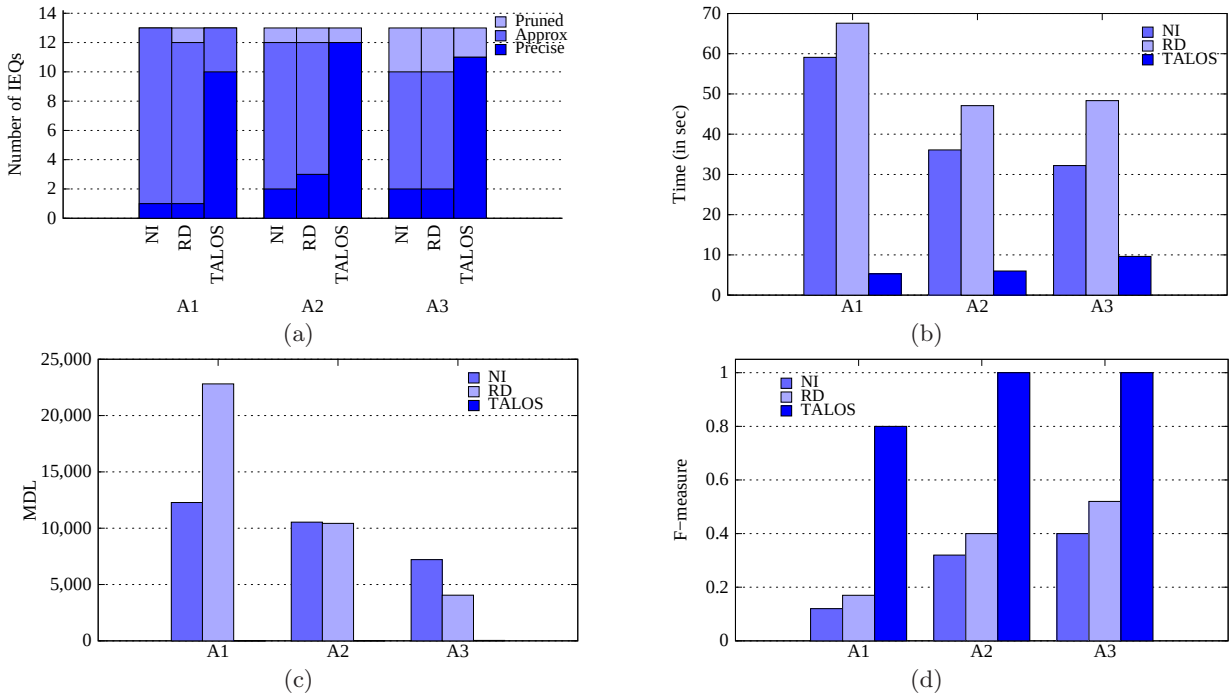


Figure 4: Comparison of TALOS, NI and RD. (a) Number of IEQs (b) Running time (c) MDL metric (d) F-measure metric

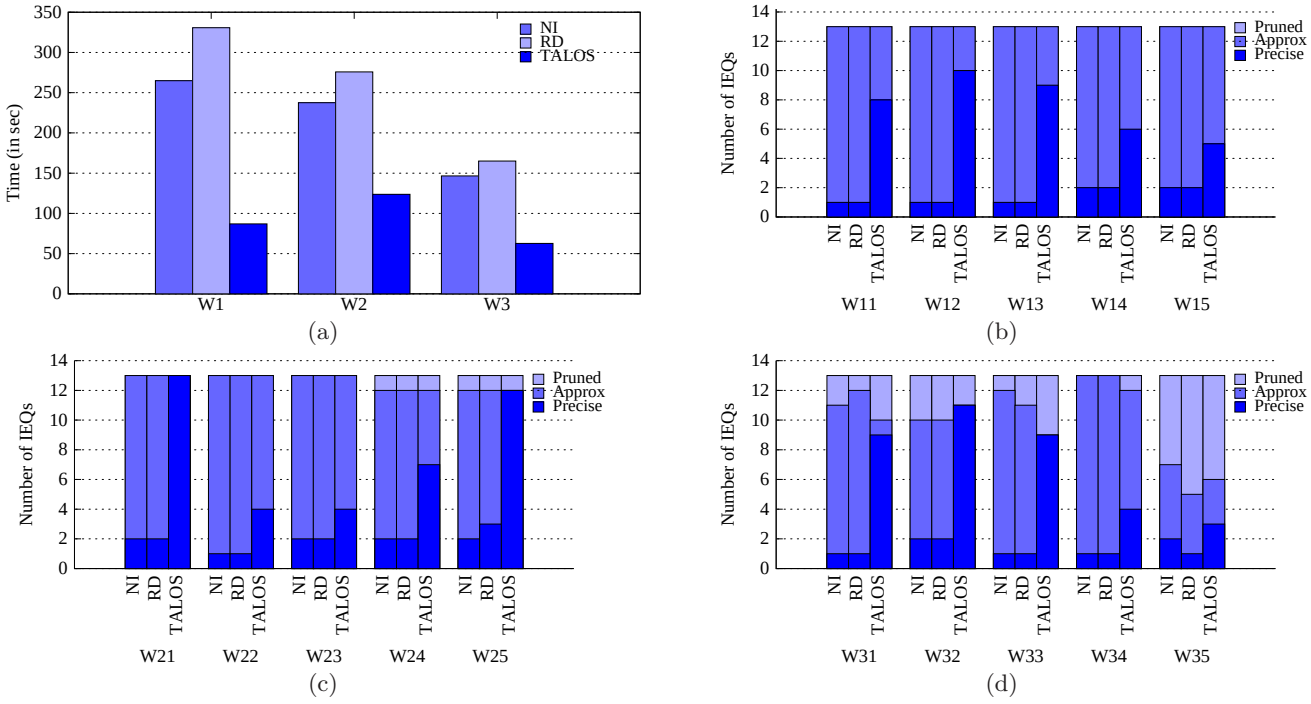


Figure 5: Comparison of TALOS, NI and RD for workload queries. (a) Running time (b) Number of IEQs for W_1 (c) Number of IEQs for W_2 (d) Number of IEQs for W_3

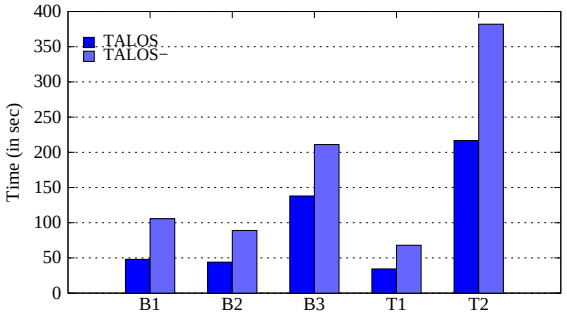


Figure 6: Comparison of TALOS and TALOS-

6.3 Effectiveness of Optimizations

Figure 6 examines the effectiveness of the optimizations by comparing the running times of `TALOS` and `TALOS-` on both the Baseball and TPC-H data sets. Note that the number and quality of IEQs produced by `TALOS` and `TALOS-` are the same as these qualities are independent of the optimizations. The results show that `TALOS` is about 2 times faster than `TALOS-`. The reason is that the attribute lists accessed by `TALOS`, which correspond to the base relations, are much smaller than the attribute lists accessed by `TALOS-`, which are based one J. For example, for query T_1, the attribute list constructed by `TALOS-` for attribute "container" in *part* relation is 4 times larger than that constructed by `TALOS`; and for query T_2, the attribute list constructed by `TALOS-` for attribute "acctbal" in *customer* relation is 10 times larger than that constructed by `TALOS`. In addition, the computation of J_{hub} by `TALOS` using join indices is also more efficient than the computation of J by `TALOS-`.

For the queries B_1, B_2, and B_3 on the Baseball data set, the number of IEQs (both precise and approximate) generated by `TALOS` is in the range $[50, 80]$ with an average running time of about 80 seconds. Thus, it takes `TALOS` about 1.2 seconds to generate one IEQ which is reasonable. Note that the comparison for query B_4 is omitted because while `TALOS` takes only 37.6 seconds to complete, the running time by `TALOS-` exceeds 15 minutes.

For the queries T_1 and T_2 on the TPC-H data set, `TALOS` takes 34.34 seconds to compute six precise IEQs for T_1, and takes 200 seconds to compute one precise and one approximate IEQs. T_2 is more costly to evaluate than T_1 because the decision trees constructed for T_2 are larger and more complex: the average height and size of the decision trees for T_2 are, respectively, 2 and 4 times, larger than those for T_1. Overall, even for the large TPC-H data set, the running time for `TALOS` is still reasonable.

6.4 Strong and Weak IEQs

In this section, we discuss some of the IEQs generated by `TALOS` for the various queries. The sample of weak and strong IEQs generated from Adult data set are shown in Tables 5 and 6, respectively. Tables 7 show sample weak IEQs generated from Baseball and TPC-H data sets, respectively[8]. For each IEQ, we also show its value for the F-measure or F_m^{est} metric; the latter is used only in Table 5 as all the IEQs shown in this table are precise. In Tables 6 and 7, the F-measure metric values are shown in terms of their $|p_a|$, $|p_b|$ and $|p_c|$ values; a IEQ is precise iff $|p_b| = 0$ and $|p_c| = 0$. We use $X_{i,j}$ to denote an IEQ for a query X_i, $X \in \{A, B, T\}$.

[8]The weak IEQs shown in Table 7 actually turn out to be strong IEQs as well for the queries B_1 to B_4.

Q	IEQ	F_m^{est}
$A_{1,1}$	$\sigma_{gain>7298 \wedge ms=\text{"Married-AF"}}$ $(adult)$	0.63
$A_{1,2}$	$\sigma_{edu=\text{"Preschool"} \wedge race=\text{"Eskimo"}}$ $(adult)$	0.25
$A_{1,3}$	$\sigma_{loss>3770}$ $(adult)$	0.24
$A_{2,1}$	$\sigma_{loss>3683 \wedge edu\_num>10}$ $(adult)$	0.07
$A_{2,2}$	$\sigma_{age\leq 24 \wedge nc=\text{"Hungary"}}$ $(adult)$	0.06
$A_{3,1}$	$\sigma_{p_1 \vee p_2}$ $(adult)$ $p_1 = (age \leq 85 \wedge hpw \leq 1 \wedge edu > 13)$ $p_2 = (age > 85 \wedge edu = \text{"Master"} \wedge hpw \leq 40)$	0.004

Table 5: Weak IEQs on Adult

Adult. In query A_1, we want to know the native country of people whose occupation is in the Armed Force. The query result is "U.S". From the weak IEQs, we learn that the people who is married to some one in the Armed Force and have high capital gain ($A_{1,1}$) have the same native country "U.S"; or people with high capital loss (> 3770) also have "U.S" as their native country ($A_{1,3}$).

In query A_2, we want to know the education level of Asians who have a spouse working in the Air Force. The result shows that they all have bachelor degrees. From the weak IEQs, we know that Hungarians who are younger than 25 also have the same education level ($A_{2,2}$). For the strong IEQs, we have some interesting characterizations: $A_{2,3}$ shows that these Asians are from Philippines, while $A_{2,4}$ shows that these Asians are wives whose age are at most 30 and work more than 52 hours per week. Such alternative queries provide more insights about query A_2 on the Adult data set.

In query A_3, we want to find the occupation and education of white females who are never married with age in the range $[64, 68]$, and have capital gain > 500. The query result has 5 records. The strong IEQ $A_{3,2}$ provides more insights about this group of people: those in the age range $[64, 66]$ are highly educated, whereas the others in the age range $[67, 68]$ have high capital gains.

Query A_4 is a skyline query looking for people with maximal capital gain and minimal age. The query result includes four people. Both strong and weak IEQs return the same IEQs for this query. Interestingly, the precise IEQ $A_{4,1}$ provides a simplification of A_4: the people selected by this skyline query are (1) very young ($age \leq 17$) and have capital gain in the range $1055 - 27828$, or (2) have very high capital gain (> 27828), work in the protective service, and whose race is classified as "others".

Baseball. In query B_1, we want to find all players who belong to team "ARI" in 2006 and have a high performance ($HR > 10$). The result includes 7 players. From the IEQ $B_{1,1}$, we know more information about these players' performance (G, RBI, etc.), and their personal information (e.g., birth year). In addition, from IEQ $B_{1,2}$, we also know that one player in this group got an award when he played in "NL" league.

In query B_2, we want to find the set of high performance players who have very high total home runs (> 600). There are four players with these characteristics. The IEQ $B_{2,1}$ indicates that some of these players play for "ATL" or "NY1" team. The IEQ $B_{2,2}$ indicates one player in this group is highly paid and has a left throwing hand.

Q	IEQ	$\|p_a\|$	$\|p_b\|$	$\|p_c\|$
$A_{1,4}$	σ_{p_1} $(adult)$ $p_1 = (48 < hpw \leq 50 \wedge race \neq$ "Eskimo","Asian" $\wedge$ $6849 < gain \leq 7298 \wedge loss \leq 0 \wedge edu\_num > 14)$	1	1	13
$A_{2,3}$	$\sigma_{ms=\text{"Married-AF"} \wedge nc=\text{"Philippines"}}$ $(adult)$	1	0	0
$A_{2,4}$	$\sigma_{race=\text{"Asian"} \wedge rs=\text{"Wife"} \wedge hpw>52 \wedge age \leq 30}$ $(adult)$	1	0	0
$A_{3,2}$	$\sigma_{p_1 \vee p_2}$ $(adult)$ $p_1 = (63 < age \leq 66 \wedge edu > 15 \wedge ms =$ "NM") $p_2 = (66 < age \leq 68 \wedge ms =$ "NM" $\wedge gain >$ 2993)	5	0	0
$A_{4,1}$	$\sigma_{p_1 \vee p_2}$ $(adult)$ $p_1 = (1055 < gain \leq 27828 \wedge age \leq 17)$ $p_2 = (gain > 27828 \wedge occ =$ P $\wedge race \neq$ O)	4	0	0

Table 6: Strong IEQs on Adult

Q	IEQ	$\|p_a\|$	$\|p_b\|$	$\|p_c\|$
$B_{1,1}$	$\sigma_{p_1 \vee p_2}$ $(Master \bowtie Batting)$ $p_1 = (team =$ "ARI" $\wedge G \leq 156 \wedge 70 < RBI \leq 79 \wedge year > 1975)$ $p_2 = (team =$ "ARI" $\wedge G > 156 \wedge BB \leq 78)$	7	0	0
$B_{1,2}$	$\sigma_{lg=\text{"NL"} \wedge year=12 \wedge 71<height \leq 72 \wedge nc \neq \text{"D.R"}}$ $(Master \bowtie AwardsPlayer)$	1	0	6
$B_{2,1}$	$\sigma_{p_1 \vee p_2}$ $(Master \bowtie Batting)$ $p_1 = (BB \leq 162 \wedge HR > 46 \wedge team \in$ {"ATL", "NY1"} $\wedge RBI \leq 127)$ $p_2 = (BB > 162)$	4	0	0
$B_{2,2}$	$\sigma_{salary>21680700 \wedge throws=\text{"L"}}$ $(Master \bowtie Salaries)$	1	0	3
$B_{3,1}$	$\sigma_{(team=\text{"WS2"} \wedge R \leq 4) \vee (team=\text{"NYA"} \wedge state=\text{"LA"})}$ $(Master \bowtie Manager)$	2	0	33
$B_{3,2}$	$\sigma_{p_1 \vee p_2)}$ $(Master \bowtie Salaries)$ $p_1 = (height \leq 78 \wedge weight > 229 \wedge country =$ "DR" $\wedge$ $180000 < salary <$ 195000) $p_2 = (height > 78 \wedge state =$ "GA" $\wedge salary <$ 302500)	2	0	33
$B_{4,1}$	$\sigma_{21<L \leq 22 \wedge SB \leq 0 \wedge 67<W \leq 70}$ $(Mananger \bowtie Master \bowtie Batting \bowtie Team)$	1	0	5
$T_{1,1}$	$\sigma_{price>2096.99 \wedge 331<avaiqty \leq 1527}$ $(part)$	1	0	0
$T_{1,2}$	$\sigma_{avaiqty \leq 1 \wedge container=\text{"SM bag"}}$ $(part \bowtie part\_supp)$	1	0	0

Table 7: Weak IEQs on Baseball and TPCH

Query B_3 is a skyline query that looks for players with maximal number of home runs (HR) and minimal number of strike outs (SO). The result has 35 players. The IEQs provide different characterizations of these players. Query $B_{3,1}$ indicates that two players in this group are also the managers of "WS2" and "NYA" teams; while query $B_{3,2}$ indicates that two players in this group are averagely paid.

Query B_4 is an interesting query that involves multiple core relations. This query asks for the managers of team "CIN" from 1983 to 1988, the year they managed the team as well as the rank that the team gained. There are 3 managers in the result. In this query, we note that `TALOS` found alternative join-paths to link the two core relations, Manager and Team. The first alternative join-path (shown by $B_{4,1}$) involves Manager, Master, Batting, and Team. The second alternative join-path (not shown) involves Manager, Master, Fielding, and Team. The IEQ $B_{4,1}$ reveals the interesting observation that there is one manger who is also a player in the same year that he managed the team with some additional information about this manager-player.

TPC-H. Query T_1 retrieves the manufacturers who supply products of brand "brand#32". The result includes one manufacturer "Manufacturer#1". The IEQ $T_{1,1}$ indicates that this manufacturer also supplies some parts at a high price, where their available quantity is in the range $[332, 1527]$. The IEQ $T_{1,2}$ indicates that this manufacturer also supplies others products in the container named "SM bag" with an available quantity of at most one.

7. RELATED WORK

Although the title of our paper is inspired by Zloof's influential work on *Query by Example* (`QBE`) [25], the problem addressed by `QBE`, which is on providing a more intuitive form-based interface for database querying, is completely different from `QBO`.

There are several recent work [2, 3, 5, 14] that share the same broad principle of "reverse query processing" as `QBO` but differ totally in the problem objectives and techniques. Binnig et al. [2, 3] addressed the problem of generating test databases: given a query Q and a desired result R, generate a database D such that $Q(D) = R$. Bruno et al. [5] and Mishra et al. [14] examined the problem of generating test queries to meet certain cardinality constraints: given a query Q, a database D, and a set of target cardinality constraints on intermediate subexpression in Q's evaluation plan, modify Q to generate a new query Q' such that the evaluation plan of Q' on D satisfies the cardinality constraints.

An area that is related and complementary to `QBO` is intensional query answering (IQA) or cooperative answering, where for a given Q, the goal of IQA is to augment the query's answer $Q(D)$ with additional "intensional" information in the form of a semantically equivalent query[9] that is derived from the database integrity constraints [7, 15]. While semantic equivalence is stronger than instance equivalence and can be computed in a data-independent manner using only integrity constraints (ICs), there are several advantages of adopting instance equivalence for `QBO`. First, in practice, many data semantics are not explicitly captured using ICs in the database for various reasons [9]; hence, the effectiveness of IQA could be limited for `QBO`. Second, even when the ICs in the database are complete, it can be very difficult to derive semantically equivalent queries for complex queries (e.g., skyline queries that select dominant objects). By being data-specific, IEQs can often provide insightful and surprising characterizations of the input query and its result. Third, as IQA requires the input query Q to be known, IQA therefore cannot be applied to `QBO` applications where only $Q(D)$ (but not Q) is available. Thus, we view IQA and our proposed data-driven approach to compute IEQs as complementary techniques for `QBO`.

More recently, an interesting direction of using Précis queries [12, 19] has been proposed. The idea is to augment a user's query result with other related information (e.g., relevant tuples from other relations) and also allow the results to be personalized based on user-specified or domain requirements. The objectives of this work is orthogonal to `QBO`; and as in IQA, it is a query-driven approach that requires the input query to be known.

In the data mining literature, a somewhat related problem to ours is the problem of *redescription mining* introduced by Ramakrishnan [17]. The goal in redescription mining is to find different subsets of data that afford multiple descriptions across multiple vocabularies covering the same data

[9]Two queries Q and Q' are *semantically equivalent* if for every valid database D, $Q(D) \equiv Q'(D)$.

set. At an abstract level, our work is different from these methods in several ways. First, we are concerned with a fixed subset of the data (the output of the query). Second, none of the approaches for redescription mining account for structural (relational) information in the data (something we explicitly address). Third, redescription mining as it was originally posited requires multiple independent vocabulary descriptions to be identified. We do not have this requirement as we are simply interested in alternative query formulations within an SQL context. Finally, the notion of "at-least-one" semantics described in our work is something redescription mining is not concerned with as it is an artifact of the SQL context of our work.

8. CONCLUSION

In this work, we have introduced a new data-driven approach called query by output (`QBO`) targeted at improving the usability of database management systems. The goal of `QBO` is to discover instance-equivalent queries. Such queries can shed light on hidden relationships within the data, provide useful information on the relational schema as well as potentially summarize the original query.

We have developed an efficient system called `TALOS` for `QBO`, and our experimental results on several real database workloads of varying complexity highlight the benefits of our approach.

Although our discussions focus primarily on `QBO` where Q is known (to identify core relations), extending to the case without Q is feasible too: it requires an additional preprocessing phase to map each column of $Q(D)$ to a set of relation attributes by comparing the column contents. In addition, our approach also could be adapted to handle duplicates in $Q(D)$ by using the at-least-k semantics instead of at-least-one semantics.

As part of our future work, we plan to explore a hybrid approach that includes an off-line phase to mine for soft constraints in the database and an online phase that exploits both the database contents as well as mined constraints. Another interesting direction to be explored is to increase the expressiveness of IEQs (e.g., SPJ + union queries).

9. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their constructive suggestions to improve the paper's presentation. The third author would also like to acknowledge the following NSF grants: IIS-0347662 (CAREER) and CCF-0702587.

10. REFERENCES

[1] P. Andritsos, R. J. Miller, and P. Tsaparas. Information-theoretic tools for mining database structure from large data sets. In *SIGMOD*, pages 731–742, 2004.

[2] C. Binnig, D. Kossmann, and E. Lo. Reverse query processing. In *ICDE*, pages 506–515, 2007.

[3] C. Binnig, D. Kossmann, E. Lo, and M. T. Özsu. QAGen: generating query-aware test databases. In *SIGMOD*, pages 341–352, 2007.

[4] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[5] N. Bruno, S. Chaudhuri, and D. Thomas. Generating queries with cardinality constraints for dbms testing. *IEEE TKDE*, 18(12):1721–1725, 2006.

[6] P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren. Curated databases. In *PODS*, pages 1–12, 2008.

[7] T. Gaasterl, P. Godfrey, and J. Minker. An overview of cooperative answering. *Journal of Intelligent Information Systems*, (2):123–157, 1992.

[8] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *SIGMOD*, pages 461–472, 2001.

[9] P. Godfrey, J. Gryz, and C. Zuzarte. Exploiting constraint-like data characterizations in query optimization. In *SIGMOD*, pages 582–592, 2001.

[10] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *SIGMOD*, pages 13–24, 2007.

[11] T. Johnson, A. Marathe, and T. Dasu. Database exploration and bellman. 26(3):34–39, 2003.

[12] G. Koutrika, A. Simitsis, and Y. Ioannidis. Précis: The essence of a query answer. In *ICDE*, page 69, 2006.

[13] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *EDBT*, pages 18–32, 1996.

[14] C. Mishra, N. Koudas, and C. Zuzarte. Generating targeted queries for database testing. In *SIGMOD*, pages 499–510, 2008.

[15] A. Motro. Intensional answers to database queries. *IEEE TKDE*, 6(3):444–454, 1994.

[16] M. P.N. Tan and V.Kumar. *Introduction to Data Mining*. Addison-Wesley, 2006.

[17] N. Ramakrishnan, D. Kumar, B. Mishra, M. Potts, and R. F. Helm. Turning cartwheels: An alternating algorithm for mining redescriptions. In *KDD*, pages 266–275, 2004.

[18] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[19] A. Simitsis, G. Koutrika, and Y. E. Ioannidis. Generalized précis queries for logical database subset creation. In *ICDE*, pages 1382–1386, 2007.

[20] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy. Query by output. Technical Report TRA4/09, National University of Singapore - School of Computing, April 2009.

[21] P. Valduriez. Join indices. *ACM Trans. Database Syst.*, 12(2):218–246, 1987.

[22] C. J. van Rijsbergen. *Information Retireval*. Butterworth, 1979.

[23] W. Wu, B. Reinwald, Y. Sismanis, and R. Manjrekar. Discovering topical structures of databases. In *SIGMOD*, pages 1019–1030, 2008.

[24] X. Xiao and Y. Tao. Output perturbation with query relaxation. *Proc. VLDB Endow.*, 1(1):857–869, 2008.

[25] M. M. Zloof. Query by example. In *AFIPS NCC*, pages 431–438, 1975.