FlexIQ: A flexible interactive Querying Framework by Exploiting the Skyline Operator

FlexIQ: A Flexible Interactive Querying Framework by Exploiting the Skyline Operator

Md. Saiful Islam[a,b,*], Chengfei Liu[a], Rui Zhou[c]

[a] *Faculty of Science, Engineering and Technology, Swinburne University of Technology, Australia*
[b] *Institute of Information Technology, University of Dhaka, Bangladesh*
[c] *College of Engineering & Science, Victoria University, Australia*

Abstract

Skyline operator has gained much attention in the last decade and is proved to be valuable for multi-criteria decision making. This paper presents a novel **Flex**ible **I**nteractive **Q**uerying (*FlexIQ*) framework for user feedback-based Select-Project-Join (SPJ) query refinement in databases. In *FlexIQ*, the user feedback is used to discover the query intent. In addition, we have used the skyline operator to confine the search space of the proposed query refinement algorithms. The user feedback consists of both unexpected information currently present in the query output and expected information that is missing from the query output. Once the feedback is given by the user, our framework refines the initial query by exploiting the skyline operator to minimize the unexpected information as well as maximize the expected information in the refined query output. In our framework, the user can also control different quality metric such as quality of results (e.g., false positive rates, false negative rates and accuracy) and complexity (i.e., quantified as the number of subqueries) in the refined query. We have validated our framework both theoretically and experimentally. In particular, we have demonstrated the effectiveness of our proposed framework by comparing its performance with the naïve decision tree based query refinement.

Keywords: Skyline Operator, User Feedback, Query Refinement

1. Introduction

Traditional database query models such as SQL implement the binary retrieval technique. In this technique, a tuple is present in the result set only if it satisfies each constraint set by the user, e.g., conditions in a Select-Project-Join (SPJ) query. Therefore, a user may miss some expected results as well as receive some unexpected results in the query output if conditions in the query predicates are not set appropriately, which could be quite frequent for naïve or inexperienced users who lack knowledge of the underlying dataset. The outcome is, users get frustrated by receiving results that are unexpected to them with no explanation. The interface is designed to hide the details of the system, not to interact with the system. To handle the above, users usually go for a number of trials before getting the ultimate or precise one. In the worst case, they end up leaving the system which is undesirable. To help users in this regard, there are two possible solutions that any system can adopt and these are given below:

- modify the original tuple values in the database so that unexpected/expected tuples dissatisfy/satisfy the query predicates;
- modify the user submitted query by fixing the initial conditions so that the refined query excludes/includes unexpected/expected tuples from/in the refined query output.

Each of the above solutions is not suitable for every data and query setting, rather each has different application scopes based on specific data and query characteristics. For example, the first solution is suitable for applications where queries are static and data are untrusted. On the other hand, the second solution is suitable for applications where query refinements are allowed (i.e., queries are modifiable) and data are trusted. The techniques proposed by Huang et al. (Huang et al., 2008) and Herschel et al. (Herschel and Hernández, 2010) fall into the first category. However, their works are limited to missing (expected) tuples only. The techniques proposed by Tran et al. (Tran and Chan, 2010) and He et al. (He and Lo, 2012) fall into the second category and again these works are limited to missing (expected) tuples only. We believe that it would be more helpful for users if we can fix the initial query conditions, not only for expected tuples, but also for unexpected tuples under the same framework in trusted data applications where query modifications are allowed.

Example 1. *Consider a selection query which is issued by a user to a publication database and the corresponding result set as shown in Fig. 1, assume that the user has a different image of the query output in her mind and the result set does not match her expectation, she may then ask "How can I exclude* $P1$*,* $P3$ *and* $P10$ *from my query output? How can I include* $P11$ *and* $P12$ *into my query output?".*

Obviously, query refinement is a good solution for the above problem if data modifications are not allowed. Query refinement approach (instead of trial and error data selection process)

*Corresponding author
Email addresses: mdsaifulislam@swin.edu.au (Md. Saiful Islam), cliu@swin.edu.au (Chengfei Liu), rui.zhou@vu.edu.au (Rui Zhou)

User Query:
SELECT PubID FROM Publication
WHERE CitationCnt ≥ 80 AND
PubYear ≥ 1986;

Expected Tuples:

PubID	CitationCnt	PubYear
P_{11}	60	1995
P_{12}	72	1996
P_{13}	64	1996
P_{14}	67	1995

Query Output:

PubID	CitationCnt	PubYear
P_1	96	1989
$P2$	128	1986
P_3	100	1989
P_4	90	1993
P_5	148	1986
P_6	148	1990
P_7	81	1996
P_8	103	1986
P_9	82	1994
P_{10}	117	1987

Figure 1: Motivating example

can also be suitable for initial data selection in many other applications where the selected data (having certain features) are used for further processing (e.g., training/testing data in machine learning applications). Recently, Islam et al. (Islam et al., 2012a, 2013a) proposed a decision tree (DT) based query refinement technique for minimizing the unexpected information and maximizing the expected information in the refined query output. Tran et al. (Tran and Chan, 2010) also proposed DT based query refinement technique, but their proposed method is limited to expected tuples (i.e., why-not objects) only. On the other hand, Ma et al. (Ma et al., 2006) proposed DT based technique to exclude unexpected information (i.e., false positives) in the refined query output by learning both the structure and query conditions. To the best of our knowledge, there is no other work except the one proposed in (Islam et al., 2012a, 2013a) that makes a unified (considering both expected and unexpected tuples) attempt to solve the above problem.

In this paper, we propose a **Flex**ible **I**nteractive **Q**uerying (*FlexIQ*) framework[1] to address the aforementioned problem by exploiting the skyline operator. In our framework, the feedback is used to discover the query intent of the user and the skyline operator is used to confine the search space of the query refinement algorithms. In comparison to the naïve decision tree based query refinement, our framework promises to provide better results and proved to be more effective in minimizing the unexpected information as well as maximizing the expected information in the refined query output. We believe that the proposed querying framework can contribute towards developing a user friendly decision support system module on top of traditional data management systems (visions expressed in (Amer-Yahia et al., 2005; Jagadish et al., 2007)), and this model will help users to make decisions in situations where there is uncertainty about the possible outcomes of the initial submitted query.

1.1. Why skyline operator? Why-not DT?

The skyline operator retrieves all objects from the database that are not dominated by others (Börzsönyi et al., 2001). An object dominates another object if the first object is better than the second object in terms of at least one aspect and is equally good to the second object in terms of all other aspects. The skyline operator is well-known for its intuitive query formalization and easy to understand semantics (Su et al., 2010). This operator can be used to draw a boundary between answers and non-answers (explained in Section 2) of the user submitted query and can also be used to refine the initial query based on the feedback provided by the user (explained in Section 3, Section 4 and Section 5). The advantage of this skyline-operator-based-boundary is that it is data-driven and its formulation does not depend on any underlying data distribution. On the other hand, the DT is basically an information-gain-theory based linear classifier and its success relies heavily on the underlying data distribution (Mitchell, 1997). The construction of optimal decision trees from data is also an NP-complete problem (Hyafil and Rivest, 1976). In real life, relational data (i.e., query output) does not guarantee any statistical data distribution (though there are functional dependencies and sometimes correlations between attributes). Therefore, the DT based query refinement technique is not very successful in relational data settings if we do not have sufficient number of both negative and positive samples for DT learning, the performance of the DT based query refinement becomes very poor (experiments suggest this too). In contrast, our framework is free from the curse-of-data-distribution as it exploits the skyline operator (i.e., data driven).

1.2. Contributions

The main contributions of this paper are the followings:

- We provide the definition of data-driven query output semantics which serves as a summarized explanation of query output.
- We formally define the user feedback and its properties. We also describe approaches to derive minimal user feedback for the purpose of efficient query refinement.
- We then show how one can construct a new boundary for the refined query output by exploiting the skyline operator. We also propose a solution for the mentioned query refinement problem.

[1] A preliminary version of this work has been appeared in ER, 2012 (Islam et al., 2012b).

- Finally, we validate our approach with extensive experiments for three different datasets and demonstrate its effectiveness by comparing our results with DT based query refinement.

The remainder of the paper is organized as follows: Section 2 provides preliminaries, problem statement and overview of FlexIQ; Section 3 describes how we construct redundancy-free feedback and resolve user conflict; Section 4 describes how we redraw the boundary for the refined query output; Section 5 describes the query refinement techniques proposed in this paper; Section 6 presents experimental results; Section 7 describes the related work; and finally, Section 8 concludes the paper.

2. Background

2.1. Preliminaries

Let Q be the query, D be the database tuples (universe of discourse for Q), $Q(D)$ denote all tuples satisfied by the predicates given in Q, $Q'(D)$ denote all tuples not satisfied by the predicates given in Q, where $D = Q(D) \cup Q'(D)$. We use R and $Q(D)$ alternatively in this paper. Let G be the predicate preference in Q which consists of d atomic predicates i. e., $\{g_1, g_2, ..., g_d\}$. Each g_k is a triple $< a_k, op_k, v_k >$, where a_k is the attribute in D (i.e., column_name), op_k is the operator and v_k is the binding value. We assume that each atomic predicate g_k is equally important to the user. We consider G^d as the d-dimensional space where each dimension represents a particular preference $g_k \in G$. Therefore, each tuple $t_i \in D$ represents a d-dimensional point in G^d. We use $t_i.v_k$ to denote the k^{th} dimensional value of t_i so that t_i can be represented as $t_i =< t_i.v_1, t_i.v_2, ..., t_i.v_d >$. We use $Q \vdash t_i$ to indicate that t_i satisfies the preference G in Q and $Q \nvdash t_i$ to indicate that t_i does not satisfy the preference G in Q.

Definition 1. *(**Tuple Dominance**) A tuple t_i is said to dominate another tuple t_j, denoted by $t_i \succ_G t_j$, iff $\forall k \in \{1, 2, ..., d\}, t_i \succeq_{g_k} t_j$ and $\exists l \in \{1, 2, ..., d\}, t_i \succ_{g_l} t_j$; t_i is said to be as good as t_j, denoted by $t_i \succeq_G t_j$, iff $\forall k \in \{1, 2, ..., d\}, t_i \succeq_{g_k} t_j$.*

Without loss of generality, we only assume numerical attributes in the above definition. The relation $t_i \succeq_{g_k} t_j$ holds if $t_i.v_k \otimes t_j.v_k$, where $\otimes$ is op_k that appears in g_k with equality ('=') added. The relation $t_i \succ_{g_k} t_j$ holds if $t_i.v_k \otimes' t_j.v_k$, where $\otimes'$ is $\otimes$ with equality ('=') dropped. If relation $t_i \succ_G t_j$ holds for tuples t_i and t_j in D, then the relation $t_i \succeq_G t_j$ holds implicitly. The relation $t_i \succ_G t_j$ is known as *weak pareto-dominance* (Kießling, 2002; Voorneveld, 2003). It should be noted that $t_i \succeq t_i$, but $t_i \not\succ t_i$. That is, a tuple t_i does not dominate itself. The above definition can also be generalized for categorical attributes. However, if two tuples have different values for a categorical attribute, then they do not dominate each other in that dimension unless one of them satisfies the predicate or a partial order exists for it (Wong et al., 2008). We have only considered the numerical attributes in this paper.

Example 2. *Consider the dataset given in Fig. 1. We can see that tuple $P3$ dominates tuple $P1$ in terms of CitationCnt (i.e., $P3 \succ_{CitationCnt} P1$) and is as good as $P1$ in terms of PybYear (i.e., $P3 \succeq_{PybYear} P1$). Therefore, we say $P3 \succ_G P1$ (also $P3 \succeq_G P1$). Similarly, $P12 \succ_G P11$ (also $P12 \succeq_G P11$) and $P3 \succeq_G P3$ (but $P3 \not\succ_G P3$).*

Algorithm 1 Computing Boundary

Input: $Q(D)$ **Output:** Γ

1: $\Gamma \leftarrow Q(D)$;//initialization
2: **for** each $t_i \in \Gamma$ **do**
3: **if** $\exists t_j \in \Gamma$ such that $t_i \neq t_j$ and $t_i \succeq_G t_j$ **then**
4: Remove t_i from Γ;
5: **end if**
6: **end for**

Lemma 1. *For all $t_i, t_j \in D$, if $t_i \succeq_G t_j$ and $Q(D)$ includes t_j (i.e., $Q \vdash t_j$), then $Q(D)$ must include t_i (i.e., $Q \vdash t_i$). Similarly, if $t_i \succeq_G t_j$ and $Q(D)$ does not include t_i (i.e., $Q \nvdash t_i$), then $Q(D)$ must not include t_j(i.e., $Q \nvdash t_j$).*

Proof. We know that the predicate preference in Q is G and $t_i \succeq_G t_j$ (or $t_i \succ_G t_j$). That is, t_i satisfies the preferences in Q better than t_j (if $t_i \succ_G t_j$) or at least as t_j does (if $t_i \succeq_G t_j$). Therefore, the output of Q must include t_i. That is, $Q \vdash t_j$ and $t_i \succeq_G t_j \vee t_i \succ_G t_j$ implies $Q \vdash t_i$.
The proof of the second part is similar to the proof given for the first part. □

Definition 2. *(**Space**) A tuple t_j is said to be in space(t_i) iff $t_j \succeq t_i$. That is, space(t_i) includes each tuple t_j that dominates or is as good as t_i.*

Example 3. *The space($P1$) in $Q(D)$ consists of $\{P1, P3, P6\}$ as everyone of them dominates or is as good as $P1$.*

Definition 3. *(**Skyline**) A tuple $t_i \in Q(D)$ is said to be a skyline tuple for $Q(D) = \{t_1, t_2, ..., t_n\}$ in terms of G iff $\nexists m \in \{1, 2, ..., n\}, t_m \succ_G t_i$.*

In other words, the skyline of $Q(D)$ consists of all non-dominating tuples $t_i \in Q(D)$ that are not dominated by other tuples $t_j \in Q(D)$ in terms of G.

Example 4. *The skyline of $Q(D)$ consists of $\{P4, P6, P7, P9\}$.*

Definition 4. *(**Boundary**) A tuple $t_i \in Q(D)$ is said to be a boundary tuple for $Q(D) = \{t_1, t_2, ..., t_n\}$ in terms of G iff $\forall m \in \{1, 2, ..., n\}, t_i \not\succ_G t_m$.*

No two boundary tuples dominate each other. That is, if t_i and t_j are two boundary tuples, then $t_i \not\succ_G t_j$ and $t_j \not\succ_G t_i$. Let Γ be the *boundary* of $Q(D)$. Then, we compute Γ as follows:

- Γ is initialized to $Q(D)$ (step 1 in Algorithm 1) and
- for each $t_i \in \Gamma$, if $\exists t_j \in \Gamma$ such that $t_i \neq t_j$ and $t_i \succeq_G t_j$, then we remove t_i from Γ (steps 2 through 6 in Algorithm 1).

Complexity of Computing Boundary: Let us assume that checking the pairwise dominance between tuples (Definition 1) requires constant time. The for loop in Algorithm 1 (lines 2

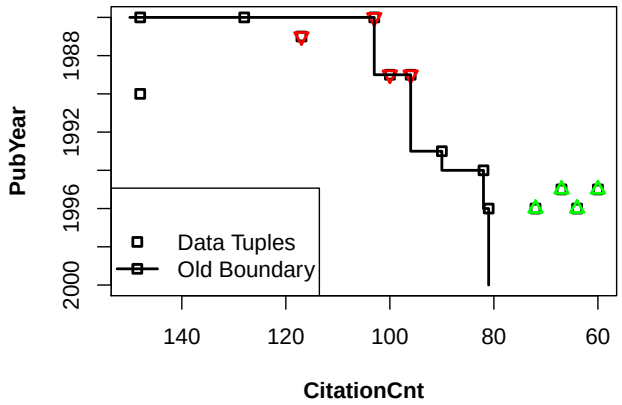


Figure 2: Data tuples and Old boundary: unexpected and expected tuples are marked with lower and upper triangles, respectively

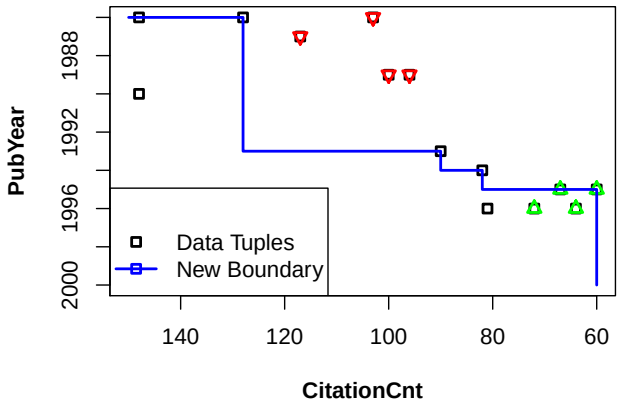


Figure 3: Data tuples and New boundary: unexpected and expected tuples are marked with lower and upper triangles, respectively

to 6) compares each tuple t_i with the rest of tuples t_j in $Q(D)$. If t_i dominates any other tuple in t_j, then t_i is removed from Γ. Therefore, the worst-case complexity of computing boundary Γ of $Q(D)$ is $O(n^2)$, where $n = |Q(D)|$.

There is a subtle difference between boundary tuples (defined in this paper) and traditional skyline tuples (Börzsönyi et al., 2001). Boundary tuples capture the contour of query output (i.e., define the boundary between $Q(D)$ and $Q'(D)$ as shown in Fig. 2) and are dominated by other tuples. On the contrary, skyline tuples usually refer to the best tuples in the query output and generally dominate other tuples (two skyline tuples do not dominate each other too) (Börzsönyi et al., 2001; Su et al., 2010). We twist the definition of traditional skyline tuples from "dominate" to "be dominated" to serve our purpose in this paper. The advantage of our proposed *boundary* is that we can construct the query output semantics by utilizing the member tuples of it (see Definition 5 given later in this section).

Example 5. *According to Definition 4 and Algorithm 1, we compute Γ for the example query output given in Fig. 1 as {P1, P4, P7, P8, P9}. These boundary tuples separate $Q(D)$ and $Q'(D)$ as shown in Fig. 2. We also see from Fig. 2 that this boundary includes the unexpected tuples (marked with red colored lower triangles) in $Q(D)$, but misses the expected tuples (marked with green colored upper triangles) .*

Definition 5. ***(Query Output Semantics)*** *Let Ω be the semantics of $Q(D)$. We then define Ω as consisting of $\{s_i\}$ and each s_i is defined as follows:*

$$s_i = \{< a_1, \otimes_1, t_i.v_1 >, ..., < a_d, \otimes_d, t_i.v_d >\}, \forall t_i \in \Gamma \qquad (1)$$

where a_k and $\otimes_k$ ($\otimes_k$ is op_k with equality ('=') added) come from $g_k =< a_k, op_k, v_k >$ and $d =| G |$. The s_i describes all tuples t_j that are in space(t_i).

Example 6. *According to Definition 5, we compute Ω of $Q(D)$ given in Fig. 1 as Ω={*
s_1={<CitationCnt, ≥, 96>, <PubYear, ≥, 1989>},
s_4={<CitationCnt,≥, 90>,<PubYear,≥,1993>},
s_7={<CitationCnt,≥,81>, <PubYear,≥,1996>},
s_8= {<CitationCnt, ≥, 103>,<PubYear, ≥, 1986>},
s_9={<CitationCnt, ≥, 82>, <PubYear, ≥, 1994>} }.
The s_1, s_4, s_7, s_8 and s_9 describe the tuple set {P1, P3, P6}, {P4}, {P7}, {P2, P5, P6, P8, P10} and {P9}, respectively.

Lemma 2. *Semantics Ω precisely describes the query output $Q(D)$.*

Proof. We know from the definition of the semantics Ω of $Q(D)$ that for every tuple $t_i \in \Gamma$ there exists an $s_i \in \Omega$. We also know from the definition of Γ that for all $t_l \in Q(D)$ there exists a boundary tuple $t_i \in \Gamma$ such that $t_l \geq t_i$. Therefore, we can say that Ω precisely describes the query output $Q(D)$. □

2.2. Problem Statement

Let U be the set of unexpected tuples, $U \subseteq R$ and E be the set of expected tuples, $E \subseteq Q'(D)$. We use $Q \vdash U$ to indicate that every tuple $t_i \in U$ satisfies $G \in Q$ and $Q \nvdash E$ to indicate that every tuple $t_j \in E$ does not satisfy $G \in Q$. We then formally define our query refinement problem as follows:

Definition 6. ***(Query Refinement Problem)*** *Given query Q, result set R, set of unexpected tuples U and set of expected tuples E, modify the initial query Q to Q^f in a way so that $t_i \in U$ does not satisfy the modified preference $G^f \in Q^f$ but $t_j \in E$ satisfies the modified preference $G^f \in Q^f$. In other words, we need to find the refined query Q^f such that the following holds:*

$$Q^f \vdash (R \setminus U) \cup E.$$

We argue that the discovery of the refined query Q^f relies heavily on the discovery of a new boundary that can separate unexpected and expected tuples. This can be easily observed from Fig. 2 and Fig. Fig. 3. This suggests that the query refinement problem is eventually transformed to the adjustment of boundary tuples for the refined query (described in Section 4). The next challenge is how we can transform this adjusted boundary to the refined query, Q^f. We explain this transformation step in Section 5.

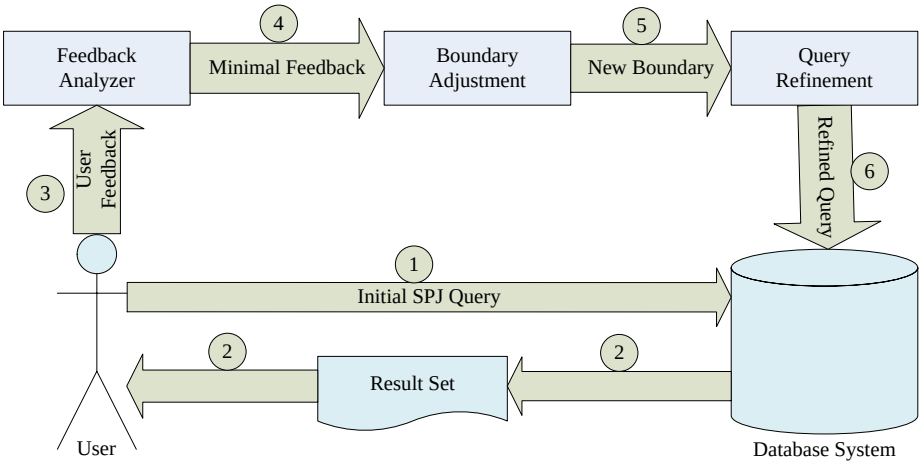


Figure 4: Working procedure of *FlexIQ*

2.3. *Overview of* FlexIQ

The proposed query refinement framework, *FlexIQ*, is user feedback driven. That is, a user actively participates in the solution process. Therefore, we collect both unexpected and expected tuples as feedback from the user after presenting the initial result set to her. Then, we analyze this feedback to find a minimal representation (described in Section 3). We also find the boundary that separates answers (i.e., $Q(D)$) from non-answers (i.e., $Q'(D)$), as shown in Fig. 2. Then, we form a new boundary that excludes unexpected tuples and includes expected tuples in $Q^f(D)$, as shown in Fig. 3 (Algorithm is given in Section 4). Then, we offer a baseline algorithm for constructing the refined query Q^f in relation to the new boundary (see Section 5.1). Finally, we offer a trade-off algorithm to minimize the number of clauses in the refined query so that it becomes semantically as close as possible to the original query (see Section 5.2). The overall working procedure of *FlexIQ* is shown in Fig. 4.

Table 1 provides an overview of the most basic symbols used in this paper.

Table 1: Overview of Symbols

Symbol	Description
D	Database
Q	User Query
G	Conditional Preferences in Q
R or $Q(D)$	Result Set or Query Output
$Q'(D)$	Non-resultant Tuples Set
U	Set of Unexpected Tuples
U^+	Extended Set of Unexpected Tuples
E	Set of Expected Tuples
E^+	Extended Set of Expected Tuples
Ω	Query Output Semantics
ζ	Minimal Set of Unexpected Tuples
ξ	Minimal Set of Expected Tuples
Γ'	Modified Set of Boundary Tuples
t_{min}	Minimal combination of t_i and t_{i+1}
t_{max}	Maximal combination of t_i and t_{i+1}
δ_i	Denotes $\|space(t_i)\|$
FS	Fitness Score
Q^f	Modified or Refined Query
$Q^f(D)$	Refined Query Output

3. User Feedback

We assume that the user (who issues the query Q to the system) may have little knowledge of the underlying dataset. Therefore, the user may find fixing the predicates (the binding values v_k in the predicate preferences g_k) in the given query to be laborious and troublesome. In this case, the system can work as a helping hand by receiving the feedback from the user and thereafter refining the initial predicates. Therefore, a user actively participates in the solution process (query refinement) in *FlexIQ*. That is, once the initial query result is presented to the user, she then identifies the portion of the current result that is unexpected (U) and the part of the new information that is expected (E) as shown in Fig. 4. The unexpected information can be labeled by the user from the current answer set. The expected information can be provided to the user as *you-may-also-like* (YMAL) results (Drosou and Pitoura, 2013) or relaxed results (Koudas et al., 2006) recommended by the system. The user can also provide concrete (e.g., tuple IDs) or even virtual (e.g., relaxed predicates) tuples or a combination of them as expected information in the new answer set.

In collecting the above feedback, there is a possibility of *conflict* and *computational redundancy* which is described below.

Definition 7. ***(Conflict)*** *Tuples t_i and t_j conflict with each other iff any of the following holds:*

- *$t_i \in U$ and $t_j \in Q(D) \setminus U$ such that $t_i \succeq_G t_j$;*
- *$t_i \in E$ and $t_j \in Q'(D) \setminus E$ such that $t_j \succeq_G t_i$ and*
- *$t_i \in U$ and $t_j \in E$ such that $t_i \succeq_G t_j$.*

We resolve conflicts as pre-processing in our framework. The basic idea of this pre-processing is checking the pairwise dominance between tuples of U and $Q(D) \setminus U$; E and $Q'(D) \setminus E$;

finally between tuples of U^+ and E^+, where U^+ and E^+ are the extended version of U and E respectively. The construction of U^+ is done as follows:

- U^+ is initialized to U (see step 1 in Algorithm 2) and
- for all $t_i \in U$ if there exists any $t_j \in Q(D) \setminus U$ such that $t_i \succeq t_j$, then t_j is added to U^+ (see steps 2 through 6 in Algorithm 2).

A similar approach is followed for the construction of E^+ (see steps 7 through 12 in Algorithm 2). Finally, we resolve user conflict by deleting tuples from E^+ that are dominated by tuples in U^+ (see steps 13 through 17 in Algorithm 2). The rationale of this deletion is that expected tuples are believed to be better than unexpected tuples in terms of G.

Complexity Analysis of Resolving Conflict in User Feedback: The unexpected tuple set U is provided by the user and we know $U \subseteq Q(D)$ (also we expect $|U| \ll |Q(D)|$). We reuse the pairwise dominance check performed for computing Γ (Algorithm 1) to compute U^+ (lines 2 to 6 of Algorithm 2). Therefore, the worst-case complexity of computing U^+ is $O(n)$ if the dominance relationships between tuples in $Q(D)$ is known. To compute E^+ in Algorithm 2, we do not compare each tuple $t_i \in E$ with the tuples $t_j \in Q'(D)$ as $Q'(D) \gg Q(D)$ (which is true for large database D). Rather, we compute a tuple $t_j \in \Gamma$ that dominates $t_i \in E$ (i.e., $t_j \succeq_G t_i$) and run a range query consisting of t_i and t_j in the database D. Assume that running a range query in the database D requires constant time. Therefore, the worst-case complexity of computing E^+ (lines 8 to 12 of Algorithm 2) is $O(|E| \times |\Gamma|)$. The complexity of executing lines 13 to 17 of Algorithm 2 is $O(|U^+| \times |E^+|)$. As $|\Gamma| \ll |Q(D)|$ and $|U^+| \ll |Q(D)|$, and also we expect $|E^+| \ll |Q(D)|$, the overall worst-case complexity of Algorithm 2 is $O(n^2)$.

Example 7. *Consider the user feedback U={$P1$, $P3$, $P10$} and E={$P11$, $P12$} as given in Fig. 1. Then, we compute U^+ as* {$P1$, $P3$, $P8$, $P10$} *and E^+ as* {$P11$, $P12$, $P13$, $P14$ }. *The tuple $P8$ is added to U^+ as $P10 \succeq_G P8$. Similarly, $P13$ and $P14$ are added to E^+ as $P13 \succeq_G P11$ and $P14 \succeq_G P11$.*

As user feedback may have conflict, we restate the query refinement problem given in Section 2.2 as follows:

Definition 8. ***(Redefined Query Refinement Problem)*** *Given query Q, result set R, set of unexpected tuples $U \subseteq R$ and set of expected tuples $E \subseteq Q'(D)$, we need to find a new refined query Q^f such that the following holds:*

$$Q^f \vdash (R \setminus U^+) \cup E^+.$$

The above problem statement allows users to incompletely define the feedback. That is, the user does not need to mention all of her feedback as long as other members of feedback are no better than the currently provided unexpected tuples and no worse than the currently provided expected tuples.

Definition 9. ***(Computational Redundancy)*** *We define computational redundancy as follows:*

Algorithm 2 Resolving User Conflict

Input: U and E **Output:** U^+ and E^+

1: $U^+ \leftarrow U$;//initialization
2: **for** each $t_i \in U$ **do**
3: **if** $\exists t_j \in Q(D) \setminus U$ such that $t_i \succeq_G t_j$ **then**
4: Add t_j to U^+; // t_j is no better than t_i
5: **end if**
6: **end for**
7: $E^+ \leftarrow E$;//initialization
8: **for** each $t_i \in E$ **do**
9: **if** $\exists t_j \in Q'(D) \setminus E$ such that $t_j \succeq_G t_i$ **then**
10: Add t_j to E^+; // t_j is as good as t_i
11: **end if**
12: **end for**
13: **for** each $t_i \in U^+$ **do**
14: **if** $\exists t_j \in E^+$ such that $t_i \succeq_G t_j$ **then**
15: Remove t_j from E^+;
16: **end if**
17: **end for**

- *A tuple $t_i \in U^+$ is said to be redundant wrt G iff $\exists t_j \in U^+$ such that $t_j \succeq_G t_i$.*
- *A tuple $t_i \in E^+$ is said to be redundant wrt G iff $\exists t_j \in E^+$ such that $t_i \succeq_G t_j$.*

In the above definition, we say t_i is computationally redundant because the exclusion of t_j ensures the exclusion of t_i in $Q^f(D)$ (see Lemma 1). Similarly, the inclusion of t_j ensures the inclusion of t_i in $Q^f(D)$ (Lemma 1).

Let ζ and ξ be the *redundancy-free* feedback for unexpected and expected information, respectively. Then, we define the properties of these sets as follows:

Definition 10. *$\forall t_i \in \zeta$ the following holds:*

- *$\nexists t_j \in U^+ \setminus \zeta$ such that $t_j \succ_G t_i$ and*
- *$\nexists t_j \in \zeta$ such that $t_j \succ_G t_i$.*

Definition 11. *$\forall t_i \in \xi$ the following holds:*

- *$\nexists t_j \in E^+ \setminus \xi$ such that $t_i \succ_G t_j$ and*
- *$\nexists t_j \in \xi$ such that $t_i \succ_G t_j$.*

The computation of ζ is done as follows:

- ζ is initialized to U^+ and
- for each $t_i \in \zeta$, if $\exists t_j \in \zeta$ such that $t_i \neq t_j$ and $t_j \succeq_G t_i$, then we remove t_i from ζ.

Complexity Analysis of Computing ζ: The complexity of computing ζ depends largely on the pairwise dominance check between tuples in U^+. We already know the dominance relationships between tuples in $Q(D)$ (while computing Γ in 1) and therefore, the dominance relationships between tuples in

U^+ are known too. We reuse them here. Assume that checking the above dominance information for each tuple $t_i \in U^+$ ($\exists t_j \in U^+$ such that $t_j \succeq_G t_i$) requires constant time and also $|U^+| \ll |Q(D)|$. Therefore, we get the worst-case complexity of computing ζ is $O(n)$ if the dominance relationships between tuples in U^+ (or $Q(D)$) is known.

Similarly, ξ is computed as follows:

- ξ is initialized to E^+ and
- for each $t_i \in \xi$, if $\exists t_j \in \xi$ such that $t_i \neq t_j$ and $t_i \succeq_G t_j$, then we remove t_i from ξ.

Complexity Analysis of Computing ξ: The complexity of computing ξ depends on the pairwise dominance check between tuples in E^+. Assume that checking the dominance between tuples in E^+ requires constant time and $|E^+| \ll |Q(D)|$. We get the worst-case complexity of computing ξ is $O(n^2)$.

We say that the minimal user feedback consists of ζ and ξ.

Example 8. *Consider the U^+ and E^+ given in Example 7. According to the definition of ζ and ξ given above, we get ζ={P3, P10} and ξ={P11}.*

Proposition 1. *The ζ and ξ are the necessary and sufficient information needed for updating G to exclude U^+ and include E^+ into $Q^f(D)$.*

Proof. We know that $\zeta \subseteq U^+$ and $\xi \subseteq E^+$. Definition 10 ensures $\forall t_i \in \zeta$, $\nexists t_j \in U^+ \setminus \zeta$ such that $t_j \succ_G t_i$ and $\forall t_i \in \zeta$, $\exists t_j \in U^+$ such that $t_i \succeq_G t_j$. Therefore, ζ is the necessary and sufficient information needed for updating G (i.e., the initial query Q) for the exclusion of U^+ from $Q^f(D)$ (follows from Lemma 1). Similarly, Definition 11 ensures $\forall t_i \in \xi$, $\nexists t_j \in E^+ \setminus \xi$ such that $t_i \succ_G t_j$ and $\forall t_j \in E^+$, $\exists t_i \in \xi$ such that $t_j \succeq_G t_i$. Therefore, ξ is the necessary and sufficient information needed for updating G (i.e., the initial query Q) for the inclusion of E^+ into $Q^f(D)$ (follows from Lemma 1). Therefore, we say that ζ and ξ are the necessary and sufficient information needed for updating G to exclude U^+ and include E^+ into $Q^f(D)$, respectively. □

Lemma 3. *The refined query Q^f can be constructed from the result set R and the minimal feedback ζ and ξ such that $Q^f \vdash R \setminus U^+ \cup E^+$.*

Proof. It follows from proposition 1. □

4. Boundary Adjustment

We have described in Section 2 that the query refinement problem is essentially transformed to be the adjustment (i.e., refinement) of original boundary tuples for $Q^f(D)$. That is, the adjusted boundary captures the semantics of the refined query output $Q^f(D)$ (as the original boundary captures the semantics of $Q(D)$). We prove this later in this section. Now, let Γ be the original boundary between $Q(D)$ and $Q'(D)$, ζ and ξ be the minimal feedback for the unexpected and expected tuples respectively. Then, we define our boundary adjustment problem as follows:

Definition 12. ***(Boundary Adjustment Problem)*** *Given the original boundary Γ of the initial query Q and minimal feedback ζ and ξ, find the new boundary Γ' for the refined query Q^f such that $Q^f \vdash R \setminus U^+ \cup E^+$.*

The basic idea of our boundary adjustment algorithm is checking the pairwise dominance between feedback tuples and boundary tuples of the original query Q. That is, if any unexpected tuple t_i dominates any boundary tuple $t_j \in \Gamma$ then we adjust the boundary in a way so that t_i will not dominate any boundary tuple in Γ' again (steps 2 through 14 in Algorithm 3). We also apply similar idea to include expected tuples in the new boundary Γ' (steps 15 through 21 in Algorithm 3). Algorithm 3 implements the above and computes the new boundary Γ' given Γ, ζ and ξ for the refined query Q^f.

Example 9. *Given dataset in Fig. 1, ζ={P3, P10}, ξ={P11} and Γ={P1, P4, P7, P8, P9}, we compute the new boundary for $Q^f(D)$ according to Algorithm 3 as follows: Γ' is initialized to* {P1, P4, P7, P8, P9}.
Now, to encounter unexpected tuples we iterate for each member in ζ as follows*:*

- *(For P3 $\in$ ζ) Since P3 $\in$ ζ dominates P1 $\in$ Γ' and P3 $\in$ space(P1), P1 is removed from Γ'. We then compute space(P1) as* {P1, P3, P6} *and assign this set to $temp\_tuple\_set_1$. We then remove P1 and P3 from $temp\_tuple\_set_1$ as both of them is dominated by or as good as P3. We then compute boundary of $temp_tuple_set_1$ as* {P6} *and add it to Γ'. Now Γ' becomes* {P4, P6, P7, P8, P9}.
- *(For P10 $\in$ ζ) Since P10 $\in$ ζ dominates P8 $\in$ Γ' and P10 $\in$ space(P8), P8 is removed from Γ'. We then compute space(P8) as* {P2, P5, P6, P8, P10} *and assign this set to $temp\_tuple\_set_1$. We then remove P8 and P10 from $temp\_tuple\_set_1$ as both of them is dominated by or as good as P10. We then compute boundary of $temp\_tuple\_set_1$ as* {P2}*, also remove P6 from Γ' as P6 $\in$ space(P2) and finally, add P2 to Γ'. Now, Γ' becomes* {P2, P4, P7, P9}.

Now, to encounter expected tuples we iterate for each member in ξ as follows*:*

- *(For P11 $\in$ ξ) We compute space(P11) as* {P7, P11, P12, P13, P14}*. Therefore, we remove P7 from Γ' computed so far and add P11 to Γ'. Finally, Γ' becomes* {P2, P4, P9, P11}.

This new boundary {P2, P4, P9, P11} *clearly separates E^+ and U^+ as shown in Fig. 3.*

Complexity Analysis of Boundary Adjustment: The runtime complexity of Algorithm 3 largely depends on the *for* loops in line 2 and line 15, respectively. The *for* loop in line 2 of Algorithm 3 compares each tuple $t_i \in \zeta$ with the tuples $t_j \in \Gamma'$. However, as we already know the dominace relationships between tuples in $Q(D)$ (Algorithm 1) and therefore, the

Algorithm 3 Boundary Adjustment

Input: original boundary, Γ and minimal feedback, unexpected ζ and expected ξ
Output: new boundary Γ' for the refined query Q^f

1: $\Gamma' \leftarrow \Gamma$;
2: **for** each $t_i \in \zeta$ **do**
3: **if** $\exists t_j \in \Gamma'$ such that $t_i \succeq_G t_j$ and $t_i \in space(t_j)$ **then**
4: Remove t_j from Γ';
5: $temp\_tuple\_set_1 \leftarrow space(t_j)$;
6: **if** $\exists t_m \in temp\_tuple\_set_1$ such that $t_i \succeq t_m$ **then**
7: Remove t_m from $temp\_tuple\_set_1$;
8: **end if**
9: $temp_boundary \leftarrow$Compute boundary for $temp\_tuple\_set_1$;//Algorithm 1
10: $temp\_tuple\_set_2 \leftarrow space(temp\_boundary)$;
11: $\Gamma' \leftarrow \Gamma' \setminus temp\_tuple\_set_2$;
12: Add $temp\_boundary$ to Γ';
13: **end if**
14: **end for**
15: **for** each $t_i \in \xi$ **do**
16: **if** $\exists t_j \in \Gamma'$ such that $t_j \succeq_G t_i$ and $t_j \in space(t_i)$ **then**
17: Remove t_j from Γ'; Add t_i to Γ'; // $space(t_j) \subseteq space(t_i)$
18: **else**
19: Add t_i to Γ';
20: **end if**
21: **end for**

dominace relationships between tuples in ζ and Γ', checking the dominace $t_i \succeq_G t_j$ and computing $t_i \in space(t_j)$ in line 3 should be done in constant time for each tuple $t_i \in \zeta$. The line 6 should also run in constant time for the above reason. The code in line 9 runs in $O(n)$ time as the dominance relationships is already known. The lines 10-12 runs in constant time. Therefore, the worst-case complexity of executing lines 2 to 14 of Algorithm 3 is $O(n^2)$. The *for* loop in line 15 of Algorithm 3 compares each tuple $t_i \in \xi$ with the tuples $t_j \in \Gamma'$ and replace t_j with t_i iff (a) $t_j \succeq_G t_i$ and (b) $t_j \in space(t_i)$ hold. Assume that the operations (a) and (b) execute in constant time. Also, asumme that $|\xi| \ll |E| \ll |Q(D)|$ and $|\Gamma'| \ll |Q(D)|$. Therefore, the worst-case complexity of executing lines 15 to 21 of Algorithm 3 is $O(n^2)$. Considering the above two cases, the overall worst-case complexity of Algorithm 3 is $O(n^2)$.

The following proposition proves that the new boundary Γ' precisely captures the semantics of the refined query output $Q^f(D)$.

Proposition 2. *The new boundary Γ' precisely captures the semantics of the refined query output $Q^f(D)$.*

Proof. Let Ω' be the semantics of the refined query output $Q^f(D)$ and be constructed from Γ' according to the definition of query output *semantics* given in Section 2.1. Algorithm 3 ensures that $\forall t_i \in \Gamma'$, $\nexists t_j \in U^+$ such that $t_j \succeq t_i$. Algorithm 3 also ensures that $\forall t_j \in E^+$, $\exists t_i \in \Gamma'$ such that $t_j \succeq t_i$. In other words, Γ' defines the contour for $R \setminus U^+ \cup E^+$. Therefore, we can say that Γ' precisely captures the semantics of $Q^f(D)$. □

The following example shows how the new boundary precisely describes the refined query output (i.e., refined query output semantics).

Example 10. *According to Definition 5, we compute Ω' of $Q^f(D)$ given the new boundary $\Gamma' = \{P2, P4, P9, P11\}$ as follows: Ω'={*
s_2={<CitationCnt, ≥, 128>, <PubYear, ≥, 1986>},
s_4={<CitationCnt,≥, 90>,<PubYear,≥,1993>},
s_9={<CitationCnt, ≥,82>, <PubYear,≥,1994>},
s_{11}= {<CitationCnt, ≥, 60>, <PubYear, ≥, 1995>} }.
The s_2, s_4, s_9, and s_{11} describe the tuple set {P2, P5, P6}, {P4}, {P9} and {P7, P11, P12, P13, P14}, respectively. We can easily verify that the refined query output semantics do not describe the unexpected tuple set {P1, P3, P8, P10}, but precisely describe the expected tuple set {P11, P12, P13, P14} and other positive tuples {P2, P4, P5, P6, P7, P9} only.

5. Query Refinement

This section presents the final step of our framework. The basic idea of our query refinement framework is constructing the new boundary for the refined query Q^f by analyzing the user feedback and then transforming the new boundary to the refined query Q^f. In Section 4, we show how to construct the new boundary Γ' given the original boundary Γ and the minimal feedback ζ and ξ. Now, we provide two algorithms for constructing the refined query Q^f from the adjusted boundary Γ': a baseline algorithm and then a trade-off algorithm.

5.1. The Baseline Algorithm

The baseline algorithm (TBA) converts the adjusted boundary Γ' to disjunction of conjunctions for the refined query Q^f.

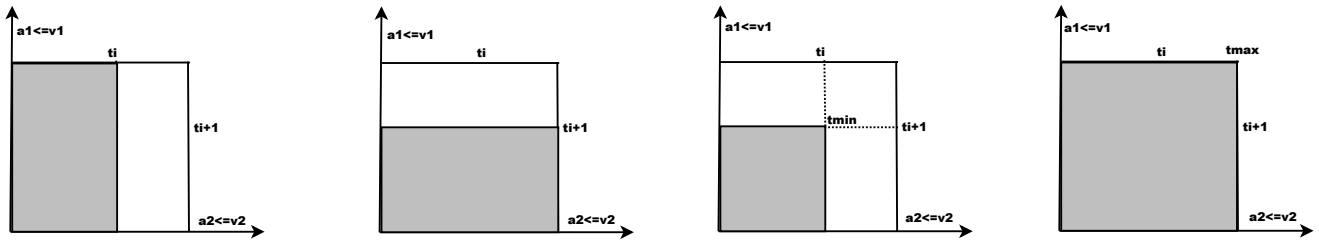


Figure 5: The four combinations of tuples t_i and t_{i+1} are t_i, t_{i+1}, t_{min} and t_{max}. The shaded area indicates the portion covered by the corresponding combination.

That is, the constituents s_i of the semantics Ω' constructed from Γ' are transformed directly to the disjunctive query. The transformation formula is given below:

$$Q^f \leftarrow \vee_{i=1}^{|\Omega'|} conjunct(s_i), \forall s_i \in \Omega' \quad (2)$$

The function $conjunct(s_i)$ returns the corresponding conjunction for $t_i \in \Gamma'$ as follows:

$$conjunct(s_i) \leftarrow a_1 \otimes_1 t_i.v_1 \wedge a_2 \otimes_2 t_i.v_2 \wedge ... \wedge a_d \otimes_d t_i.v_d \quad (3)$$

The size of the refined query Q^f returned by TBA is $|\Omega'|$ (also $|\Gamma'|$). That is, it consists of $|\Omega'|$ conjunctions (i.e., subqueries).

Example 11. *The baseline algorithm computes the refined query for the example dataset given in Fig. 1 as follows: SELECT pubid FROM publication WHERE ((citationcnt ≥ 128 AND pubyear ≥ 1986) OR (citationcnt ≥ 90 AND pubyear ≥ 1993) OR (citationcnt ≥ 82 AND pubyear ≥ 1994) OR (citationcnt ≥ 60 AND pubyear ≥ 1995)). It is easy to verify that the refined query consists of four subqueries as there four boundary tuples in the new boundary,* $\Gamma' = \{P2, P4, P9, P11\}$.

Lemma 4. *The refined query Q^f obtained by following Eq. 2 and Eq. 3 optimally separates unexpected and expected tuples. That is, Eq. 2 and Eq. 3 ensures $Q^f \vdash R \setminus U^+ \cup E^+$.*

Proof. The proof follows from Proposition 2. □

5.2. The Trade-Off Algorithm

The TBA converts each boundary tuple $t_i \in \Gamma'$ to a subquery for Q^f. The number of subqueries is equal to $|\Omega'|$, which is a major disadvantage of TBA. We can reduce this number of subqueries by combining them (i.e, approximation). That is, we replace one or more boundary tuples in Γ' with another one by combining them intelligently. Let us consider two tuples t_i and t_{i+1} from Γ'. We propose four types of combination strategies for these two tuples: (a) *maximal* (b) *minimal* (c) t_i and (d) t_{i+1}.

Definition 13. ***(maximal combination)*** *Two tuples t_i and t_{i+1} are said to be maximally combined to t_{max} only if $space(t_i) \cup space(t_{i+1}) \subseteq space(t_{max})$. The definition of $space(t_i)$ is the same as we define it in Section 2.1. The construction of t_{max} is done as follows:*

$$t_{max}.v_k \leftarrow \begin{cases} min(t_i.v_k, t_{i+1}.v_k) \text{ if } op_k \in \{\text{`} >= \text{'}, \text{`} > \text{'}\} \\ max(t_i.v_k, t_{i+1}.v_k) \text{ if } op_k \in \{\text{`} < \text{'}, \text{`} <= \text{'}\} \end{cases} \quad (4)$$

Definition 14. ***(minimal combination)*** *Two tuples t_i and t_{i+1} are said to be minimally combined to t_{min} only if $space(t_i) \cap space(t_{i+1}) \neq \emptyset$ and $space(t_i) \cap space(t_{i+1}) \subseteq space(t_{min})$. The construction of t_{min} is done as follows:*

$$t_{min}.v_k \leftarrow \begin{cases} max(t_i.v_k, t_{i+1}.v_k) \text{ if } op_k \in \{\text{`} >= \text{'}, \text{`} > \text{'}\} \\ min(t_i.v_k, t_{i+1}.v_k) \text{ if } op_k \in \{\text{`} < \text{'}, \text{`} <= \text{'}\} \end{cases} \quad (5)$$

Let G represent a two-dimensional space with predicates $a_1 \leq v_1$ and $a_2 \leq v_2$. Then, we can visualize t_i, t_{i+1}, t_{min} and t_{max} as we see it in Fig. 5. We replace t_i and t_{i+1} by any of them. The selection of t_{max} depends on whether any unexpected tuple $t_u \in U^+$ dominates any tuple $t_j \in space(t_{max})$ or not (i.e., valid combination). We define this condition as given as follows.

Definition 15. ***(Replacement Strategy)*** *if $\nexists t_u \in U^+$ such that $t_u \geq t_{max}$, then replace t_i and t_{i+1} with t_{max}.*

If the condition given above does not hold, then we select the one among t_i, t_{i+1}, t_{min} and t_{max} which retains the highest fraction of positive tuples from $space(t_i) \cup space(t_{i+1})$ and introduces fewer false positives. We term the fraction of truly positive tuples retained by any combination as *fitness_score* (FS). Let δ_i denotes $|space(t_i)|$, δ_{i+1} denotes $|space(t_{i+1})|$, δ_{min} denotes $|space(t_{min})|$ and δ_{max} denotes $|space(t_{max})|$. We define the *fitness_score* for t_i, t_{i+1}, t_{min} and t_{max} as given as follows:

$$FS(t_i) = \frac{\delta_i}{\delta_i + \delta_{i+1} - \delta_{min}} \quad (6)$$

$$FS(t_{i+1}) = \frac{\delta_{i+1}}{\delta_i + \delta_{i+1} - \delta_{min}} \quad (7)$$

$$FS(t_{min}) = \frac{\delta_{min}}{\delta_i + \delta_{i+1} - \delta_{min}} \quad (8)$$

$$FS(t_{max}) = \frac{\delta_i + \delta_{i+1} - \delta_{min}}{\delta_{max}} \quad (9)$$

To minimize the loss, we further maintain a predefined threshold, δ. That is, if the selected combination does not maintain its FS above δ then we leave t_i and t_{i+1} as they were before. Finally, we establish the order by which we can traverse the tuples in Γ'. We propose to sort the tuples in Γ' along any dimension and then consider successive pair of tuples for combination. The pseudo-code given in Algorithm 4 implements all these approximations described above. We call this algorithm as trade-off algorithm (TOA) in this paper.

Algorithm 4 Trade-Off Algorithm

Input: New Boundary Γ' **Output:** Refined Query Q^f

1: Sort Γ' in any dimension;
2: **for** $\forall t_i, t_{i+1} \in \Gamma'$ **do**
3: $t_{max} \leftarrow$ maximal combination of t_i and t_{i+1};//According to Eq. 4
4: **if** $\nexists t_u \in U^+$ such that $t_u \geq t_{max}$ **then**
5: Replace t_i, t_{i+1} with t_{max}; //t_{max} is a valid combination
6: **else**
7: $t_{min} \leftarrow$ minimal combination of t_i and t_{i+1};//According to Eq. 5
8: Compute $FS(t_i)$, $FS(t_{i+1})$, $FS(t_{min})$ and $FS(t_{max})$;//Accord. to Eq. 6 - Eq. 9
9: $t_c \leftarrow$ argmax $\{FS(t_i), FS(t_{i+1}), FS(t_{min}), F(t_{max})\}$;
10: **if** $FS(t_c) \geq \delta$ **then**
11: Replace t_i and t_{i+1} with t_c;// t_i and t_{i+1} is approximately replaced with t_c
12: **end if**
13: **end if**
14: **end for**
15: Compute Ω' for Γ';//According to Eq. 1
16: $Q^f \leftarrow \vee_{i=1}^{|\Omega'|} conjunct(s_i), \forall s_i \in \Omega'$// According to Eq. 2 and Eq. 3

Approximated Query:
SELECT pubid FROM publication WHERE
((citationcnt ≥ 128 AND pubyear ≥ 1986) OR
(citationcnt ≥ 60 AND pubyear ≥ 1993));

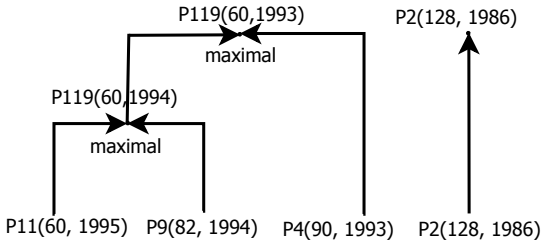


Figure 6: An example of computational steps made by TOA

Example 12. *Consider the new boundary given in Section 4 for the refined query output,* $\Gamma' = \{P2, P4, P9, P11\}$. *Now, TOA sorts the entries of* Γ' *in terms of citationcnt. The sorted list is* $\{P11, P9, P4, P2\}$. *Then, TOA combines* $P11$ *and* $P9$, *and replaces them with their maximal combination,* $P119(60, 1994)$ *as the FS value of* $P119$ *is greater than the predefined threshold,* δ. *Similarly,* $P119$ *is again combined with* $P4$ *and both of them is replaced by their maximal combination,* $P1194(60, 1993)$. *Now, TOA stops as* $P1194$ *and* $P2$ *cannot be combined as the FS value of their combination cannot succeed the according to the predefined threshold,* δ. *Therefore, the refined query returned by TOA for dataset given in Fig. 1 is: SELECT pubid FROM publication WHERE ((citationcnt ≥ 128 AND pubyear ≥ 1986) OR (citationcnt ≥ 60 AND pubyear ≥ 1993)). The construction of this query is illustrated in Fig. 6.*

Complexity Analysis of TOA: The trade-off algorithm starts with sorting the entries of Γ' in any dimension (Step 1 in Algorithm 4) which requires $O(|\Gamma'|log|\Gamma'|)$ time. The for loop in Steps 2 through 14 compares and combines successive entries in Γ' which requires $O(C_1 \times |\Gamma'|)$ time. The Steps 15 through 16 requires constant time again, i.e., $O(C_2 \times |\Gamma'|)$. Therefore, the overall complexity of TOA is $O(|\Gamma'| \times log|\Gamma'| + C \times |\Gamma'|)$, where $C = max(C_1, C_2)$. As we know $|\Gamma| \ll |Q(D)|$ and expect $|E^+| \ll |Q(D)|$, we get $|\Gamma'| \ll |Q(D)|$, the worst-case time complexity of TOA reduces to $O(nlogn + Cn) \equiv O(nlogn)$.

Considering the worst-case time complexities of all algorithms, i,e., Algorithm 1, Algorithm 2, Algorithm 3 and Algorithm 4, proposed in this paper, the overall time complexity of *FlexIQ* becomes $O(n^2)$, where we expect $|E^+| \ll |Q(D)|$.

6. Experiments

6.1. Setup

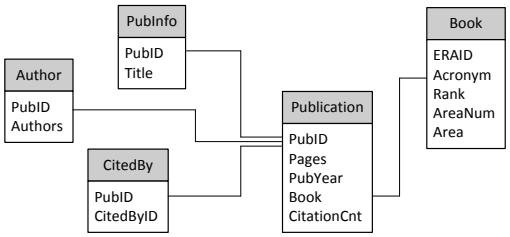
6.1.1. Environment:

We have run our experiments using an Intel(R) Core(TM) Duo E8400 3.0 GHz Windows XP PC with 3.49 GB RAM. The refinement algorithms have been implemented in Java along with MySQL server 5.1 and SDK is Eclipse 3.5.

6.1.2. Dataset and Queries

We have used DBLP dataset[2] of size 456 MB and two UCI machine learning datasets (Abalone and Automobile)[3]. We have converted the XML based DBLP dataset into relational data format which consists of five tables as shown in Fig. 7(a).

[2] http://dblp.uni-trier.de/xml/
[3] http://archive.ics.uci.edu/ml/



(a) DBLP dataset schema

Dataset	#size	#tables
DBLP	456 MB	5
UCI Automobile	64 KB	1 (26 columns)
UCI Abalone	304 KB	1 (9 columns)

(b) Dataset statistics

Figure 7: Datasets used in our experiment

Dataset	Group# 1 Base Query
DBLP	$\pi_{pubid}\sigma_{citationcnt \geq const_{11} \wedge pubyear \geq const_{12} \wedge pages \geq const_{13} \wedge rank \leq const_{14} \wedge book=acronym}(publication \times book)$
Automobile	$\pi_{id}\sigma_{compressionratio \geq const_{21} \wedge wheelbase \geq const_{22} \wedge price \geq const_{23} \wedge bore \geq const_{24}}(automobile)$
Abalone	$\pi_{id}\sigma_{diameter \leq const_{31} \wedge wholeweight \leq const_{32} \wedge shuckedweight \leq const_{33} \wedge rings \leq const_{34}}(abalone)$

Table 2: Group#1 base query for different datasets

We have also incorporated ERA conference/journal ranking information[4] into the DBLP dataset. The DBLP dataset schema and some statistics of the datasets used in our experiment are shown in Fig. 7(b). We have created a set of test queries consisting of both uni-point queries (no disjunction in the query condition) and multi-point queries (disjunction of several subqueries). We have also created two groups of uni-point test queries as described below:

- **Group #1**: To create the first group of queries, we have used a base query to create different test queries having different initial result size. For example, we have used the following base query to create 8 different queries (Q_1 to Q_8) of different initial result size (i.e., queries that return 15 to 178 tuples in the initial result set).

  $$\pi_{pubid}\sigma_{citationcnt \geq const_{11} \wedge pubyear \geq const_{12} \wedge pages \geq const_{13} \wedge rank \leq const_{14} \wedge book=acronym}(publication \times book)$$

  The purpose of this group of test queries is to justify the performance of the proposed algorithms as well as the effect of result size on the performance metrics. The base queries for other two datasets including DBLP dataset in this group are given in Table 2.

- **Group #2**: The second group of queries is created by having conditions on different attribute groups. The purpose of this group is to test the performance of the proposed algorithms on diverse queries and the corresponding diverse initial resultant tuples.

The uni-point test queries for all datasets follow the characteristics from both Group #1 and Group #2, while the multi-point test queries (i.e., queries having more than one subquery) follow the characteristics from Group #1 only. The queries also follow the distribution of the dataset in general. A summary of the tested queries for all datasets are given in Table 3.

[4] http://www.core.edu.au/

6.1.3. User Feedback

We randomly pick a set of tuples from the initial result set as unexpected tuples and limit the threshold to no more than $T\%$ of the resultant tuples. To get expected tuples, we pick tuples from non-answers (i.e., $Q'(D)$) by randomly relaxing certain predicates in the submitted query. Once we get expected tuples and unexpected tuples, we then construct *redundancy free feedback* and resolve the *conflict* (as described in Section 3).

6.1.4. Decision-Tree Based Query Refinement

A *decision tree* (DT) is a tree-like model of decisions. Given an input with well-defined attributes, the DT can classify the input entirely based on making choices about each attribute (Mitchell, 1997). Tran et al. (Tran and Chan, 2010) and Ma et al. (Ma et al., 2006) propose DT based query refinement techniques to handle expected tuples and unexpected tuples in the refined query output, respectively. To compare the effectiveness of our framework, we have used the DT based query refinement approach described in (Islam et al., 2012a) for managing both unexpected and expected tuples in the refined query output. We have also used the best-known and most widely-used C4.5 decision tree learning algorithm and WEKA implementation of it (Hall et al., 2009). To boost the performance of DT, we have set WEKA parameters as follows: *minNumObj* to 1, *unpruned* to False and *subtreeRaising* to True.

6.1.5. Evaluation Process

The performance of our proposed algorithms is evaluated in two different aspects: (a) quality of results and (b) query complexity.

- Quality of Results: Let n_{tp} be the number of truly positive tuples, n_{fp} be the number of false positive tuples (i.e., unexpected tuples), n_{tn} is the number of truly negative tuples and n_{fn} is the number of false negative tuples (i.e., expected tuples). The quality of results is then measured in terms of false positive rate (FPR), false negative rate

Dataset	Query Types	Query Groups (Uni-Point)	Initial Result Size
DBLP	Uni-point and Multi-point	Group#1 (Q_1 to Q_8), Group#2 (Q_9 to Q_{12})	Group#1 (15 to 178)& Group#2 (33 to 430)
UCI Automobile	Uni-point and Multi-point	Group#1 (Q_1 to Q_5), Group#2 (Q_6 to Q_{10})	Group#1 (41 to 120)& Group#2 (11 to 103)
UCI Abalone	Uni-point and Multi-point	Group#1 (Q_1 to Q_5), Group#2 (Q_6 to Q_{10})	Group#1 (56 to 358)& Group#2 (49 to 304)

Table 3: Characteristics of the tested queries in all datasets

(FNR) and accuracy (ACC) in IR-style (Mitchell, 1997) as follows:

$$FPR = \frac{n_{fp}}{n_{tp} + n_{fp}} \tag{10}$$

$$FNR = \frac{n_{fn}}{n_{tn} + n_{fn}} \tag{11}$$

$$ACC = \frac{n_{tp} + n_{tn}}{n_{tp} + n_{fp} + n_{tn} + n_{fn}} \tag{12}$$

- Query Complexity: We define query complexity as the number of subqueries (#NSUB) in the refined query.

We have compared the performances of the proposed query refinement algorithms as well as the DT based query refinement (Islam et al., 2012a) as follows: (a) we have made 100 different input instances for each tested query by randomly picking the result tuples as the unexpected tuples and setting the virtual tuples (randomly relaxing the predicates in the given query) from non-answers as the expected tuple set; and (b) finally, we have calculated the average of FPR, FNR and ACC (in %), and the refined query complexity (i.e., the number of subqueries(#NSUB)) of the proposed algorithms (TBA and TOA) and the DT based query refinement.

6.2. Results

6.2.1. Performance Summary

In this subsection, we have summarized and compared the performances of the proposed query refinement algorithms as well as DT based query refinement.

TBA vs Other Methods: The baseline algorithm (TBA) performs well compared to TOA and DT based query refinement in terms of quality of the refined query results. The FPR and FNR measures for TBA is 0% as we see these measures in Table 4 and Table 6. This means that TBA optimally separates answers from non-answers, as we have proved this in Section 5.1 (Lemma 4). Therefore, the ACC measure of TBA is 100% (see Table 5 and Table 7). To achieve this quality of results, TBA offers complex formulation of the refined queries compared to other methods as we see in Table 5 and Table 7 (#NSub) which is a major disadvantage of the baseline algorithm, TBA.

TOA vs DT: We have conducted nine different experiments for TOA by setting threshold δ to nine different values from ranging from 0.65 to 0.99[5] for both uni-point and multi-point test queries to demonstrate its effectiveness. The observed results are shown in Table 4, Table 5, Table 6 and Table 7.

The summaries of the observed uni-point test query results are given below:

- The TOA offers less FPR compared to the DT based query refinement if we set δ to $\geq$ 0.85 and $\geq$ 0.65 for group#1 and group#2 test queries in DBLP dataset on average, as we see in Table 4(a)-Table 4(b), respectively. The TOA offers less FPR if we set δ to $\geq$ 0.90 for group#2 test queries in both UCI Automobile and Abalone datasets on average (see Table 4(c)-Table 4(d)).

- The TOA offers less FNR for any setting of δ compared to the DT based query refinement for both group#1 and group#2 test queries in DBLP dataset on average, as we see in Table 4(a)-Table 4(b), and for group#2 test queries in Automobile dataset on average, as we see in Table 4(c). TOA also performs better or similar to DT in terms of FNR(%) for group#2 test queries in Abalone dataset on average except for experiments Exp#5 and Exp#6, as we see in Table 4(d).

- The TOA offers better ACC measure compared to the DT based query refinement for any setting of δ and for any group of test queries in DBLP and Automobile datasets on average as we see in Table 5(a), Table 5(b) and Table 5(c). We have achieved better accuracy by setting δ to 0.90 or above for for group#2 test queries in Abalone dataset on average as we see in Table 5(d).

- The refined query complexity offered by TOA depends on the particular setting of δ. We have achieved less complex queries for TOA compared to DT by setting δ to $\leq$ 0.80 and to $\leq$ 0.90 for group#1 and group#2 test queries in DBLP dataset on average as we see in Table 5(a) and Table 5(b), respectively. We have achieved less complex queries for TOA compared to DT by setting δ to $\leq$ 0.85 and to $\leq$ 0.90 for group#2 test queries in UCI Automobile and Abalone datasets, respectively (see in Table 5(c) and Table 5(d)).

The summaries of the observed multi-point test query results are given below:

- The TOA offers less FPR compared to the DT based query refinement if we set δ to $\geq$ 0.80 for multi-point test queries in DBLP dataset on average, as we see in Table 6(a). The TOA offers less FPR if we set δ to $\geq$ 0.90 for other two datasets on average as we see in Table 6(b) and Table 6(c), respectively.

- The TOA offers less FNR for any setting of δ compared to the DT based query refinement for multi-point test queries in DBLP dataset on average, as we see in Table 6(a). However, TOA performs worse or similar compared to DT in terms of FNR(%) in other two datasets as we see in Table 6(b) and Table 6(c). But, we can always achieve less

[5] We can set δ to any value in the range 0.0 to 1.0 and the effect of δ in TOA is explained later in this section.

Table 4: Average FPR(%) and FNR(%) in uni-point test queries

(a) DBLP dataset (group#1 queries)

Expr.	TBA		TOA		DT	
	FPR(%)	FNR(%)	FPR(%)	FNR(%)	FPR(%)	FNR(%)
Exp#1$_{(\delta=0.65)}$	0.00	0.00	26.27	06.39	09.72	30.47
Exp#2$_{(\delta=0.70)}$	0.00	0.00	20.94	06.75	09.66	29.20
Exp#3$_{(\delta=0.75)}$	0.00	0.00	15.32	07.71	08.99	25.97
Exp#4$_{(\delta=0.80)}$	0.00	0.00	11.25	06.83	08.79	30.05
Exp#5$_{(\delta=0.85)}$	0.00	0.00	05.74	06.19	14.34	27.99
Exp#6$_{(\delta=0.90)}$	0.00	0.00	02.18	04.54	09.57	30.74
Exp#7$_{(\delta=0.95)}$	0.00	0.00	00.38	01.35	09.79	28.59
Exp#8$_{(\delta=0.97)}$	0.00	0.00	00.11	01.23	09.54	30.47
Exp#9$_{(\delta=0.99)}$	0.00	0.00	00.00	00.04	08.06	28.96

(b) DBLP dataset (group#2 queries)

Expr.	TBA		TOA		DT	
	FPR(%)	FNR(%)	FPR(%)	FNR(%)	FPR(%)	FNR(%)
Exp#1$_{(\delta=0.65)}$	0.00	0.00	09.47	07.59	24.41	15.61
Exp#2$_{(\delta=0.70)}$	0.00	0.00	08.08	04.80	24.49	29.30
Exp#3$_{(\delta=0.75)}$	0.00	0.00	05.88	07.56	24.50	31.24
Exp#4$_{(\delta=0.80)}$	0.00	0.00	03.13	08.70	18.91	30.55
Exp#5$_{(\delta=0.85)}$	0.00	0.00	02.02	05.97	25.78	29.23
Exp#6$_{(\delta=0.90)}$	0.00	0.00	01.21	05.49	26.69	28.26
Exp#7$_{(\delta=0.95)}$	0.00	0.00	00.13	04.75	25.63	30.94
Exp#8$_{(\delta=0.97)}$	0.00	0.00	00.02	01.79	24.81	29.87
Exp#9$_{(\delta=0.99)}$	0.00	0.00	00.00	00.02	26.85	29.91

(c) Automobile dataset (group#2 queries)

Expr.	TBA		TOA		DT	
	FPR(%)	FNR(%)	FPR(%)	FNR(%)	FPR(%)	FNR(%)
Exp#1$_{(\delta=0.65)}$	0.00	0.00	6.62	29.17	1.27	35.81
Exp#2$_{(\delta=0.70)}$	0.00	0.00	9.86	24.84	1.06	45.22
Exp#3$_{(\delta=0.75)}$	0.00	0.00	6.98	24.64	1.27	45.59
Exp#4$_{(\delta=0.80)}$	0.00	0.00	4.04	25.83	1.13	46.04
Exp#5$_{(\delta=0.85)}$	0.00	0.00	1.63	19.70	1.38	46.71
Exp#6$_{(\delta=0.90)}$	0.00	0.00	0.58	10.89	1.31	44.70
Exp#7$_{(\delta=0.95)}$	0.00	0.00	0.35	5.75	1.42	45.84
Exp#8$_{(\delta=0.97)}$	0.00	0.00	0.03	3.07	1.30	45.74
Exp#9$_{(\delta=0.99)}$	0.00	0.00	0.00	0.00	1.25	35.97

(d) Abalone dataset (group#2 queries)

Expr.	TBA		TOA		DT	
	FPR(%)	FNR(%)	FPR(%)	FNR(%)	FPR(%)	FNR(%)
Exp#1$_{(\delta=0.65)}$	0.00	0.00	14.39	06.17	3.34	7.34
Exp#2$_{(\delta=0.70)}$	0.00	0.00	12.38	05.98	3.34	7.82
Exp#3$_{(\delta=0.75)}$	0.00	0.00	10.48	05.24	3.57	7.12
Exp#4$_{(\delta=0.80)}$	0.00	0.00	08.37	07.19	2.99	6.86
Exp#5$_{(\delta=0.85)}$	0.00	0.00	04.44	13.05	3.50	7.04
Exp#6$_{(\delta=0.90)}$	0.00	0.00	01.95	10.43	3.03	7.05
Exp#7$_{(\delta=0.95)}$	0.00	0.00	00.44	04.18	3.46	6.69
Exp#8$_{(\delta=0.97)}$	0.00	0.00	00.13	02.07	3.44	7.77
Exp#9$_{(\delta=0.99)}$	0.00	0.00	00.00	00.38	3.62	7.31

Table 5: Average ACC(%) and #NSUB in uni-point test queries

(a) DBLP dataset (group#1 queries)

Experiments	TBA		TOA		DT	
	ACC(%)	#NSUB	ACC(%)	#NSUB	ACC(%)	#NSUB
Exp#1$_{(\delta=0.65)}$	100.00	13.04	77.05	1.35	78.19	2.84
Exp#2$_{(\delta=0.70)}$	100.00	12.93	81.33	1.66	78.23	2.86
Exp#3$_{(\delta=0.75)}$	100.00	13.01	86.00	2.05	81.80	2.87
Exp#4$_{(\delta=0.80)}$	100.00	13.04	89.57	2.63	78.09	2.88
Exp#5$_{(\delta=0.85)}$	100.00	12.94	93.85	3.95	78.47	2.92
Exp#6$_{(\delta=0.90)}$	100.00	13.13	97.04	5.02	78.19	2.94
Exp#7$_{(\delta=0.95)}$	100.00	12.97	99.32	6.70	78.41	2.91
Exp#8$_{(\delta=0.97)}$	100.00	12.97	99.67	7.03	78.59	2.82
Exp#9$_{(\delta=0.99)}$	100.00	12.82	99.99	7.22	78.28	2.91

(b) DBLP dataset (group#2 queries)

Experiments	TBA		TOA		DT	
	ACC(%)	#NSUB	ACC(%)	#NSUB	ACC(%)	#NSUB
Exp#1$_{(\delta=0.65)}$	100.00	6.48	90.73	1.16	68.75	2.81
Exp#2$_{(\delta=0.70)}$	100.00	6.86	91.80	1.24	69.89	2.78
Exp#3$_{(\delta=0.75)}$	100.00	6.54	93.31	1.30	69.06	2.77
Exp#4$_{(\delta=0.80)}$	100.00	6.50	95.21	1.45	67.92	2.72
Exp#5$_{(\delta=0.85)}$	100.00	6.35	96.67	1.73	67.45	2.76
Exp#6$_{(\delta=0.90)}$	100.00	6.43	97.78	2.15	66.12	2.76
Exp#7$_{(\delta=0.95)}$	100.00	6.48	99.08	2.74	68.17	2.70
Exp#8$_{(\delta=0.97)}$	100.00	6.38	99.51	2.94	68.78	2.69
Exp#9$_{(\delta=0.99)}$	100.00	6.62	99.99	3.86	67.00	2.66

(c) Automobile dataset (group#2 queries)

Experiments	TBA		TOA		DT	
	ACC(%)	#NSUB	ACC(%)	#NSUB	ACC(%)	#NSUB
Exp#1$_{(\delta=0.65)}$	100.00	15.37	80.14	01.55	78.80	7.26
Exp#2$_{(\delta=0.70)}$	100.00	15.50	84.78	02.26	78.12	7.40
Exp#3$_{(\delta=0.75)}$	100.00	15.25	87.52	02.74	78.47	7.17
Exp#4$_{(\delta=0.80)}$	100.00	15.15	91.60	03.85	78.35	7.12
Exp#5$_{(\delta=0.85)}$	100.00	15.34	94.92	05.70	78.47	7.05
Exp#6$_{(\delta=0.90)}$	100.00	15.26	96.85	07.49	78.73	7.20
Exp#7$_{(\delta=0.95)}$	100.00	15.46	99.06	09.70	78.34	7.16
Exp#8$_{(\delta=0.97)}$	100.00	15.59	99.75	10.74	78.07	7.07
Exp#9$_{(\delta=0.99)}$	100.00	15.09	100.0	10.78	78.42	7.04

(d) Abalone dataset (group#2 queries)

Experiments	TBA		TOA		DT	
	ACC(%)	#NSUB	ACC(%)	#NSUB	ACC(%)	#NSUB
Exp#1$_{(\delta=0.65)}$	100.00	16.52	88.30	1.11	95.44	7.91
Exp#2$_{(\delta=0.70)}$	100.00	16.76	89.62	1.34	95.15	8.50
Exp#3$_{(\delta=0.75)}$	100.00	16.19	90.51	1.46	95.07	8.18
Exp#4$_{(\delta=0.80)}$	100.00	16.44	92.16	1.85	95.62	8.17
Exp#5$_{(\delta=0.85)}$	100.00	16.23	94.32	3.55	95.09	8.16
Exp#6$_{(\delta=0.90)}$	100.00	15.75	96.68	5.96	95.63	8.23
Exp#7$_{(\delta=0.95)}$	100.00	16.60	98.96	9.22	95.26	8.49
Exp#8$_{(\delta=0.97)}$	100.00	16.59	99.60	10.38	95.19	8.46
Exp#9$_{(\delta=0.99)}$	100.00	16.48	99.98	10.92	94.68	8.46

Table 6: Average FPR(%) and FNR(%) in multi-point test queries

(a) DBLP dataset

Expr.	**TBA**		**TOA**		**DT**	
	FPR(%)	FNR(%)	FPR(%)	FNR(%)	FPR(%)	FNR(%)
Exp#1$_{(\delta=0.65)}$	0.00	0.00	19.32	42.25	2.78	75.15
Exp#2$_{(\delta=0.70)}$	0.00	0.00	07.03	08.70	2.61	70.32
Exp#3$_{(\delta=0.75)}$	0.00	0.00	04.25	10.32	3.64	71.57
Exp#4$_{(\delta=0.80)}$	0.00	0.00	05.62	10.87	7.21	47.02
Exp#5$_{(\delta=0.85)}$	0.00	0.00	02.99	11.57	7.06	51.35
Exp#6$_{(\delta=0.90)}$	0.00	0.00	00.89	07.05	6.12	51.51
Exp#7$_{(\delta=0.95)}$	0.00	0.00	00.21	00.59	6.98	48.66
Exp#8$_{(\delta=0.97)}$	0.00	0.00	00.00	00.00	2.52	67.64
Exp#9$_{(\delta=0.99)}$	0.00	0.00	00.00	00.00	3.09	70.99

(b) Automobile dataset

Expr.	**TBA**		**TOA**		**DT**	
	FPR(%)	FNR(%)	FPR(%)	FNR(%)	FPR(%)	FNR(%)
Exp#1$_{(\delta=0.65)}$	0.00	0.00	10.68	56.51	1.09	4.01
Exp#2$_{(\delta=0.70)}$	0.00	0.00	06.96	54.01	1.40	2.53
Exp#3$_{(\delta=0.75)}$	0.00	0.00	06.33	26.49	1.12	4.08
Exp#4$_{(\delta=0.80)}$	0.00	0.00	03.48	29.26	1.38	3.18
Exp#5$_{(\delta=0.85)}$	0.00	0.00	02.47	26.72	1.07	2.98
Exp#6$_{(\delta=0.90)}$	0.00	0.00	00.81	14.26	1.12	3.26
Exp#7$_{(\delta=0.95)}$	0.00	0.00	00.07	13.82	0.57	2.67
Exp#8$_{(\delta=0.97)}$	0.00	0.00	00.01	08.98	1.09	2.71
Exp#9$_{(\delta=0.99)}$	0.00	0.00	00.00	00.00	0.83	3.51

(c) Abalone dataset

Expr.	**TBA**		**TOA**		**DT**	
	FPR(%)	FNR(%)	FPR(%)	FNR(%)	FPR(%)	FNR(%)
Exp#1$_{(\delta=0.65)}$	0.00	0.00	9.37	00.00	4.64	5.41
Exp#2$_{(\delta=0.70)}$	0.00	0.00	9.63	00.98	4.89	3.36
Exp#3$_{(\delta=0.75)}$	0.00	0.00	9.81	04.81	5.41	5.01
Exp#4$_{(\delta=0.80)}$	0.00	0.00	7.37	24.14	4.98	5.87
Exp#5$_{(\delta=0.85)}$	0.00	0.00	6.05	26.84	4.88	5.78
Exp#6$_{(\delta=0.90)}$	0.00	0.00	4.18	36.02	4.47	5.57
Exp#7$_{(\delta=0.95)}$	0.00	0.00	0.67	27.08	5.36	4.55
Exp#8$_{(\delta=0.97)}$	0.00	0.00	0.18	13.91	5.25	4.47
Exp#9$_{(\delta=0.99)}$	0.00	0.00	0.01	02.05	5.25	2.97

Table 7: Average ACC(%) and #NSUB in multi-point test queries

(a) DBLP dataset

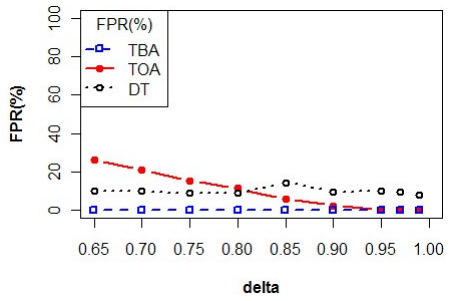
Experiments	**TBA**		**TOA**		**DT**	
	ACC(%)	#NSUB	ACC(%)	#NSUB	ACC(%)	#NSUB
Exp#1$_{(\delta=0.65)}$	100.00	9.16	78.09	1.83	64.29	4.34
Exp#2$_{(\delta=0.70)}$	100.00	9.06	91.43	1.89	65.49	4.48
Exp#3$_{(\delta=0.75)}$	100.00	9.15	93.92	1.97	65.04	4.58
Exp#4$_{(\delta=0.80)}$	100.00	9.01	91.84	2.34	69.92	4.54
Exp#5$_{(\delta=0.85)}$	100.00	8.84	94.26	2.79	68.94	4.80
Exp#6$_{(\delta=0.90)}$	100.00	8.96	97.25	3.45	68.63	4.78
Exp#7$_{(\delta=0.95)}$	100.00	8.81	99.61	4.49	68.55	4.68
Exp#8$_{(\delta=0.97)}$	100.00	9.13	100.0	3.79	66.04	4.70
Exp#9$_{(\delta=0.99)}$	100.00	9.13	100.0	3.61	68.08	4.90

(b) Automobile dataset

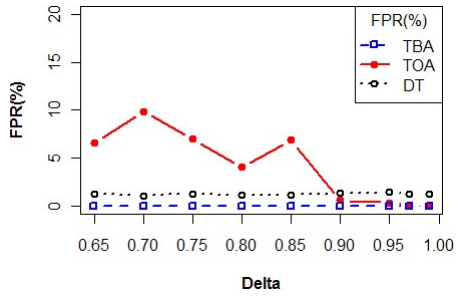
Experiments	**TBA**		**TOA**		**DT**	
	ACC(%)	#NSUB	ACC(%)	#NSUB	ACC(%)	#NSUB
Exp#1$_{(\delta=0.65)}$	100.00	15.24	72.09	01.35	97.27	7.06
Exp#2$_{(\delta=0.70)}$	100.00	15.96	75.30	01.51	97.71	7.64
Exp#3$_{(\delta=0.75)}$	100.00	14.99	85.42	02.12	97.56	7.00
Exp#4$_{(\delta=0.80)}$	100.00	15.79	86.82	02.88	97.44	7.72
Exp#5$_{(\delta=0.85)}$	100.00	15.84	90.88	04.24	98.01	7.78
Exp#6$_{(\delta=0.90)}$	100.00	15.68	96.56	06.04	97.72	7.48
Exp#7$_{(\delta=0.95)}$	100.00	15.48	98.57	08.07	98.35	7.76
Exp#8$_{(\delta=0.97)}$	100.00	15.21	99.33	08.65	97.85	7.28
Exp#9$_{(\delta=0.99)}$	100.00	16.03	100.0	10.14	97.84	8.00

(c) Abalone dataset

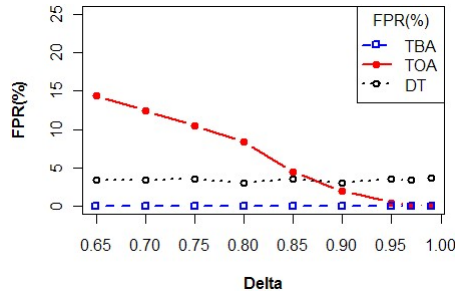
Experiments	**TBA**		**TOA**		**DT**	
	ACC(%)	#NSUB	ACC(%)	#NSUB	ACC(%)	#NSUB
Exp#1$_{(\delta=0.65)}$	100.00	132.59	90.65	01.07	95.01	17.68
Exp#2$_{(\delta=0.70)}$	100.00	132.46	90.34	01.24	95.01	17.85
Exp#3$_{(\delta=0.75)}$	100.00	133.64	89.72	01.45	94.29	21.49
Exp#4$_{(\delta=0.80)}$	100.00	134.82	90.70	02.55	94.70	15.75
Exp#5$_{(\delta=0.85)}$	100.00	134.04	91.18	05.86	94.67	19.00
Exp#6$_{(\delta=0.90)}$	100.00	135.06	91.96	10.40	95.20	20.30
Exp#7$_{(\delta=0.95)}$	100.00	134.44	96.65	22.22	94.40	16.07
Exp#8$_{(\delta=0.97)}$	100.00	134.08	98.44	28.90	94.63	19.56
Exp#9$_{(\delta=0.99)}$	100.00	135.78	99.92	30.69	94.73	18.67



(a) DBLP

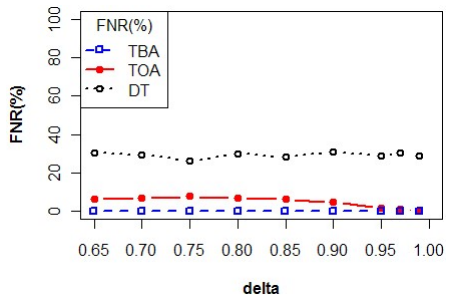


(b) Automobile

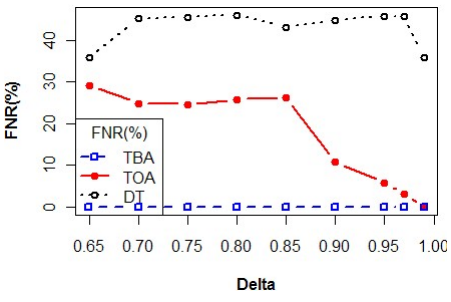


(c) Abalone

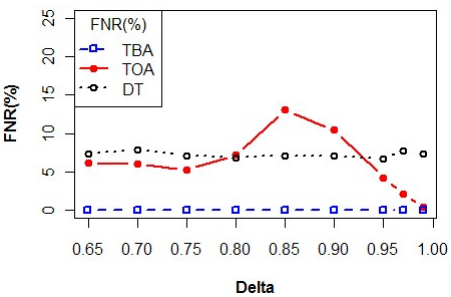
Figure 8: Effect of delta (δ) on FPR(%)



(a) DBLP



(b) Automobile



(c) Abalone

Figure 9: Effect of delta (δ) on FNR(%)

FNR for TOA by setting δ close to 1.0 as we see in Table 6(a)-Table 6(c).

- The TOA offers better ACC measure compared to the DT based query refinement for any setting of δ in DBLP dataset, on average, as we see in Table 7(a). For other two datasets, TOA offers better ACC measure if we set δ to 0.95 or above as we see in Table 7(b) and Table 7(c).

- We have achieved less complex queries for TOA compared to DT for multi-point test queries by setting δ to ≤ 0.99 in DBLP dataset, on average, as we see in Table 7(a). For other two datasets, we have achieved less complex refined queries by setting δ to ≤ 0.90, on average, as we see in Table 7(b) and Table 7(c), respectively.

6.2.2. Effect of Input Parameters

In this subsection, we have described the effects of different input parameters on the performances of the proposed query refinement algorithms and the DT based query refinement.

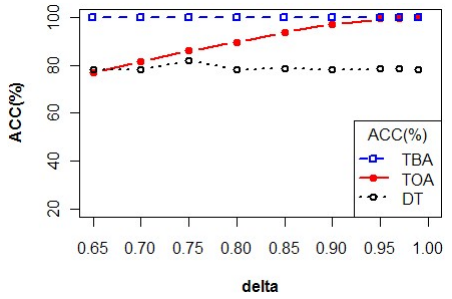
Effect of δ in TOA: The effect of δ on FPR, FNR, ACC and #NSUB in TOA can be observed from graphs shown in Fig. 8, Fig. 9, Fig. 10 and Fig. 11. The FPR and FNR tend to zero in TOA if δ tends to 1.0 and the complexity of the refined queries tends to be minimum if δ tends to zero. The ACC measure for TOA also tends to 100% as δ tends to 1.0. In contrast, the complexity of the refined query tends to be the minimum if δ tends to zero. Therefore, we can trade-off δ in TOA to achieve different quality metrics (quality of results and refined query size) as follows:

- If we are interested in low false positive rates and low false negative rates (as well as better ACC measure) we can set δ close to 1.0.

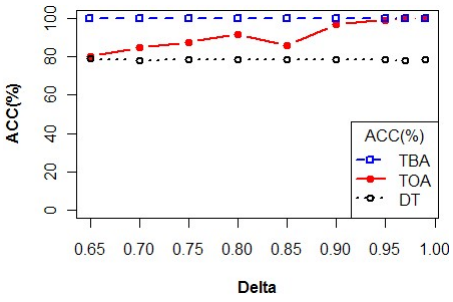
- On the other hand, if we need less complex refined queries we can set δ to $\ll 1$.

The δ has no effect on the performances of TBA and DT-based query refinement.

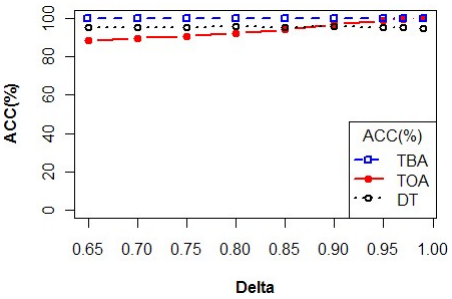
Effect of dimensionality: The effect of the number of (unique) attributes used in the query predicates has no or little effect on the performances of the proposed trade-off algorithm TOA as we see in Fig. 12, Fig. 13, 14 and Fig. 15. The observed results are summarized as follows:



(a) DBLP

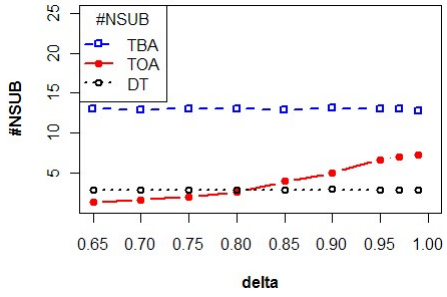


(b) Automobile

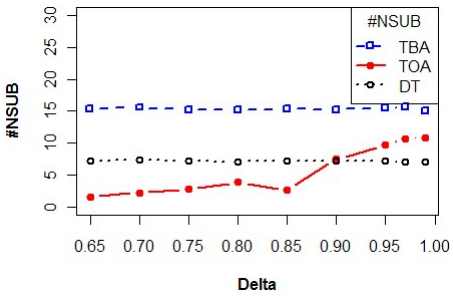


(c) Abalone

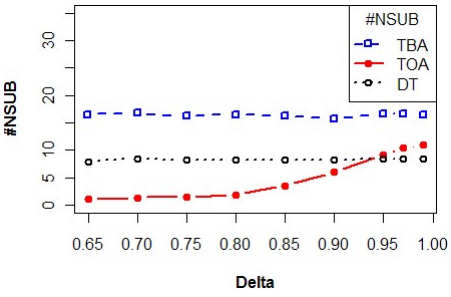
Figure 10: Effect of delta (δ) on ACC(%)



(a) DBLP



(b) Automobile



(c) Abalone

Figure 11: Effect of delta (δ) on #NSUB

- The FPR(%) tends to increase slowly in TOA as the number of attributes used in the query predicates increase for DBLP and Abalone datasets except Automobile dataset as we see in Fig. 12(a)-Fig. 12(c). However, FPR(%) tends to decrease in DT as the number of attributes used in the query predicates increase for all datasets. The number of (unique) attributes has no effect in TBA in terms of FPR(%) which is always 0%.

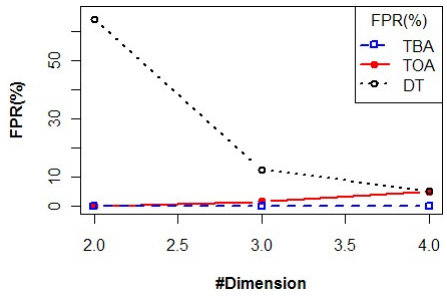
- The FNR(%) tends to decrease in both TOA and DT as the number of attributes used in the query predicates increase for DBLP and Abalone datasets except Automobile dataset as we see in Fig. 13(a)-Fig. 13(c). Again, the number of (unique) attributes has no effect in TBA in terms of FNR(%) which is always 0%.

- The ACC(%) tends to increase or remains almost constant in TOA as the number of attributes used in the query predicates increase for all datasets as we see in Fig. 14(a)-Fig. 14(c). However, ACC(%) tends to increase in DT as the number of attributes used in the query predicates increase for all datasets except Automobile dataset. Again, the number of (unique) attributes has no effect in TBA in terms of ACC(%) which is always 100%.

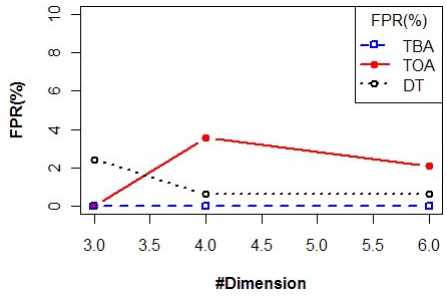
- The refined query complexities increase in both DT and TBA as the number of attributes used in the query predicates increase for all datasets as we see in Fig. 15(a)-Fig. 15(c).

Effect of User Feedback: We measure user feedback as the number of unexpected tuples selected randomly from the initial result set (T%). Recall that we have resolved conflict in Section 3 as follows "if any tuple $t_i \in E^+$ is dominated by any tuple $t_j \in U^+$, we remove t_i from E^+" (lines 13 to 17 in Algorithm 2). Therefore, for each random selection of unexpected tuple set we also receive varied expected tuple set. The effect of user feedback is shown in Fig. 16, Fig. 17, Fig. 18 and Fig. 19. The observed results are summarized as follows:

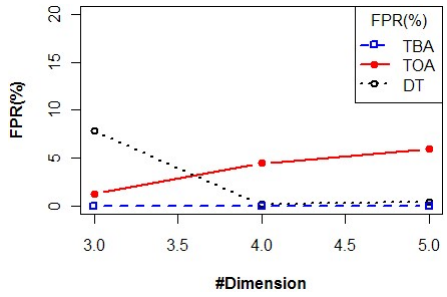
- The FPR(%) tends to decrease in TOA as the user feedback increases for both DBLP and Abalone datasets except Automobile dataset as we see in Fig. 16(a)-Fig. 16(c). However, FPR(%) tends to increase in DT as the user feedback increases for all datasets. The user feedback has no effect in TBA in terms of FPR(%) which is always 0%.



(a) DBLP

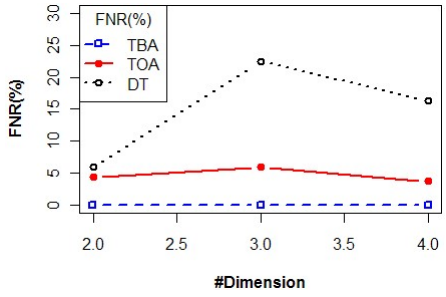


(b) Automobile

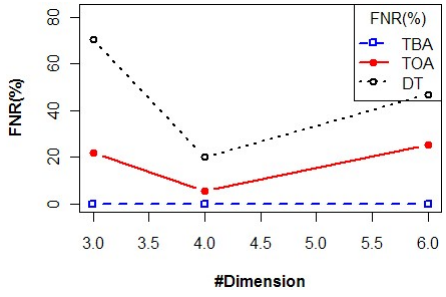


(c) Abalone

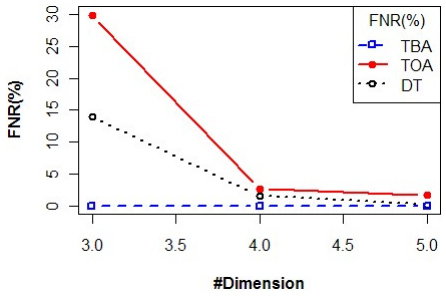
Figure 12: Effect of dimension size on FPR(%)



(a) DBLP



(b) Automobile



(c) Abalone

Figure 13: Effect of dimension size on FNR(%)

- The FNR(%) tends to decrease in both TOA and DT as the user feedback increases for all datasets as shown in Fig. 17(a)-Fig. 17(c). Again, the user feedback has no effect in TBA in terms of FNR(%) which is always 0%.

- The ACC(%) tends to increase in both TOA and DT as the user feedback increases for all datasets as shown in Fig. 18(a)-Fig. 18(c). Again, the user feedback has no effect in TBA in terms of ACC(%) which is always 100%.

- The refined query complexities have little or no effect in TOA as the user feedback increases as we see in Fig. 19(a)- Fig. 19(c). However, #NSUB tends to decrease in both TBA and DT as the user feedback increases for all datasets.

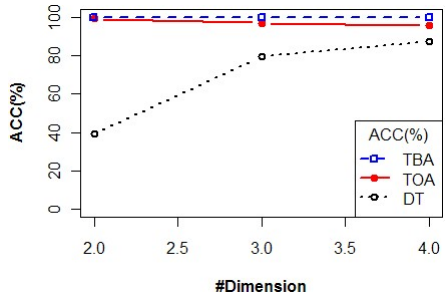
Effect of Initial Result Size: The effect of initial query result size is shown in Fig. 20, Fig. 21, Fig. 22 and Fig. 23. The observed results are summarized as follows:

- The FPR(%) tends to remain almost constant in TOA as the initial query result size increases for all datasets as we see in Fig. 20(a)-Fig. 20(c). The FPR(%) tends to remain almost constant in DT too with few exceptions. The initial query result size has no effect in TBA in terms of FPR(%) which is always 0%.

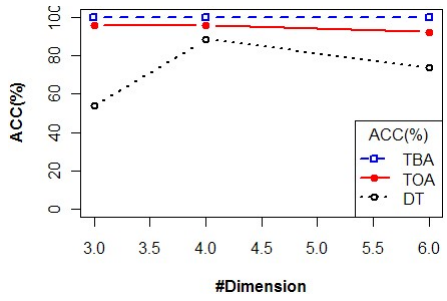
- The FNR(%) tends to remain constant or decrease in TOA as the initial query result size increases for all datasets as shown in Fig. 21(a)-Fig. 21(c). The FNR(%) tends to decrease in DT with few exceptions. Again, the initial query result size has no effect in TBA in terms of FNR(%) which is always 0%.

- The ACC(%) tends to increase in both TOA and DT as the initial query result size increases for all datasets as shown in Fig. 22(a)-Fig. 22(c) with few exceptions. Again, the initial query result size has no effect in TBA in terms of ACC(%) which is always 100%.

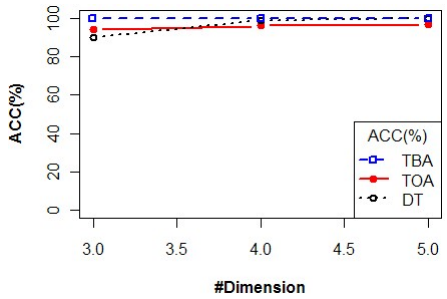
- The refined query complexities tend to increase for every algorithms in all datasets as we see in Fig. 23(a)- Fig. 23(c) if the initial query result size increases. This is because the number of boundary tuples tend to increase for TBA and TOA, and also the number of learned decision rules tend to increase for the DT-based query refinement if there is an increase in initial result size.



(a) DBLP

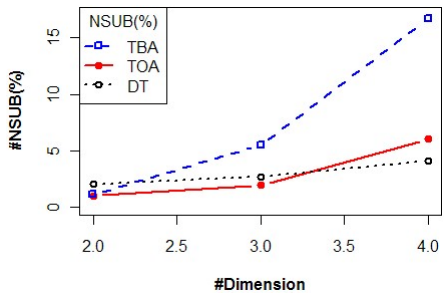


(b) Automobile

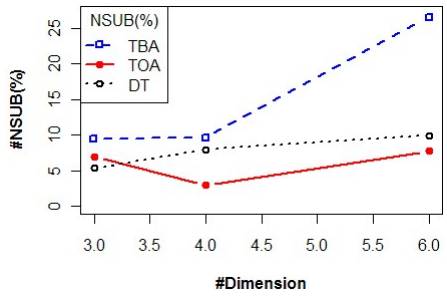


(c) Abalone

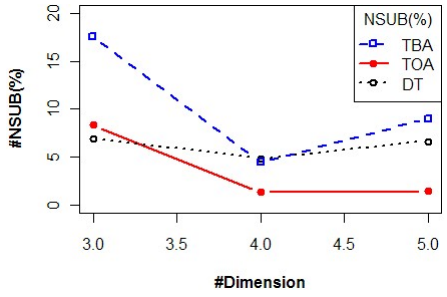
Figure 14: Effect of dimension size on ACC(%)



(a) DBLP



(b) Automobile



(c) Abalone

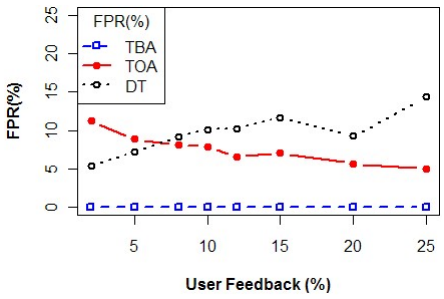
Figure 15: Effect of dimension size on #NSUB

6.2.3. *Statistical Significance*

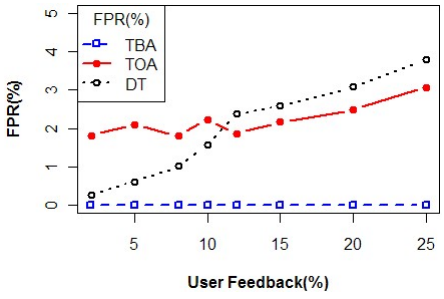
To show that the performance of the proposed *trade-off algorithm* (TOA) is also statistically significant (not by chance or on average only), we have performed 2-sample t-test (Blalock, 1972; David and Gunnink, 1997) on our achieved results. To do so, we have compared the outcome of TOA (i.e., FPR, FNR and ACC measures) for individual query input instances by setting δ to 0.90 for DBLP and Automobile datasets, and δ to 0.97 for Abalone dataset. Table 8 shows the p-values of 2-sample t-test performed between TOA and DT based query refinements for the tested query input instances of the three datasets used in our experiment. From Table 8, we see that TOA provides better accuracies in more than 99% input instances probabilistically, except for the query Q_{10} in the Abalone dataset for which the performance of TOA is insignificant in comparison with DT based query refinement. Table 9 shows the number of input instances (out of 100), for which we have achieved better performance for TOA (less FPR, less FNR and greater ACC) compared to the DT based query refinement. However, the performance of TOA in comparison with DT based query refinement depends on particular setting of δ as we have explained in Section 6.2.2, e.g., the probability of getting better accuracy is increased if δ tends to 1.0. The purpose of this test is to show that TOA is likely to perform better than DT based query refinement not only on average but also for individual input instances.

6.3. *Discussion*

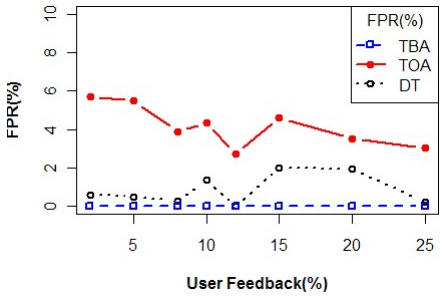
We observe that decision-tree (DT) based query refinement performed poorly, on average, compared to the refinement algorithms proposed in this paper. On may ask why does DT achieve this poor performance? Recall that DT is an information-gain-theory based linear classifier and its success relies heavily on the underlying data distribution as we explained it early in the paper. In contrast, the proposed query refinement algorithms (TBA and TOA) are data-driven and do not depend on any particular data distribution. This can be easily verified from the experiments presented in this paper. In addition, we can control the different quality metrics in TOA by setting δ to any value in the range 0.0 to 1.0 as required. In TOA, we sacrifice the quality of results (achieved by TBA) for attaining less complex refined queries. The fitness score defined in Section 5.2 tries to replace two or more boundary tuples by their combined version with respect to the particular setting of δ. The combination strategy developed in this paper tries to



(a) DBLP

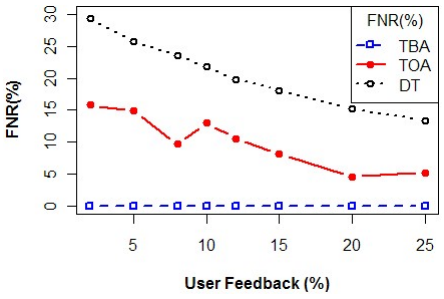


(b) Automobile

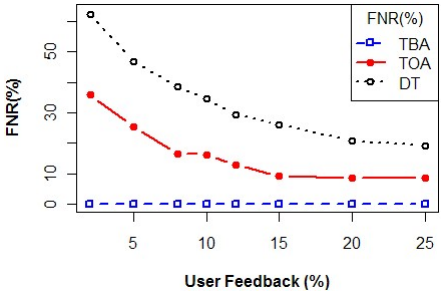


(c) Abalone

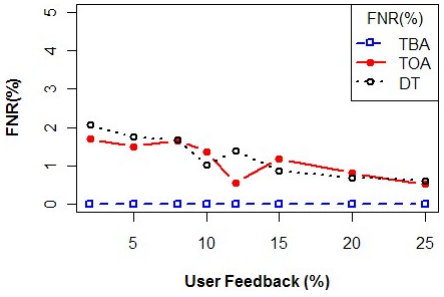
Figure 16: Effect of user feedback (%) on FPR (%)



(a) DBLP



(b) Automobile



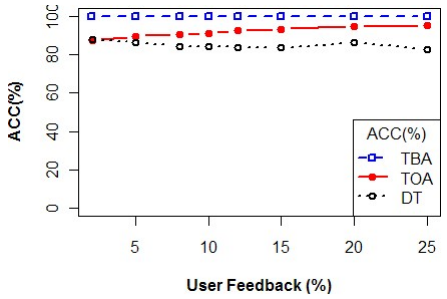
(c) Abalone

Figure 17: Effect of user feedback (%) on FNR (%)

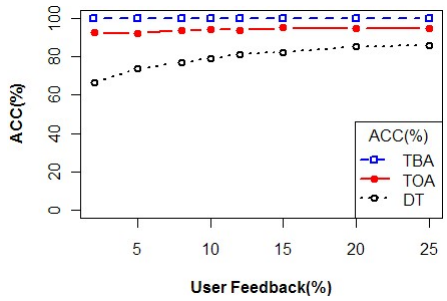
minimize the total number unexpected and expected tuples in the refined query output. This strategy does not show any biasness towards any particular set. However, the proposed framework can be extended and/or enhanced by redefining the fitness score (showing some biasness towards a particular set via sensitivity and specificity measures (Mitchell, 1997)) for controlling FPR (%) and FNR(%) in the refined query output.

7. Related Work

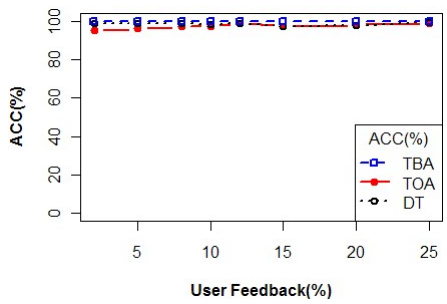
This work is inspired by the previous works found in (Huang et al., 2008), (Herschel and Hernández, 2010), (Tran and Chan, 2010), (He and Lo, 2012), (Ma et al., 2006), (Liu et al., 2010), (Koudas et al., 2006) and (Ma and Mehrotra, 2007). In (Huang et al., 2008), Huang et al. and in (Herschel and Hernández, 2010), Herschel et al. propose to modify the original tuple values in the database so that missing tuples become part of the query output. However, their techniques are not applicable in trusted data applications. In (Chapman and Jagadish, 2009) Chapman et al. propose to identify the culprit operator(s) that filters out expected tuple(s). As a next step, Tran and Chan (Tran and Chan, 2010) model query refinement by collecting missing tuples as feedback from the user. These authors exploit the idea of *skyline queries* to report the closest refined query wrt the original one to minimize the distance between the refined query and the original query. In the refined query, they also consider adding/dropping new predicates to/from the refined query. However, this approach can add/drop wrong predicates to/from the query and refined queries may introduce unexpected tuples in the result set. In (He and Lo, 2012), He et al. propose an approach to answer why-not questions (missing tuples) on top-k queries. This work also does not consider why (unexpected) tuples. In (Ma et al., 2006), Ma et al. model query refinement as both learning the structure of the query as well as learning the relative importance of query components. But they do not consider what new information a user expects to see (i.e. what is missing). In (Ma and Mehrotra, 2007), Ma et al. propose a framework that combines the positive aspects of both similarity retrieval and skyline retrieval into one single technique so that the user can retrieve results in the order of relevance. In (Liu et al., 2010), Liu *et al.* collect *false positives* (which we call unexpected information in this paper) which are identified by users to modify the initial rules in information extraction setting. Koudas et al. (Koudas et al., 2006) propose



(a) DBLP

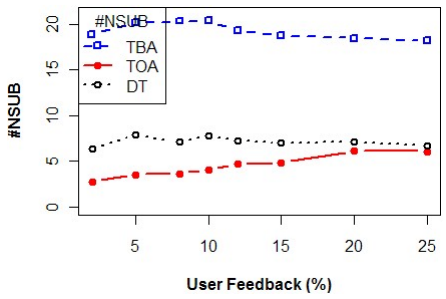


(b) Automobile

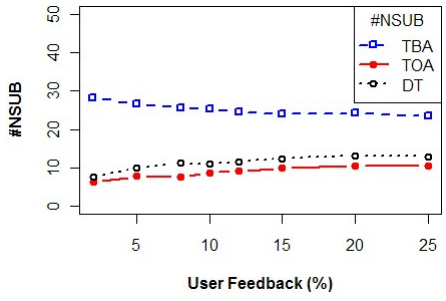


(c) Abalone

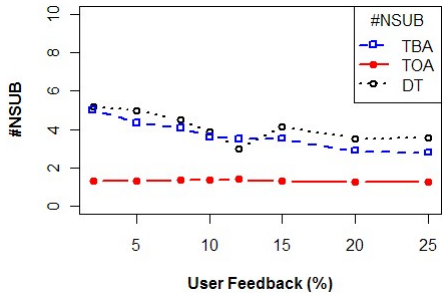
Figure 18: Effect of user feedback (%) on ACC(%)



(a) DBLP



(b) Automobile



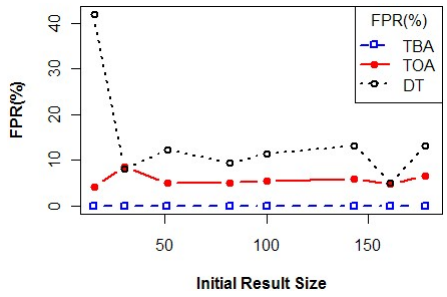
(c) Abalone

Figure 19: Effect of user feedback (%) on #NSUB

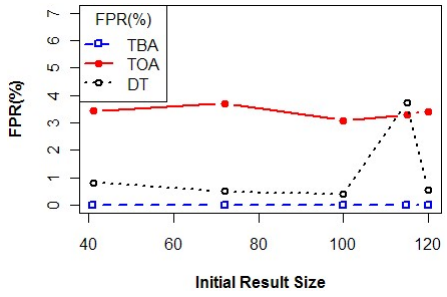
relaxation skyline as a solution for the empty answers problem. In a very recent work (Islam et al., 2013b), Islam et al. propose an approach for answering *why-not* questions in reverse skyline queries to start an automatic negotiation between customer and product of the company. The authors apply both data and query refinement techniques for producing meaningful answers.

None of the above models treats both unexpected and expected feedback. In (Islam et al., 2012a, 2013a), Islam et al. propose a decision tree based query refinement for minimizing unexpected information as well as maximizing expected information in the refined query output. The disadvantage of DT based query refinement is that it is based on information gain theory and relies heavily on underlying data distribution. In this paper, we propose *FlexIQ*, a framework for feedback based query refinement exploiting skyline operator. The advantage of *FlexIQ* is that it is independent of the underlying data distribution and the user can also trade-off different quality metrics in the refined queries. To the best of our knowledge, this is the first attempt ever made where skyline operator is exploited for controlling both unexpected and expected information in the refined query output. To do so, we rely on modifying the query conditions and their binding values only, rather than fixing weights to rerank expected answer objects for top-k queries (He and Lo, 2012).

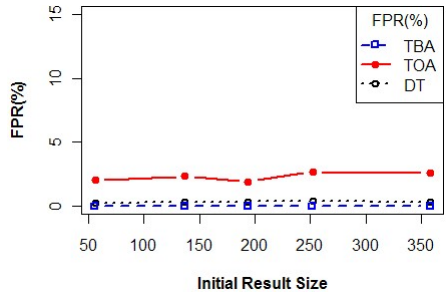
User feedback is also considered as first-class citizen in other areas of database systems such as semi-structured data (Cao et al., 2010), information integration (Belhajjame et al., 2011) and schema mappings (Belhajjame et al., 2010) and extensively studied in information retrieval techniques (Moon et al., 2010) including image (Hoi and Wu, 2011) and video retrieval (Vrochidis et al., 2010). In (Cao et al., 2010), Cao et al. show that feedback can be an effective tool for exploring large semi-structured data collections. Given a path query and a set of results identified by the system to this query over the data, the authors consider two types of feedback: Soft feedback (preference for some features over the others) and Hard feedback (whether certain features should be further enforced or are to be avoided). Finally, the authors describe a system for adaptive and exploratory path retrieval in semi-structured data. In (Belhajjame et al., 2011), Belhajjame et al. argue that user feedback should be considered and managed as a first class citizen in order to maximize the benefits in building information integration systems and present preliminary solutions for it. In (Belhajjame et al., 2010), the authors explore an approach for incrementally



(a) DBLP

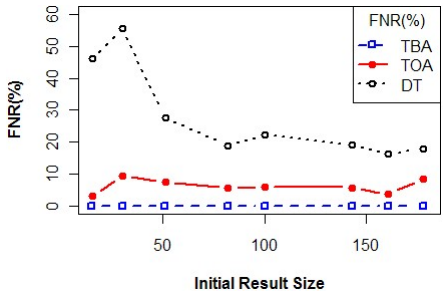


(b) Automobile

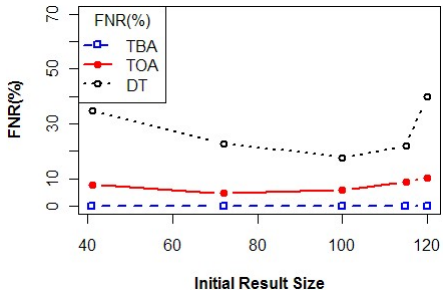


(c) Abalone

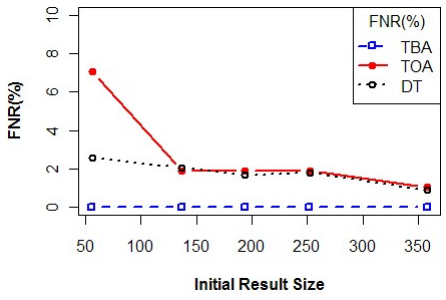
Figure 20: Effect of initial result size on FPR(%)



(a) DBLP



(b) Automobile



(c) Abalone

Figure 21: Effect of initial result size on FNR(%)

annotating schema mappings using feedback obtained from end users. In doing so, they do not require users to examine mapping specifications; rather, comment on results to queries evaluated using the mappings. In (Hoi and Wu, 2011), a relevance feedback technique is equipped to learn with users' feedback and thereby refine the image search results interactively. In (Vrochidis et al., 2010), the authors describe an approach to optimize query by visual example results, by combining visual features and implicit user feedback in interactive video retrieval. In this paper, we aim to develop a user feedback-based interactive SPJ querying framework in relational data setting, which exploits the skyline operator for minimizing the unexpected information as well as maximizing the expected information in the refined query output.

8. Conclusion

In this paper, we have presented a novel user feedback based query refinement framework, *FlexIQ*, which exploits the skyline operator. In our framework, we have used user feedback to discover the query intent of the user. In addition, the skyline operator is exploited to confine the search space of the query refinement algorithms and render the approach to be more intuitive to the user. The experimental results demonstrate that the proposed framework can effectively minimize the number of unexpected tuples as well as maximize the number of expected tuples in the refined query output, and outperform the naïve decision tree based query refinement. In *FlexIQ*, the user can also trade-off different quality metric such as quality of results and query complexity (i.e., number of subqueries) in the refined query, which is certainly a great advantage of *FlexIQ* in comparison with the decision tree based query refinement.

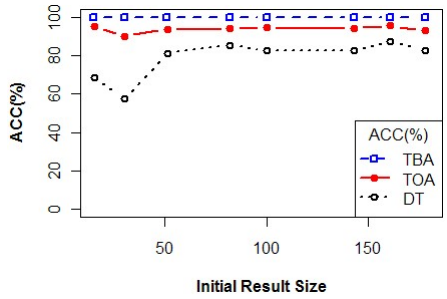
9. Acknowledgement

This work is supported by the grant of ARC Discovery Project No. DP120102627 and Project No. DP140103499.

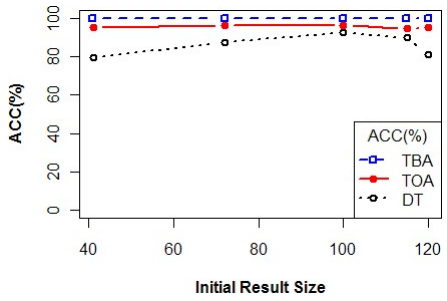
References

Amer-Yahia, S., Case, P., Rölleke, T., Shanmugasundaram, J., Weikum, G., 2005. Report on the db/ir panel at sigmod 2005. SIGMOD Record 34, 71–74.

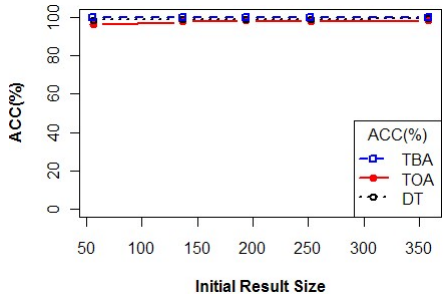
Belhajjame, K., Paton, N.W., Embury, S.M., Fernandes, A.A.A., Hedeler, C., 2010. Feedback-based annotation, selection and refinement of schema mappings for dataspaces, in: EDBT, pp. 573–584.



(a) DBLP

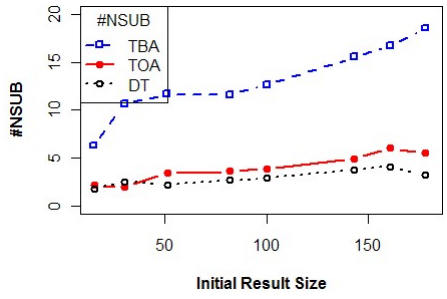


(b) Automobile

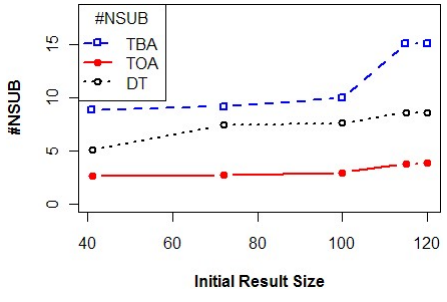


(c) Abalone

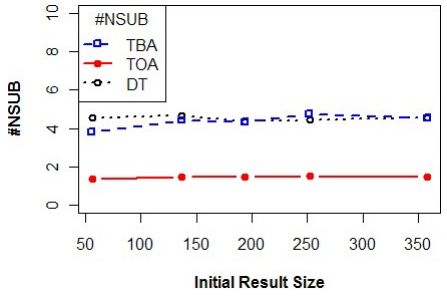
Figure 22: Effect of initial result size on ACC(%)



(a) DBLP



(b) Automobile



(c) Abalone

Figure 23: Effect of initial result size on #NSUB

Belhajjame, K., Paton, N.W., Fernandes, A.A.A., Hedeler, C., Embury, S.M., 2011. User feedback as a first class citizen in information integration systems, in: CIDR, pp. 175–183.

Blalock, H.M., 1972. Social statistics. New York: McGraw-Hill.

Börzsönyi, S., Kossmann, D., Stocker, K., 2001. The skyline operator, in: ICDE, pp. 421–430.

Cao, H., Qi, Y., Candan, K.S., Sapino, M.L., 2010. Feedback-driven result ranking and query refinement for exploring semi-structured data collections, in: EDBT, pp. 3–14.

Chapman, A., Jagadish, H.V., 2009. Why not?, in: SIGMOD Conference, pp. 523–534.

David, H.A., Gunnink, J.L., 1997. The paired t test under artificial pairing. The American Statistician 51, pp. 9–12.

Drosou, M., Pitoura, E., 2013. Ymaldb: exploring relational databases via result-driven recommendations. VLDB J. 22, 849–874.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H., 2009. The weka data mining software: an update. SIGKDD Explorations 11, 10–18.

He, Z., Lo, E., 2012. Answering why-not questions on top-k queries, in: ICDE, pp. 750–761.

Herschel, M., Hernández, M.A., 2010. Explaining missing answers to spjua queries. PVLDB 3, 185–196.

Hoi, S.C.H., Wu, P., 2011. Sire: a social image retrieval engine, in: ACM Multimedia, pp. 817–818.

Huang, J., Chen, T., Doan, A., Naughton, J.F., 2008. On the provenance of non-answers to queries over extracted data. PVLDB 1, 736–747.

Hyafil, L., Rivest, R.L., 1976. Constructing optimal binary decision trees is np-complete. Inf. Process. Lett. 5, 15–17.

Islam, M.S., Liu, C., Zhou, R., 2012a. On modeling query refinement by capturing user intent through feedback, in: Zhang, R., Zhang, Y. (Eds.), Australasian Database Conference, ACS, Melbourne, Australia. pp. 11–20.

Islam, M.S., Liu, C., Zhou, R., 2012b. User feedback based query refinement by exploiting skyline operator, in: ER, pp. 423–438.

Islam, M.S., Liu, C., Zhou, R., 2013a. A framework for query refinement with user feedback. Journal of Systems and Software 86, 1580–1595.

Islam, M.S., Zhou, R., Liu, C., 2013b. On answering why-not questions in reverse skyline queries, in: ICDE, pp. 973–984.

Jagadish, H.V., Chapman, A., Elkiss, A., Jayapandian, M., Li, Y., Nandi, A., Yu, C., 2007. Making database systems usable, in: SIGMOD Conference, pp. 13–24.

Kießling, W., 2002. Foundations of preferences in database systems, in: VLDB, pp. 311–322.

Koudas, N., Li, C., Tung, A.K.H., Vernica, R., 2006. Relaxing join and selection queries, in: VLDB, pp. 199–210.

Liu, B., Chiticariu, L., Chu, V., Jagadish, H.V., Reiss, F., 2010. Automatic rule refinement for information extraction. PVLDB 3, 588–597.

Ma, Y., Mehrotra, S., 2007. Integrating similarity retrieval and skyline exploration via relevance feedback, in: DASFAA, pp. 1045–1049.

Ma, Y., Mehrotra, S., Seid, D.Y., Zhong, Q., 2006. Raf: An activation framework for refining similarity queries using learning techniques, in: DASFAA, pp. 587–601.

Mitchell, T.M., 1997. Machine learning. McGraw Hill series in computer science, McGraw-Hill.

Moon, T., Li, L., Chu, W., Liao, C., Zheng, Z., Chang, Y., 2010. Online learning for recency search ranking using real-time user feedback, in: CIKM, pp.

Table 8: Statistical significance (p-value): TOA vs. DT

(a) DBLP dataset: $TOA_{\delta=0.90}$ vs. DT

Query	FPR	FNR	ACC
Q_1	7.857e-08	< 2.2e-16	< 2.2e-16
Q_2	1.61e-06	< 2.2e-16	< 2.2e-16
Q_3	6.416e-05	< 2.2e-16	< 2.2e-16
Q_4	1.361e-10	6.299e-12	< 2.2e-16
Q_5	4.65e-10	< 2.2e-16	< 2.2e-16
Q_6	3.698e-11	< 2.2e-16	< 2.2e-16
Q_7	2.048e-06	< 2.2e-16	< 2.2e-16
Q_8	2.363e-08	< 2.2e-16	< 2.2e-16
Q_9	0.002387	< 2.2e-16	< 2.2e-16
Q_{10}	1.022e-07	< 2.2e-16	< 8.01e-16
Q_{11}	9.68e-11	< 2.2e-16	< 2.2e-16
Q_{12}	7.857e-08	0.02531	< 2.2e-16

(b) Automobile dataset: $TOA_{\delta=0.90}$ vs. DT

Query	FPR	FNR	ACC
Q_1	0.4488	< 2.2e-16	< 2.2e-16
Q_2	0.5826	< 2.2e-16	< 2.2e-16
Q_3	0.9518	1.404e-12	< 2.2e-16
Q_4	0.0001841	5.175e-14	< 2.2e-16
Q_5	0.06356	< 2.2e-16	< 2.2e-16
Q_6	0.0001062	< 2.2e-16	< 2.2e-16
Q_7	8.657e-06	< 2.2e-16	< 2.2e-16
Q_8	0.9108	< 2.2e-16	< 2.2e-16
Q_9	0.01546	< 2.2e-16	< 2.2e-16
Q_{10}	0.1287	5.541e-11	3.168e-05

(c) Abalone dataset: $TOA_{\delta=0.97}$ vs. DT

Query	FPR	FNR	ACC
Q_1	0.09084	2.422e-07	2.151e-12
Q_2	0.1457	0.2712	3.735e-07
Q_3	0.2727	0.1724	9.241e-12
Q_4	0.7682	1.204e-10	1.013e-09
Q_5	0.5768	0.01948	0.0003673
Q_6	1.944e-06	0.01829	1.337e-11
Q_7	1.042e-15	< 2.2e-16	< 2.2e-16
Q_8	0.2435	0.004863	1.181e-06
Q_9	< 2.2e-16	7.359e-07	< 2.2e-16
Q_{10}	0.9482	0.9969	0.9988

Table 9: Better performance in # instances (out of 100): TOA vs. DT

(a) DBLP dataset: $TOA_{\delta=0.90}$ vs. DT

Query	FPR	FNR	ACC
Q_1	94	100	100
Q_2	86	100	100
Q_3	82	99	95
Q_4	89	90	95
Q_5	87	100	100
Q_6	88	99	100
Q_7	75	100	100
Q_8	87	92	97
Q_9	87	100	100
Q_{10}	93	93	94
Q_{11}	92	97	100
Q_{12}	100	100	100

(b) Automobile dataset: $TOA_{\delta=0.90}$ vs. DT

Query	FPR	FNR	ACC
Q_1	80	100	100
Q_2	85	98	98
Q_3	75	96	95
Q_4	77	92	94
Q_5	67	100	100
Q_6	100	100	100
Q_7	100	100	100
Q_8	81	100	98
Q_9	84	95	95
Q_{10}	78	83	81

(c) Abalone dataset: $TOA_{\delta=0.97}$ vs. DT

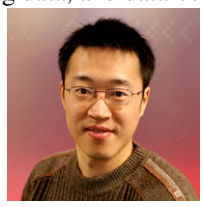
Query	FPR	FNR	ACC
Q_1	98	98	98
Q_2	89	90	89
Q_3	93	97	98
Q_4	88	93	93
Q_5	91	96	93
Q_6	100	94	95
Q_7	100	100	100
Q_8	90	85	87
Q_9	100	78	97
Q_{10}	80	74	65

1501–1504.
Su, I.F., Chung, Y.C., Lee, C., 2010. Top-*k* combinatorial skyline queries, in: DASFAA (2), pp. 79–93.
Tran, Q.T., Chan, C.Y., 2010. How to conquer why-not questions, in: SIGMOD Conference, pp. 15–26.
Voorneveld, M., 2003. Characterization of pareto dominance. Oper. Res. Lett. 31, 7–11.
Vrochidis, S., Kompatsiaris, I., Patras, I., 2010. Optimizing visual search with implicit user feedback in interactive video retrieval, in: CIVR, pp. 274–281.
Wong, R.C.W., Fu, A.W.C., Pei, J., Ho, Y.S., Wong, T., Liu, Y., 2008. Efficient skyline querying with variable user preferences on nominal attributes. Proc. VLDB Endow. 1, 1032–1043.

Dr. Md. Saiful Islam is a research fellow currently working at Swinburne University of Technology, Australia. He has finished his PhD in Computer Science and Software Engineering from Swinburne University of Technology, Australia in 2014. He has also received his BSc (Hons) and MS

degree in Computer Science and Engineering from University of Dhaka, Bangladesh, in 2005 and 2007, respectively. He has also been serving as a faculty member in the Institute of Information Technology, University of Dhaka since January 2008. He has published over 30 peer-reviewed papers in various journals and conference proceedings. His current research interests are in the areas of data management, information retrieval, machine learning and computer architecture.

Dr. Chengfei Liu is a Professor and the head of the Web and Data Engineering research group in the Faculty of Information and Communication Technologies, Swinburne University of Technology, Australia. He received the BS, MS and PhD degrees in Computer Science from Nanjing University, China in 1983, 1985 and 1988, respectively. Prior to joining Swinburne, he taught at the University of South Australia and the University of Technology Sydney, and was a Research Scientist at Cooperative Research Centre for Distributed Systems Technology, Australia. He also held visiting positions at the Chinese University of Hong Kong, the University of Aizu in Japan, and IBM Silicon Valley Lab in USA. He has published over 160 peer-reviewed papers in various journals and conference proceedings and has served on technical program committees and organizing committees of over 100 international conferences or workshops in the areas of database systems, Web information systems, and workflow systems. His current main research interests include keywords search on structured data, query processing and refinement for advanced database applications, query processing on uncertain data and big data, and data-centric workflows.



Dr. Rui Zhou is a research fellow in the Centre for Applied Informatics (CAI), College of Engineering & Science, Victoria University (VU). Before joining VU, he was a research fellow at Swinburne University of Technology, Australia (Swinburne) from 2010-2013. He obtained his PhD in Computer Science from Swinburne in 2010, and his MSc and BSc from Northeastern University, China in 2006 and 2004. His research interest is to design solid solutions in data management areas.