TIGER: Querying Large Tables through Criteria Extension

Yoann Pitarch[1], Dominique Laurent[2], Pascal Poncelet[1], Nicolas Spyratos[3]

(1) LIRMM - CNRS - Univ Montpellier 2, F-34392 Montpellier Cedex 5 - France
(2) ETIS - CNRS - Univ Cergy-Pontoise, F-95000 Cergy-Pontoise - France
(3) LRI - CNRS, Univ Paris 11, F-91405 Orsay - France

***Abstract*—Sales on the Internet have increased significantly during the last decade, and so, it is crucial for companies to retain customers on their web site. Among all strategies towards this goal, providing customers with a flexible search tool is a crucial issue.**

In this paper, we propose an approach, called TIGER, for handling such flexibility automatically. More precisely, if the search criteria of a given query to a relational table or a Web catalog are too restrictive, our approach computes a new query combining extensions of the criteria. This new query maximizes the quality of the answer, while being as close as possible to the original query. Experiments show that our approach improves the quality of queries, in the sense explained just above.

I. Introduction

Sales on the Internet have increased significantly during the last decade, and so, it is crucial for companies to retain customers on their web site. Among all strategies towards this goal, providing customers with a flexible search tool is a crucial issue. Additionally to provide recommendation systems ([AT05], [Bur02], [MSR04]), an important point is to ease search when customers are browsing a web site. To this end, the standard approach consists in providing customers with forms in which all search criteria are filled in.

For example, let us consider the case of a customer looking for a car to buy. Then, once logged on a car seller web site, several behaviors are possible. A first option is to scan all cars on sale. Although exhaustive, this option might not be realistic, due to the large number of cars on sale. A second option is focussing on the relevant cars, using criteria. Then, it might happen that the number of cars satisfying the criteria be so small that the user would like to change the search criteria (the worst situation being when no answer is given). In this case, defining less restrictive criteria amounts to issue a new query, but can result in the fact that the user gives up and visits a competing web site...

In this paper, we propose an approach, called TIGER[1], for handling search criteri on flexibility automatically. More precisely, if the search criteria of a given query to a relational table or a Web catalog are too restrictive, our approach computes a new query combining extensions of the criteria, so as to maximize the quality of the answer, while keeping the new query as close as possible to the original query. Experiments show that (i) our approach can be safely embedded in a web site, and that (ii) the quality of answers is improved.

Regarding related work, an early approach for query relaxation can be found in [GGM92], and in [CYC+96], the authors propose a relaxation system called CoBase, in which relaxations must be explicitly specified. More recently, approaches to flexibility have been proposed, and can be classified as *automatic* and *interactive*.

Automatic approaches ([JF03], [TC04], [JRMG06]) are generally based on "pseudo-relevance feedback", and as noticed in [Rut03], the quality of the new query is debatable because some

[1]TIGER means *TryIng to Get Extra Responses*.

terms might be irrelevant. In interactive approaches ([Fur85], [Ani03]), the user is asked to choose between automatically generated extensions.

The remainder of the paper is organized as follows: In Section II, we introduce an example to illustrate our approach. Basic concepts are introduced in Section III and Section IV deals with criterion and query extension. The TIGER approach is presented in Section V and Section VI reports on experiments. Section VII concludes the paper.

II. A Case Study

We consider a user wishing to buy a car, an example that will be used as a running example.

We assume that all cars on sale are stored in the table called `CARS`, shown in Table I and defined over attributes Id, Type, Color, Km and Price, whose meaning is clear from the context.

We consider the following query: *Give all cars of type Clio, of withe color, with less than 5000 km, and whose price is below 5000 euros.*

As the table `CARS` contains no row satisfying these criteria, it is expected that the user will enhance the criteria by issuing a new but more general query. We propose an approach for an automatic query rewriting method that generalizes a given query Q into a new one, as close as possible to Q, but whose answer becomes acceptable. To do so, we assume the following user preferences:

- The user really wants the price to be less than 5000 euros (as in the original query).
- The user prefers french cars to german cars, and then, to any other car.
- The user prefers white cars to black or gray cars, and then, to any other color.
- The user accepts to enhance the criteria, first on kilometers, then on the color or on the type.

Based on this knowledge and the original query, we look for another query taking user preferences into account and such that:

1) The criteria that cannot be changed remain unchanged (the price in our example).
2) For all criteria that can be changed, the change must be as limited as possible.
3) The new query returns more tuples than the original one.

Note: The information concerning user preferences, as well as which criteria (attributes) can change and which cannot will actually have to be elicited from the user or extracted from query logs. Moreover, when preference elicitation involves the user, a user friendly interface is indispensable. However, this topic lies outside the scope of the present paper.

In order to generate such a query, we define a score function that measures the "distance" between two queries Q and Q'.

Id	Type	Color	Km	Price
1	Clio	White	6000	5000
2	Polo	Gray	10000	4000
3	206	Gray	7000	5000
4	Golf	Black	6000	4500
5	Ibiza	Yellow	7000	4000
6	Ibiza	Red	4000	5000
7	Clio	Red	4000	5000
8	206	Black	6000	4500
9	Polo	Black	7000	5000
10	Polo	White	10000	4000
11	Ibiza	Red	10000	5000
12	Golf	White	7000	5000
13	206	White	6000	4500
14	Polo	Gray	4000	4000
15	Clio	Gray	7000	4000

TABLE I
SNAPSHOT OF TABLE `CARS`.

III. Basic Definitions

Let $\mathcal{T} = (A_1, \ldots, A_n)$ be a relational table defined over the attribute set $\{A_1, \ldots, A_n\}$, which we denote by $\mathcal{A}$. As usual, we assume that each attribute A_i $(i = 1, \ldots, n)$ is associated with a domain of values, denoted by $dom(A_i)$.

Definition 1. *A* criterion *c is an expression of the form $c = (A \in a)$ where A is an attribute of $\mathcal{T}$ and a is a subset of $dom(A)$. If A is a numeric attribute then a is an interval.*

The semantics *in $\mathcal{T}$ of a criterion $c = (A \in a)$, denoted by $Sem_{\mathcal{T}}(c)$ or by $Sem(c)$ when $\mathcal{T}$ is clear from the context, is the set of all tuples in $\mathcal{T}$ that satisfy c, that is the set: $\{t \mid (t \in \mathcal{T}) \wedge (t.A \in a)\}$.*

We note that if $c = (A \in a)$ is a criterion such that $|a| = 1$, then c corresponds to a selection condition $(A = \alpha)$, where α is the element of a.

Definition 2. *An attribute A of $\mathcal{T}$ is said to be* extensible *if there exists an ordering $<_A$ over $dom(A)$. The set of all extensible attributes of $\mathcal{T}$ is denoted by $\mathcal{A}_E$. A criterion $c = (A \in a)$ is said to be extensible if so is A.*

An attribute A of $\mathcal{T}$ is said to be fix *if A is not extensible. The set of all fix attributes of $\mathcal{T}$ is denoted by $\mathcal{A}_F$, and $c = (A \in a)$ is fix if so is A.*

Example 1. *In the context of our running example, assuming that attribute $Price$ is fix entails that the criterion $c = (Price \in [0; 5000])$ is fix as well. Notice that, in this case, no ordering on $dom(Price)$ is considered during the processing.*

On the other hand, attribute Km is considered extensible, using the natural ordering over numbers. Consequently, $c = (Km \in [0; 5000])$ is extensible.

Similarly, considering $Type$ and $Color$ as extensible attributes requires the definitions of orderings $<_{Type}$ and $<_{Color}$ over the corresponding domains.

Preferences given in Section II yield: $Ibiza <_{Type} \{Golf, Polo\} <_{Type} \{Clio, 206\}$, and $\{Red, Yellow\} <_{Color} \{Black, Gray\} <_{Color} White$.

Definition 3. *A* query *Q is a conjunction of the form $EC_Q \wedge FC_Q$ where EC_Q is a non empty conjunction of extensible criteria and FC_Q is a (possibly empty) conjunction of fix criteria.*

Every attribute of $\mathcal{T}$ occurs at most once in the criteria defining Q, and we denote by $\mathcal{E}_Q$ the set of all extensible attributes occurring in Q.

Every query Q is associated with its semantics in $\mathcal{T}$, denoted by $Sem_{\mathcal{T}}(Q)$ or $Sem(Q)$ when $\mathcal{T}$ is clear from the context, as follows: $Sem(Q)$ is the intersection of all $Sem(c)$, for every c in Q.

Definition 4. *Let $c = (A \in a)$ be an extensible criterion. An* extension *of c is a criterion c' of the form $c' = (A \in a')$ where $a \subseteq a'$.*

Let $Q = EC_Q \wedge FC_Q$ be a query. An extension *of Q is a query $Q' = EC_{Q'} \wedge FC_{Q'}$ such that $FC_Q = FC_{Q'}$ and every criterion of $EC_{Q'}$ is an extension of a criterion of EC_Q.*

Example 2. *In the context of our running example, the considered query is: $Q = EC_Q \wedge FC_Q$ where $EC_Q = (Type = Clio) \wedge (Km \in [0; 5000]) \wedge (Color = White)$ and $FC_Q = (Price \in [0; 5000])$. Referring to Table I, $Sem(Q) = \emptyset$.*

Moreover, $(Km \in [0; 7000])$ is an extension of $(Km \in [0; 5000])$ and $(Color \in \{Black, White\})$ is an extension of $(Color \in \{White\})$.

Thus $Q' = ((Type = Clio) \wedge (Km \in [0; 7000]) \wedge (Color \in \{Black, White\})) \wedge FC_Q$ is an extension of Q, and $Sem(Q')$ is not empty.

We notice that if c' is an extension of c, then $Sem(c) \subseteq Sem(c')$. Thus, if Q' is an extension of Q, then $Sem(Q) \subseteq Sem(Q')$. The problem addressed in this paper can be summarized as follows: Given a query Q, find an extension Q' of Q such that: (i) the extended criteria in Q' are as close as possible to those in Q, and (ii) $Sem(Q')$ is larger than $Sem(Q)$.

IV. Criterion and Query Extension

Every extensible attribute A is associated with a function $Rank_A$ such that, for all α, α' in $dom(A)$, if $\alpha' <_A \alpha$, then $Rank_A(\alpha) < Rank_A(\alpha')$.

Example 3. *In the example of Section II, Km is a numerical extensible attribute whose domain is assumed to be discretized through the intervals $[0; 4000[$, $[4000; 6000[$, $[6000; 7000[$, $[7000; 10,000[$ and $[10,000; \infty[$. Table II (top) depicts the associated function $Rank_{Km}$. On the other hand, the user preferences about colors are expressed on $dom(Color)$, according to the function $Rank_{Color}$ shown in Table II.*

Regarding preferences on types, we assume an ontology on car types, manufacturers and countries, so that the function $Rank_{Type}$ shown in Table II can be generated.

A. Rewriting Extensible Criteria

Definition 5. *Given an extensible attribute A and a value α in $dom(A)$, we call* α-extension *of α, denoted by $\rho_A(\alpha)$, the set $\rho_A(\alpha) = \{\alpha' \in dom(A) \mid Rank_A(\alpha) = Rank_A(\alpha')\}$.*

If a is a subset of $dom(A)$, the a-extension *of a, denoted by $\rho_A(a)$, is the set $\rho_A(a) = \bigcup_{\alpha \in a} \rho_A(\alpha)$.*

The ρ-extension of $c = (A \in a)$, denoted by $\rho(c)$, is defined by $\rho(c) = (A \in \rho_A(a))$.

x	$Rank_{Km}(x)$
$[0; 4000[$	0
$[4000; 6000[$	1
$[6000; 7000[$	2
$[7000; 10,000[$	3
$[10,000; \infty[$	4

x	$Rank_{Color}(x)$
$White$	0
$\{Black, Gray\}$	1
$\{Yellow, Red\}$	2

x	$Rank_{Type}(x)$
$\{Clio, 206\}$	0
$\{Polo, Golf\}$	1
$Ibiza$	2

TABLE II
THE FONCTIONS $Rank_{Km}$, $Rank_{Color}$ AND $Rank_{Type}$

Referring back to Example 3, $\rho_{Color}(Black) = \{Black, Gray\}$, since $Rank_{Color}(Black) = Rank_{Color}(Gray) = 1$. Thus, $\rho(Color \in \{Black\}) = (Color \in \{Black, Gray\})$.

Notice that the set $\{\rho_A(\alpha) \mid \alpha \in dom(A)\}$ is the partition of $dom(A)$ induced by $Rank_A$. Thus, for all α and α' such that $Rank_A(\alpha) \neq Rank_A(\alpha')$, $\rho_A(\alpha) \cap \rho_A(\alpha') = \emptyset$. Moreover, Definition 5 shows that the ρ-extension of c is an extension of c.

B. Distance between Queries

Definition 6. *Let $c = (A \in a)$ and $c' = (A \in a')$ be two criteria over A. The* distance *between c and c', denoted $\delta(c, c')$, is defined as follows:*

- *If $\rho_A(a) \cap \rho_A(a') \neq \emptyset$ then $\delta(c, c') = 0$.*
- *Otherwise, $\delta(c, c') = \min\{Rank_A(\alpha') - Rank_A(\alpha) \mid \alpha \in A \wedge \alpha' \in a' \wedge Rank_A(\alpha) < Rank_A(\alpha')\}$.*

Although the function δ defined above is *not* a distance function (because $\delta(c, c') = 0$ with $c \neq c'$ is possible) we use the term "distance" because of intuition. Moreover, Definition 6 implies that: (i) for all $c = (A \in a)$ and $c' = (A \in a')$, $\delta(c, c') = \delta(\rho(c), \rho(c'))$, and (ii) $\delta(c, \rho(c)) = 0$, for every criterion c.

The function δ is extended from criteria to queries as follows: let Q and Q' be queries such that EC_Q and $EC_{Q'}$ consist respectively of m extensible criteria $c_1, \ldots, c_m$ and $c'_1, \ldots, c'_m$, defined over the same attributes $A_1, \ldots, A_m$, we have:

$$\delta(Q, Q') = \sum_{j=1}^{m} \delta(c_j, c'_j).$$

Since user preferences may involve the attributes themselves, we assume that the extensible attributes of Q are partially ordered through a relation, denoted by $\prec$. We associate every extensible attribute A with a positive number w_A, called the *weight of* A, and such that $A \prec A'$ implies $w_A < w_{A'}$.

Definition 7. *Let $Q = EC_Q \wedge FC_Q$ and $Q' = EC_{Q'} \wedge FC_Q$ be such that EC_Q and $EC_{Q'}$ consist respectively of m extensible criteria $c_1, \ldots, c_m$ and $c'_1, \ldots, c'_m$, defined over the same attributes $A_1, \ldots, A_m$. The* weighted distance *between Q and Q', denoted by $\delta_w(Q, Q')$, is defined by:*

$$\delta_w(Q, Q') = \sum_{j=1}^{m} w_{A_j} \times \delta(c_j, c'_j).$$

Example 4. *In the context of our running example, we recall that we consider the query Q where $EC_Q = (Type \in \{Clio\}) \wedge (Color \in \{White\}) \wedge (Km \in [0; 5000])$ and $FC_Q = (Price \in [0; 5000])$, and that $Q' = ((Type \in \{Clio\}) \wedge (Color \in \{Black, White\}) \wedge (Km \in [0; 7000])) \wedge FC_Q$ is an extension of Q.*

For extending Q into Q', we consider the criteria $c_1 = (Color \in \{Black\})$ and $c_2 = (Km \in \{7000\})$, for which Table II shows that $\rho(c_1) = (Color \in \{Black, Gray\})$ and $\rho(c_2) = (Km \in [7000; 10,000[)$, and $\delta((Color \in \{White\}), c_1) = 1$ and $\delta((Km \in [0; 5000]), c_2) = 2$.

Therefore, denoting by Q'' the query $Q'' = ((Type \in \{Clio\}) \wedge (Color \in \{Black\}) \wedge (Km \in [7000; 10,000[)) \wedge FC_Q$, $\delta(Q, Q'') = 0+1+2 = 3$.

As the user prefers to extend the kilometer criterion over the type and the color criteria, we have $Km \prec Type$ and $Km \prec Color$. Assuming $w_{Km} = 1$ and $w_{Type} = w_{Color} = 3$, we obtain $\delta_w(Q, Q'') = 3 \times 0 + 3 \times 1 + 1 \times 2 = 5$.

V. THE TIGER APPROACH

A. Basic Properties

Let Q be a query with extensible criteria $c_j = (A_j \in a_j)$ $(j = 1, \ldots, m)$, and t a tuple in $\mathcal{T}$ such that $t.(\mathcal{E}_Q) = (\alpha_1, \ldots \alpha_m)$. We denote by $Q(t)$ the query obtained from Q by replacing every extensible criterion c_j of EC_Q by $(A_j \in \rho_{A_j}(\alpha_j))$.

Moreover, we associate t with the tuple $\delta_Q(t) = (\delta_1^t, \ldots, \delta_m^t)$ such that, for every $j = 1, \ldots, m$, $\delta_j^t = \delta(c_j, (A_j \in \rho_{A_j}(\alpha_j)))$, and we denote by $\delta_Q(\mathcal{T})$ the set $\{\delta_Q(t) \mid t \in \mathcal{T}\}$. For every ν in $\delta_Q(\mathcal{T})$, we denote by $\tau_Q(\nu)$ the number of tuples t in $\mathcal{T}$ such that $\delta_Q(t) = \nu$.

We notice that if t and t' are tuples in $\mathcal{T}$ such that $\delta_Q(t) = \delta_Q(t') = \nu$, then $Q(t) = Q(t')$. Therefore, we denote this query by $Q(\nu)$.

Example 5. *In the context of our running example, consider again the query Q such that $FC_Q = (Price \in [0; 5000])$ and $EC_Q = (Type \in \{Clio\}) \wedge (Color \in \{White\}) \wedge (Km \in [0; 5000])$.*

Let t_3 be the tuple of $\mathcal{T}$ with identifier 3 in Table I. As $\rho_{Type}(Clio) = \{Clio, 206\}$, $\rho_{Color}(Gray) = \{Gray, Black\}$ and $\rho_{Km}(7000) = [7000; 10,000[$, we have $Q(t_3) = (Price \in [0; 5000]) \wedge (Type \in \{Clio, 206\}) \wedge (Color \in \{Gray, Black\}) \wedge (Km \in [7000; 10,000[)$.

Moreover, it can be seen from Table II that $\delta_Q(t_3) = (0, 1, 2)$, and that the last tuple of $\mathcal{T}$ in Table I, which we call t_{15}, is also such that $\delta_Q(t_{15}) = (0, 1, 2)$. In fact, it can be checked that $\tau_Q(0, 1, 2) = 2$, since t_3 and t_{15} are the only tuples of $\mathcal{T}$ corresponding to $(0, 1, 2)$.

Based on the notation just introduced, we have the following proposition.

Proposition 1. *Let Q be a query with extensible criteria $c_j = (A_j \in a_j)$ for $j = 1, \ldots, m$.*

1) *If ν and ν' are distinct tuples in $\delta_Q(\mathcal{T})$, then: $Sem(Q(\nu)) \cap Sem(Q(\nu')) = \emptyset$.*
2) *For every ν in $\delta_Q(\mathcal{T})$, $|Sem(Q(\nu))| = \tau_Q(\nu)$.*

Now, given a positive integer m, we consider the following ordering $\leq_m$ over tuples in the cartesian product $\mathbb{N}^m$: for all $(n_1, \ldots, n_m)$ and $(n'_1, \ldots, n'_m)$ in $\mathbb{N}^m$, $(n_1, \ldots, n_m) \leq_m (n'_1, \ldots, n'_m)$ if for every $j = 1, \ldots, m$, $n_j \leq n'_j$.

Given a tuple ν in $\delta_Q(\mathcal{T})$, we denote by $I(\nu)$ the set $I(\nu) = \{\nu' \in \delta_Q(\mathcal{T}) \mid \nu' \leq_m \nu\}$.

Denoting for every $j = 1, \ldots, m$ by $\rho_j(\nu_j)$ the ρ-extension of any α_j such that $Rank_{A_j}(\alpha_j) = \nu_j$, the ν-extension of Q is defined as follows.

Definition 8. *Given a query Q such that $EC_Q = ((A_1 \in a_1) \wedge \ldots \wedge (A_m \in a_m))$ and a tuple ν in $\delta_Q(\mathcal{T})$, the* ν-extension *of Q, denoted by $\overline{Q}(\nu)$, is the query $\overline{Q}(\nu) = EC_{\overline{Q}(t)} \wedge FC_{\overline{Q}(t)}$ such that*

- *$FC_{\overline{Q}(\nu)} = FC_Q$, and*
- *$EC_{\overline{Q}(\nu)}$ is the conjunction of the criteria $\overline{c}_j = (A_j \in \overline{a}_j)$ where, for every $j = 1, \ldots, m$, $\overline{a}_j = \rho_j(a_j) \cup \left(\bigcup_{\nu' \in I(\nu)} \rho_j(\nu'_j)\right)$.*

As for every $j = 1, \ldots, m$, $a_j \subseteq \rho_j(a_j) \subseteq \overline{a}_j$, by Definition 4, $\overline{Q}(\nu)$ is an extension of Q.

Example 6. *In the context of our running example, referring back to the query Q of Example 5, the first three columns of Table III show the content of $\delta_Q(\mathcal{T})$, and the fourth column contains the corresponding values of $\tau_Q(\nu)$. For example, the seventh row means that $\mathcal{T}$ contains two tuples t and t' such that $\delta_Q(t) = \delta_Q(t') = (0, 1, 2)$ (the last column of Table III will be explained later). In fact, t and t' are the tuples called t_3 and t_{15} in Example 5. Consequently, Proposition 1(2) implies that $|Sem(Q(0, 1, 2)| = 2$.*

For $\nu = (0, 1, 2)$, Table III shows that $I(\nu) = \{\nu_1, \nu_2, \nu\}$, where $\nu_1 = (0, 0, 1)$ and $\nu_2 = (0, 1, 1)$. Thus, by Proposition 1(2), $|Sem(Q(\nu)| = |Sem(Q(\nu_1))| = 2$ and $|Sem(Q(\nu_2))| = 1$.

Moreover, $\overline{Q}(\nu) = (Price \in [0; 5000]) \wedge (Type \in \{Clio, 206\}) \wedge (Color \in \{White, Gray, Black\}) \wedge (Km \in [0; 10,000[)$.

δ_{Type}	δ_{Color}	δ_{Km}	$\tau_Q(\nu)$	$Score$
0	0	1	2	1
1	1	0	1	0,5
0	2	0	1	0,5
0	1	1	1	1,5
1	0	2	1	1
1	1	1	1	$\frac{2}{3}$
0	**1**	**2**	**2**	$\frac{\mathbf{10}}{\mathbf{3}}$
1	1	2	1	$\frac{5}{2}$
2	2	0	1	0,5
1	0	3	1	$\frac{3}{4}$
1	1	3	1	$\frac{11}{5}$
2	2	2	1	$\frac{13}{6}$
2	2	3	1	$\frac{15}{7}$

TABLE III
THE TABLE $\delta_Q(\mathcal{T})$ AND THE ASSOCIATED SCORES

The following proposition shows how to compute

the cardinality of the semantics of ν-extensions.

Proposition 2. *Using the notation previously introduced, for every ν in $\delta_Q(\mathcal{T})$, we have*

$$|Sem(\overline{Q}(\nu))| = \sum_{\nu' \in I(\nu)} \tau_Q(\nu').$$

Referring back to Example 6 and applying Proposition 2, we have $|Sem(\overline{Q}(\nu))| = |Sem(Q(\nu))| + |Sem(Q(\nu_1))| + |Sem(Q(\nu_2))| = 5$.

B. The Score Function

Given a query $Q = EC_Q \wedge FC_Q$ and assuming that the preferences over the extensible attributes in $\mathcal{E}_Q$ are known, our approach works as follows:

1) Discard from $\mathcal{T}$ all tuples that do not satisfy the fix criteria in FC_Q; let $\mathcal{T}_{FC_Q}$ be the resulting table. If $\mathcal{T}_{FC_Q} = \emptyset$ then stop, otherwise, proceed to the next three steps.
2) Compute the tuples in $\delta_Q(\mathcal{T}_{FC_Q})$ along with the corresponding numbers $\tau_Q(\nu)$.
3) Find the "best" ν-extensions of Q, for all ν in $\delta_Q(\mathcal{T}_{FC_Q})$.
4) In case more than one query is returned by the previous step, choose the ν-extensions of Q having the least weighted distance to Q. Although no further criterion is set if several queries satisfy this last requirement, we assume that only one query is returned.

The ν-extension computed according the steps above is called the *best* ν-extension of Q and is denoted by $\overline{Q}_{best}$.

Regarding Step 3, we recall that the "best" ν-extension of Q must improve the quality of the answer to Q, *i.e.,* have a semantics significantly greater than that of Q, while being as "close" as possible to Q. In order to combine these two requirements, we define the following score function.

Definition 9. *Let Q be a query. For every ν in $\delta_Q(\mathcal{T}_{FC_Q})$, the* score *of the ν-extension of Q, $\overline{Q}(\nu)$, denoted by $Score_Q(\nu)$, is defined by: If $\delta(Q, Q(\nu)) = 0$ then $Score_Q(\nu) = \infty$, otherwise*

$$Score_Q(\nu) = \frac{|Sem(\overline{Q}(\nu))|}{\delta(Q,Q(\nu))/|Sem(Q(\nu))|}.$$

Clearly, in Definition 9, $Score_Q(\nu)$ is not defined when $|Sem(Q(\nu))| = 0$. This happens if $\mathcal{T}_{FC_Q}$ is empty, in which case any extension of Q has also an empty semantics. However, assuming that $\mathcal{T}_{FC_Q} \neq \emptyset$ implies that $|Sem(Q(\nu))| \neq 0$.

On the other hand, if $\delta(Q, Q(\nu)) = 0$ then $\nu = (0, \ldots, 0)$. Thus, by definition of $Q(\nu)$, all extended criteria of $Q(\nu)$ are the ρ-extensions of those of Q. Therefore, in this case, $Q(\nu)$ is the least ν-extension of Q that we can get, and this explains why the score is set to ∞. Notice that if $Score_Q(\nu) = \infty$ then $\overline{Q}_{best} = Q(\nu)$.

It is important to note that the score function of Definition 9 fits our requirements because the more tuples returned by $\overline{Q}(\nu)$ and the closer to Q is $Q(\nu)$, the higher the score. Moreover, by dividing by $\delta(Q, Q(\nu))/|Sem(Q(\nu))|$, instead of simply $\delta(Q, Q(\nu))$, allows to take explicitly into account the number of tuples satisfied by the criteria defining the considered ν-extension.

Regarding computational aspects, using the definition of δ, Proposition 1 and Proposition 2, $Score_Q(\nu)$ can be written as

$$Score_Q(\nu) = \frac{\sum_{\nu' \in I(\nu)} \tau_Q(\nu')}{(\sum_{j=1}^{m} \nu_i)/\tau_Q(\nu)}.$$

Consequently, assuming that $\delta_Q(\mathcal{T}_{FC_Q})$ along with the corresponding $\tau_Q(\nu)$ have been computed, determining the scores of all ν-extensions of Q does *not* require any access to the table $\mathcal{T}$.

Example 7. *In the context of our running example, all scores of all ν-extensions of Q are shown in the last column of Table III. It can be seen that the seventh row of this table has the highest score.*

This score is computed as follows: since $\nu = (0, 1, 2)$, we have $|Sem(Q(\nu))| = 3$ and we have seen that $|Sem(\overline{Q}(\nu))| = 5$. Since Table III shows that $\tau_Q(\nu) = 2$, we have $Score_Q(\nu) = \frac{5}{3/2} = \frac{10}{3}$.

As seen in Example 6, $\overline{Q}(\nu) = (Price \in [0; 5000]) \wedge (Type \in \{Clio, 206\}) \wedge (Color \in \{White, Gray, Black\}) \wedge (Km \in [0; 10,000[)$.

Since $\overline{Q}(\nu)$ is the only ν-extension having the highest score $\frac{10}{3}$, this query is the query $\overline{Q}_{best}$ proposed to the user, and it is easy to see that its semantics is not empty, contrary to Q.

VI. Implementation and Experiments

Our approach has been implemented in JAVA 1.6 on a MacBook computer equipped with a 2.4 GHz

Intel Core 2 Duo processor and 2 Go RAM.

The experiments reported in this section have been run on synthetic data sets whose characteristics are stated as $\mathtt{EeRrTtk}$, meaning that the data set consists of $t \times 10^3$ tuples, and that the query contains e extensible attributes, for which the $Rank$ functions have r values. Two distinct types of data sets have been generated:

- Randomly and uniformly generated data, to assess our approach against uncorrelated data.
- Biased generated data sets, to assess our approach in more realistic situations where most data values are close to each other.

In the latter case, data sets are generated so as, if $(A \in a)$ is an extensible criterion, 80% of values α over A are such that $0 \leq \delta((A \in \rho_A(\alpha)), (A \in a)) \leq M/3$, where M is the maximum value of $Rank_A$. For example, for $\mathtt{E10R20T10k}$, 80% of the generated A-values α are such that $0 \leq \delta((A \in \rho_A(\alpha)), (A \in a)) \leq 6$.

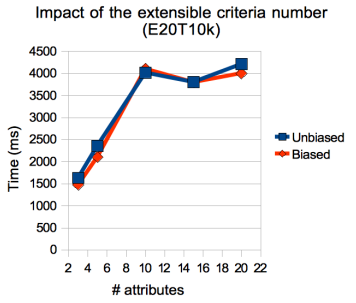
The following three parameters have been first considered in our experiments: the number of extensible attributes, the number of possible values of the $Rank$ functions and the number of tuples satisfying the fix criteria. Figure 1 shows separately the impacts on runtime of these parameters. We first notice that the fact that the data are biased or not does not significantly change runtime.

Regarding the impacts of the number of extensible attributes and of the number of possible $Rank$ values, Figures 1 (a) and (b) show that the best extension $\overline{Q}_{best}$ is computed in about 4 seconds, which is an acceptable runtime, considering that the table $\mathcal{T}_{FC_Q}$ contains $10,000$ tuples.

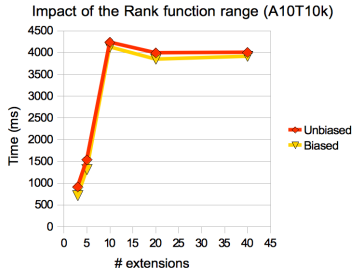
On the other hand, Figure 1 (c) clearly shows that the number of tuples in $\mathcal{T}_{FC_Q}$ has a significant impact on runtime. This is due to the fact that when the number of tuples in $\mathcal{T}_{FC_Q}$ increases, then so does the number of tuples in $\delta(\mathcal{T}_{FC_Q})$. Consequently, the number of queries $Q(\nu)$ to be processed for the computation of $\overline{Q}_{best}$ also increases accordingly. We note that, for $50,000$ tuples in $\mathcal{T}_{FC_Q}$, the runtime is below 12 seconds, an acceptable increase in runtime, considering that we have 10 extensible attributes with 20 possible $Rank$ values.

Then, the quality of the obtained ν-extensions has been assessed according to the number of extensible attributes and the number of $Rank$ values. Moreover, according to Definition 9, $|Sem(\overline{Q}_{best})|$ and $\delta(Q, \overline{Q}_{best})$ are the values that have to be considered in this respect. Whereas the distance between Q and $\overline{Q}_{best}$ is easy to assess, measuring the quality of the computed ν-extension $\overline{Q}_{best}$ in terms of the additional returned tuples is not trivial.

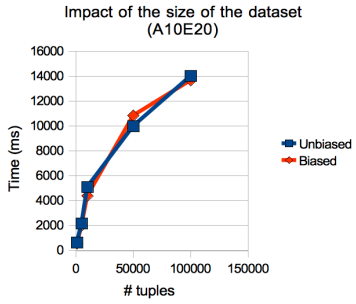
A first way could be computing the gain in tuples of $Sem(\overline{Q}_{best})$ with respect to $Sem(Q)$. However,



(a) Number of attributes



(b) Number of extensions



(c) Number of tuples

Fig. 1. Quantitative experiment results

e	$Gain$	$\delta(Q, \overline{Q}_{best})$	$\delta(Q, \overline{Q}_{max})$
3	68	1	25
5	71	10	75
10	53	19	190
15	21	50	285
20	26	61	380

r	$Gain$	$\delta(Q, \overline{Q}_{best})$	$\delta(Q, \overline{Q}_{max})$
3	63	1	20
5	65	1	40
10	50	12	90
20	23	70	190
40	19	90	390

TABLE IV
QUALITATIVE EXPERIMENT RESULTS

this gain is not relevant because, if $Sem(Q) = \emptyset$, it is equal to 100% for *any* extension of Q.

We rather measure quality improvement in terms of the difference between $|Sem(\overline{Q}_{best})|$ and the semantics of the maximal ν-extension that can be considered, based on the content of $\delta_Q(\mathcal{T}_{FC_Q})$. Intuitively, denoting by $\overline{Q}_{max}$ this particular ν-extension of Q, $Sem(\overline{Q}_{max})$ is the largest set of tuples that can be obtained from $\mathcal{T}$ by extending the criteria in Q, based on the user preferences.

Thus, we call *gain* of $\overline{Q}_{best}$, denoted by $Gain(\overline{Q}_{best})$ the number $\frac{|Sem(\overline{Q}_{best})| \times 100}{|Sem(\overline{Q}_{max})|}$.

We point out that, although providing the user with the query $\overline{Q}_{max}$ is computationally easier than computing $\overline{Q}_{best}$, this solution is not satisfactory, due to the large size of the semantics of $\overline{Q}_{max}$. Taking into account the distance between queries in the score function avoids to consider such queries.

The experiments were conducted with biased data sets `EeRrT10k`, where e, respectively r, ranges from 3 to 20, respectively from 3 to 40.

The results of these experiments, displayed in Table IV, show that for e and r less than or equal to 10, the gain is above 50%, while for larger values, the gain decreases significantly, but still, is not below 20% (except for $r = 40$, in which case the maximal distance is about 400).

It is important to note from Table IV that all distances $\delta(Q, \overline{Q}_{best})$ are much less than the distances $\delta(Q, \overline{Q}_{max})$, showing that $\overline{Q}_{best}$ is close to the original query Q, as compared with $\overline{Q}_{max}$.

VII. CONCLUSION

In this paper, we have presented an approach for handling extensible criteria in order to enhance the answer to a given query. Criterion extension makes use of user preferences, seen as partial orderings over attribute domains, in order to compute the "best" extension.

Based on this work, we are investigating the following issues: (i) testing our approach against real data, (ii) considering a static knowledge base such as WordNet [F$^+$98] to improve the quality of the extended query, (iii) investigating the coupling of our approach with recommendation systems, and (iv) considering *fuzzy* criteria.

REFERENCES

[Ani03] P. Anick. Using terminological feedback for web search refinement: a log-based study. In *Proc. of ACM SIGIR*, pages 88–95, 2003.

[AT05] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TKDE*, 17(6):734–749, 2005.

[Bur02] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[CYC$^+$96] W.W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, and C. Larson. Cobase: A scalable and extensible cooperative information system. *Journal of Intelligent Information Systems*, 6(2):223–259, 1996.

[F$^+$98] C. Fellbaum et al. *WordNet: An electronic lexical database*. MIT press Cambridge, MA, 1998.

[Fur85] G. W Furnas. Experience with an adaptive indexing scheme. In *Proc. of SIGCHI*, pages 131–135, 1985.

[GGM92] T. Gaasterland, P. Godfrey, and J. Minker. Relaxation as a platform for cooperative answering. *Journal of Intelligent Information Systems*, 1(3):293–321, 1992.

[JF03] R. Jones and D. C. Fain. Query word deletion prediction. In *Proc. of ACM SIGIR*, pages 435–436, 2003.

[JRMG06] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *Proc. of intl conf. on World Wide Web*, pages 387–396. ACM, 2006.

[MSR04] S. E Middleton, N. R Shadbolt, and D. C De Roure. Ontological user profiling in recommender systems. *ACM TOIS*, 22(1):54–88, 2004.

[Rut03] I. Ruthven. Re-examining the potential effectiveness of interactive query expansion. In *Proc. of ACM SIGIR*, pages 213–220, 2003.

[TC04] E. Terra and C. L.A. Clarke. Scoring missing terms in information retrieval tasks. In *Proc. of ACM CIKM*, pages 50–58, 2004.