Personalized Queries under a Generalized Preference Model (*)

Georgia Koutrika Yannis Ioannidis
University of Athens, Hellas
{koutrika, yannis}@di.uoa.gr

Abstract

Query Personalization is the process of dynamically enhancing a query with related user preferences stored in a user profile with the aim of providing personalized answers. The underlying idea is that different users may find different things relevant to a search due to different preferences. Essential ingredients of query personalization are: (a) a model for representing and storing preferences in user profiles, and (b) algorithms for the generation of personalized answers using stored preferences. Modeling the plethora of preference types is a challenge. In this paper, we present a preference model that combines expressivity and concision. In addition, we provide efficient algorithms for the selection of preferences related to a query, and an algorithm for the progressive generation of personalized results, which are ranked based on user interest. Several classes of ranking functions are provided for this purpose. We present results of experiments both synthetic and with real users (a) demonstrating the efficiency of our algorithms, (b) showing the benefits of query personalization, and (c) providing insight as to the appropriateness of the proposed ranking functions.

1. Introduction

A user accessing an information system with the intention of satisfying an information need, may have to reformulate the query issued several times and sift through many results until a satisfactory, if any, answer is obtained. This is a very common experience especially for Web searchers, due to information abundance and users' heterogeneity in the Web. A critical observation is that "*different users may find different things relevant when searching*" because of different preferences, goals etc. [20]. Thus, they may expect different answers to the same query. Consider a simple case: two users, Al and Julie, access a web-based movies database both searching for comedies. Al is a fan of director W. Allen, while Julie is not. Most systems would consider only the request issued and return to both users the same, exhaustive list of comedies. However, storing user preferences in user profiles gives a system the opportunity to return more focused, personalized (and hopefully smaller) answers.

Query Personalization is the process of dynamically enhancing a query with related user preferences stored in a user profile with the purpose of providing personalized answers. Focusing on the user enables a shift from what is called 'consensus relevancy' where the computed relevancy for the entire population is presumed relevant for each user, toward 'personal relevancy' where relevancy is computed based on each individual's characteristics [20]. Personalized results for Al would include W. Allen's comedies, while personalized results for Julie would not. Which preferences are related to a request and how these affect the final answer are dynamically determined based on the query, the profile and the personalization logic applied.

Query personalization approaches have recently attracted interest in both IR and Databases research communities [16, 20, 18]. This paper is concerned with query personalization in the context of databases. We adopt the query personalization framework presented in our earlier work [16]. Based on that, given a query and a profile, a personalized answer is built by specifying: *(a)* the number K of top preferences from the user profile that should affect it, and *(b)* the number L ($L \leq K$) of those preferences that should at least be satisfied. Parameters K and L can be specified directly by the user or derived based on various criteria on the query context, such as user location, time, device, etc. Essential ingredients of query personalization are: *(a)* a model for storing preferences in user profiles, and *(b)* algorithms for the generation of personalized answers. Query personalization has three phases: (Preference Selection) Top K preferences are derived from the user profile. (Preference Integration) These are combined with the query. (Personalized Answer) A personalized answer is returned satisfying L of the K preferences.

Contributions. The main contributions are:

- Modeling the plethora of preference types is a challenge. In this paper, we present a preference model that combines expressivity and concision. We model a set of dimensions along which several preference types may be uniformly formulated. The model presented in our previous work [16] captures only preferences of the kind '*I <u>like</u>*

(*) Partially supported by the Information Society Technologies (IST) Program of the European Commission as part of the DELOS Network of Excellence on Digital Libraries (Contract G038-507618)

actor W. Allen'. Preferences such as '*I like films with duration around 2h*', '*I do not like thrillers*', '*I like movies without violence*' captured by the model described here, are not expressed in the model of [16]. We have adopted from [16] the notion of implicit preferences, and the formulation of preferences as degrees of interest in query elements.

- This generalized model calls for more sophisticated preference selection algorithms than the one described in [16]. We provide efficient algorithms for the selection of preferences related to a query according to various criteria. The notion of degree of criticality is introduced for ordering preferences and selecting the top *K*.
- A simple approach for generating personalized answers is to integrate the top *K* preferences into the query issued and construct a new one. This query is, then, executed by the underlying database system [16]. We see how this simple method may be adopted to the preference model described here and discuss its shortcomings. Then, we describe an algorithm for the progressive generation of personalized results, which are ranked based on the user profile.
- Results may be ranked based on which preferences are satisfied or not. Several classes of ranking functions are described, the function provided in [16] being an instance of one of them. New functions belonging to other classes are presented.
- We present experimental results showing *(a)* the efficiency of our algorithms, *(b)* the benefits of personalized search, and *(c)* the appropriateness of the proposed ranking functions.

2. Related Work

Preference is a fundamental notion in applied mathematics [8], philosophy [9], AI [22]. In Databases, preferences have been used for cooperative query answering, i.e., for providing answers with extra or alternative information that may be meaningful to the user [17, 7]. Recently, database research has focused on studying preferences as user criteria at the query level that may be satisfied as closely as possible. Two approaches have been pursued. In the *qualitative* approach, preferences between tuples in the answer to a query are specified using *preference relations*. Two frameworks have been proposed, in which preference relations are defined using logical formulas [5] or special preference constructors [14, 15]. Preference relations are embedded into relational query languages through a relational operator that selects from its input the set of the most preferred tuples (e.g., winnow [5], BMO [14, 15]). Skylines [3, 19] are special cases of these preference queries. In the *quantitative* approach, preferences in queries are specified using *scoring functions* that associate a numeric score with every tuple of the answer [1]. Several algorithms have been proposed for efficiently answering top-*K* queries, i.e. queries that retrieve the best *K* objects that minimize a specific function [4, 23, 11].

The model of [16] associates degrees of interests (like scores) with preferences. Yet, there are substantial differences from the quantitative framework [1]. The latter does not capture preferences expressed on relationships between entities, e.g., '*I am very interested in the actors of a film*', and implicit preferences. In addition, it uses distance functions for tuple ranking; thus *top* tuples are those with the *smallest distance* from the target values. On the other hand, ranking functions [16] estimate the overall interest in a tuple with respect to a combination of preferences. *Top* tuples are those with the *highest interest* based on this function.

The model presented here has the aforementioned features of the earlier model, but is of greater expressive power. The earlier model represents only preferences of the kind '*I like actor W. Allen*' (exact positive presence preference), as opposed to the generalized that captures several types, such as '*I like films with duration around 2h*' (elastic preference), '*I do not like thrillers*' (negative preference), '*I like movies without violence*' (regarding absence of values).

Compared to our extended model, the quantitative framework [1] does not capture negative preferences and preferences for the absence of values. The qualitative frameworks [5, 15] do not capture preferences expressed on relationships between entities and implicit preferences. Besides, [15] defines specific preference constructors, thus not considering the possibility of having arbitrary constraints in preferences as we and [5] do. [5] does not express negative preferences and preferences for the absence of values. Furthermore, preference relations provide an abstract, generic way to talk about priority, and importance. Thus, [5, 15] cannot capture different degrees of interest, such as '*I like comedies very much*', '*I like dramas a little*', and preference queries return most preferred tuples without distinguishing how better is one tuple compared to another. We capture such variations in priority and importance by associating preferences with degrees of interest. Query results are also ranked based on their degree of interest. Then, an application may use qualitative descriptors for preferences and desired results defined in terms of intervals of degrees of interest. E.g., a '*best*' descriptor could map to degrees between 0.9 and 1; then a user could ask for '*best*' answers. We do not, yet, support skylines, and relative preferences [5].

All the above database approaches deal with the expression of preferences in queries. We focus on the representation of preferences in user profiles and query personalization algorithms. Although personalization is a very broad research area, and there are different approaches from information filtering and recommender systems [21, 13] to intelligent agents [2], query personal-

ization approaches in IR [20, 18] and databases [16] are just emerging. Cooperative query answering approaches put the user into perspective as well [17, 7]. These, however, have focused on providing answers that are meaningful to a human being thus containing extra or alternative information. Query personalization focuses on providing focused, smaller answers.

Capturing different types of user preferences in profiles is a challenge. Existing work has primarily focused on the population of simple, keyword profiles for IR systems. Constructing profiles of richer preference types as the ones described here has recently attracted interest in the database community [10].

3. Preference Model

Consider a movies database described by the schema below; primary keys are underlined.

```
THEATRE(tid, name, phone, region, ticket)
PLAY(tid, mid, date), GENRE(mid, genre)
MOVIE(mid, title, year, duration)
CAST(mid, aid, award, role)
ACTOR(aid, name)
DIRECTED(mid, did),  DIRECTOR(did, name)
```

Preferences may be expressed for values of attributes, and for relationships between entities. Preferences for *values* are quite involved, as the following example shows. Preferences for *relationships* indicate to what degree, if any, entities related are influenced by each other (in particular by preferences on each other).

Example 1. Al's preferences include the following.

(P_1) He likes director W. Allen a lot.
(P_2) He prefers ticket prices around 6 Euros.
(P_3) He does not like movies released before 1980.
(P_4) He likes only movies of duration around 2h.
(P_5) He is happy if the movie is not musical.
(P_6) He would rather not go to non-downtown theatres.
(P_7) He is extremely interested in the director of a movie.
(P_8) He is very interested in the movie genre.
(P_9) He cares less about what theatres show a movie.
(P_{10}) He is very concerned with the movies of a theatre.

Our approach to personalization is based on maintaining, for every user, a user profile whose structure is related to the features of the data and query models. Without loss of generality, we focus on SPJ (Select-Project-Join) queries over relational databases. Nevertheless, our approach is applicable to any graph model representing information at the level of entities and relationships. User preferences may be articulated over a higher level graph model representing the data other than the database schema. This is a useful abstraction for using a profile over multiple databases with similar information but possibly different schemas, and for hiding schema restructuring. In ongoing work, we see how preferences expressed over a higher level model may be transparently mapped to the database schema.

3.1. Stored Atomic Preferences

For an attribute *R.A* of a relational table *R*, let D_A be its domain of values. Given our focus on query personalization, we store preferences at the level of atomic query elements, which are therefore called *atomic preferences*. Preferences for values of attributes are stored as atomic selections (atomic selection preferences), and preferences for relationships are stored as atomic joins (atomic join preferences).

Atomic Selection Preferences. For any atomic selection condition q on attribute *R.A*, a user's preference for values satisfying (or not) q is expressed by the *degree of interest* in q, denoted by doi(q), defined as follows:

doi(q) = ($d_T(u)$, $d_F(u)$)

where $\forall\ u \in D_A$ satisfying q,

$d_T(u)$, $d_F(u) \in [-1, 1]$ and $d_T(u)*d_F(u) \leq 0$.

(For simplicity, we may often omit parameter u from the doi's). The last condition should hold for normal users, based on psychological evidence [6]. This model is quite general and can express several preference types. These are described below, as each part of the above definition is analyzed, by distinguishing three relevant dimensions of preferences: valence, concern, elasticity.

Valence. Preferences may be *positive* (expressing liking), *negative* (expressing dislike) or *indifferent* (expressing don't care). Valence is captured by the different values of the degrees of interest $d_T(u)$, and $d_F(u)$: a positive degree indicates increasingly higher interest; a negative degree indicates increasing dislike; a degree equal to 0 indicates indifference. Preferences with $d_T(u) = d_F(u) = 0$, are not stored in the profile.

Concern. Preferences may be *presence* (concerning the presence of values) or *absence* (concerning the absence of values). A user's concern is captured by the pair ($d_T(u)$, $d_F(u)$). As defined, $d_T(u)$ captures a user's concern for the presence of values u of *R.A* (or any other path of the schema leading to *R.A*) that make q evaluate to true. $d_F(u)$ captures a user's concern for the absence of the same values, i.e. for q evaluating to false. $d_T(u)$ is not derivable from $d_F(u)$, and vice versa. Strong interest in a value could be combined with indifference or with strong negative interest in its absence.

Elasticity. Preferences may be *exact* or *elastic* depending on whether the domain D_A is categorical or numeric. Given the mutual independence of categorical values, preferences for these are considered exact and are either satisfied exactly or not at all. On the other hand, preferences for numeric values may be smoothly continuous over their domain and may be satisfied approximately, and thus are considered elastic. Elasticity is captured by the form of the functions $d_T(u)$, and $d_F(u)$. Constant doi functions are used for exact preferences. There are many possible functions for the representation of elastic preferences. Figure 1 shows possible forms of those. Various

parameters are required for the detailed description of an elastic doi function, such as the interval of values for which the function is non-zero. For simplicity, we will use $e_{(d)}$ to denote an elastic function avoiding a detailed representation of it. The subscript denotes the maximum (minimum) degree this function returns, depending on its form, (see Figure 1). We have experimented with functions of the form of Figure 1(a). Using a set of elastic doi functions, a system may support fuzzy operators, such as 'around', for expressing elastic preferences by users.

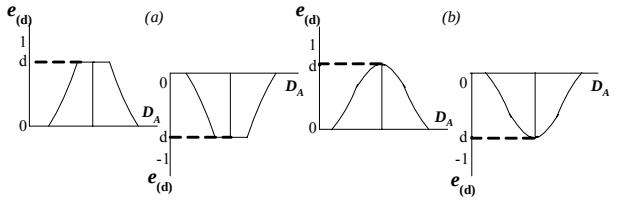

Figure 1. Forms of elastic doi functions

Using these dimensions, all (3*2*2) combinations of the above preference types are valid for formulating preferences. The model in our earlier work captured only one type: exact positive presence preferences.

Example 1 (cont'd). We draw examples from Al's preferences. Regarding valence, P_1 is an instance of a positive preference, and P_3 is an instance of a negative one. Regarding concern, one may be concerned for the presence (absence) of a value, while being indifferent for the opposite case. These are simple preferences. E.g., Al has a positive interest in the presence of W. Allen but he does not care if W. Allen has not directed a film. Consequently, P_1 is a simple positive presence preference. On the other hand, he prefers downtown theatres and he is against the idea of a theatre not being there. P_6 combines positive presence and negative absence preference as one; it is a complex preference. Regarding elasticity, P_1, and P_3 are instances of exact preferences. However, Al's preference for movies with duration around 2 hours (P_4) is elastic, as movies of 122 or 115 minutes are close matches probably of similar interest to him.

Atomic Join Preferences. Join preferences are simpler as they do not lend themselves to any of the variations mentioned above. A user's preference for a join condition q is expressed by the *degree of interest* in q, doi(q), defined as follows:

doi(q) = (d), where d ∈ [0, 1].

Degree 0 indicates lack of any interest in the join condition, while degree 1 indicates extreme ('must-have') interest. In addition, join preferences are directed. E.g. movies and theatres are related but Al thinks that theatres depend on movies (P_{10}) much more than the other way around (P_9). Therefore, a join preference expresses the dependence of the left part of the join on the right part. In other words, the left part indicates the relation already included in a query and the right corresponds to the relation that may be included influencing the final result, if the join is considered.

A user's preferences over the contents of a database can be expressed on top of a *personalization graph* [16]. This is a directed graph $G(V, E)$ (V: the set of nodes; E: the set of edges) and it is an extension of the database schema graph. Nodes in V are (*a*) *relation nodes*, one for each relation in the schema, (*b*) *attribute nodes*, one for each attribute of each relation in the schema, and (*c*) *value nodes*, one for each value that is of any interest to this user. Edges in E are (*a*) *selection edges,* from an attribute node to a value node representing a potential selection condition, and (*b*) *join edges*, from an attribute node to another attribute node representing a potential join condition between these attributes. As explained earlier, two attribute nodes may be connected through two different join edges, in the two possible directions. Given the 1-1 mapping between edges in the graph and atomic preferences, degrees of interest are placed as labels on the edges. Figure 2 shows how Al's profile may look like. Part of the personalization graph corresponding to Al's profile is illustrated in Figure 3.

(P_1) doi(DIRECTOR.name='W. Allen') = (0.8, 0)
(P_2) doi(THEATRE.ticket='6Euros') = ($e_{(0.5)}$, 0)
(P_3) doi(MOVIE.year<1980) = (-0.7, 0)
(P_4) doi(MOVIE.duration='2h') = ($e_{(0.7)}$,$e_{(-0.5)}$)
(P_5) doi(GENRE.genre='musical') = (-0.9, 0.7)
(P_6) doi(THEATRE.region='downtown')=(0.7, -0.5)
(P_7) doi(MOVIE.mid=DIRECTED.mid) = (1)
doi(DIRECTED.did=DIRECTOR.did) = (0.9)
(P_8) doi(MOVIE.mid=GENRE.mid) = (0.8)
(P_9) doi(MOVIE.mid=PLAY.mid) = (0.7)
doi(PLAY.tid=THEATRE.tid) = (1)
(P_{10}) doi(THEATRE.tid=PLAY.tid) = (1)
doi(PLAY.mid=MOVIE.mid) = (1)

Figure 2. Al's profile

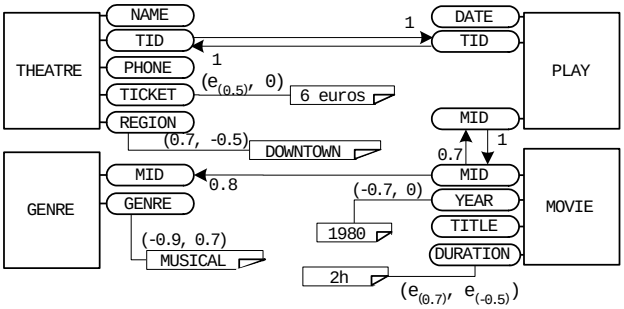

Figure 3. Part of person. graph for Al's profile

3.2. Implicit Preferences

By composing atomic user preferences that are adjacent in the personalization graph (*composable*), one is able to build *implicit preferences*, i.e., preferences expressed through relationships. Given the 1-1 mapping between edges in the personalization graph and atomic preferences, an implicit user preference is mapped to a directed path. An *implicit join preference* is mapped to a

path between two attribute nodes comprising composable join edges, and represents the "implicit" join condition between these attributes. An *implicit selection preference* is mapped to a path from an attribute node to a value node comprising join edges and a selection edge that are composable, and represents the "implicit" selection condition connecting the corresponding attribute and value. An implicit query element is the conjunction of the constituent atomic ones, and the degree of interest in it is a function of the degrees of interest in the participating atomic preferences. In principle, one may imagine several functions. All of them should satisfy the condition that the absolute doi in an implicit preference decreases as the length of the corresponding directed path increases, capturing human intuition and cognitive evidence [6]. We have chosen multiplication as this function.

Example 2. Preferences P_1 and P_7 from Al's profile are composed into the following implicit preference for movies directed by W. Allen.

```
doi( MOVIE.mid=DIRECTED.mid and
     DIRECTED.did=DIRECTOR.did and
     DIRECTOR.name='W. Allen') = ( 0.72, 0 )
```

Any directed path in the personalization graph could map to an implicit preference. However, based on human intuition and cognitive evidence [6], we deal with acyclic paths only. As a matter of notation, we use $< q, \text{doi}(q) >$ to denote an atomic or implicit preference P.

3.3. Combinations of Preferences

Satisfaction of a selection preference $< q, \text{doi}(q) >$ is equivalent to satisfaction of q if $d_T \geq 0$ or failure of q if $d_F \geq 0$. *Failure* of a preference is the exact opposite. Thus, the doi in the satisfaction of a preference is $d^+(u)=max(d_T(u), d_F(u))$. The doi in the failure is $d^-(u)=min(d_T(u), d_F(u))$.

Example 3. Consider Al's preferences P_1 and P_5. P_1 is satisfied by tuples that satisfy the corresponding condition, e.g. movies directed by W. Allen. P_5 is satisfied by tuples that do not satisfy the corresponding condition, e.g. theatres not playing musicals.

The overall doi in a combination of preferences is calculated using a *ranking function*. We distinguish the following cases: *(a)* all preferences are satisfied (positive combination), *(b)* none of the preferences is satisfied (negative combination), and *(c)* some preferences are satisfied and others not (mixed combination).

Positive Combinations. Consider a set P_+ of N_+ preferences and the set D_+ of the corresponding satisfaction (non-negative) doi's:

$D_+ = \{d_i^+ \mid d_i^+ : \text{doi in } P_i \in P_+, i = 1 \ldots N_+\}$

The doi in a positive combination should be a function of the degrees d_i^+. In principle, one may imagine several functions. A parameter that appears pivotal in this issue is $max(D_+)$. Around it, one may see three different philosophies: inflationary, dominant, and reserved.

Inflationary. The degree of interest in multiple preferences satisfied together increases with the number of these preferences, i.e., $r^+(D_+) \geq max(D_+)$, expressing a philosophy of 'the more preferences satisfied the better'. The following function proposed in [16] belongs here:

$$r_1^+ = 1 - \prod_{i=1}^{N} (1 - d_i^+) \quad (1)$$

Dominant. The degree of interest in multiple preferences satisfied together is equal to the doi of the most interesting of these preferences, i.e. $r^+(D_+) = max(D_+)$

This function captures a 'winner-takes-all' philosophy, thus it does not depend on the number of preferences. In other words, an answer is as good as its best feature.

Reserved. The degree of interest in multiple preferences satisfied together is between the highest and the lowest degrees of interest among the original preferences, i.e., $min(D_+) \leq r^+(D_+) \leq max(D_+)$. The underlying principle is that the doi in satisfying multiple preferences should primarily depend on the importance of them. The following function belongs to this category:

$$r_2^+ = 1 - \prod_{i=1}^{N} (1 - d_i^+)^{1/N} \quad (2)$$

The appropriateness of a ranking function is judged only by the philosophy of the approach taken towards personalization and, more importantly, by how closely it reflects human behavior. We have experimented with the above functions, and we will discuss results that provide insight as to the appropriateness and intuitiveness of each one of them.

Negative Combinations. A similar issue arises with respect to the doi in multiple preferences not satisfied, i.e., dealing with multiple non-positive doi's in a set D_-. This case is symmetric with the previous one and may be treated in a similar fashion. The pivotal parameter is $min(D_-)$ and one may define inflationary, dominant, and reserved ranking functions. The counterparts of r_1^+ and r_2^+ above, are exactly the same, only with an exchange of the '+' and '–' sign everywhere.

Mixed Combinations. The doi in a combination of positive (D_+) and negative (D_-) degrees is a function of the degrees of interest in the two sets satisfying the followings conditions:

$$r^-(D_-) \leq r(D_+, D_-) \leq r^+(D_+) \quad (3)$$

$$r(d, -d) = 0 \quad (4)$$

Examples of such functions are the following:

$$r_1 = r^+ + r^- \quad (5)$$

$$r_2 = \frac{N_+ * r^+ + N_- * r^-}{N_+ + N_-} \quad (6)$$

We have experimented with these formulas as well. Formula (6) seems more appropriate, as it captures the

intuition that the overall degree of interest should be affected not only by the doi's in its positive and negative parts, but also by the number of preferences contributing to each one of them.

Personalized answers may be ranked with the use of a ranking function.

3.4. Preference Order

Ordering preferences based on their importance is essential for selecting which ones should be satisfied. Such ordering should take into account both doi's d^+ and d^-. Intuitively, the most important or critical preference is the one with the highest d^+, and the lowest d^-. Accordingly, the *degree of criticality* c of an atomic or implicit preference is defined as follows

$$c = d_0^+ + d_0^- \quad (7)$$

$c \in [0, 2]$ and $d_0^+ = max(d^+(u))$, $d_0^- = |min(d^-(u))|$.

Example 4. Al's preferences P_1, P_4 and P_5 are ordered in decreasing criticality as follows:

P_5 (c_5=1.6), P_4 (c_4=1.2), P_1 (c_1=0.8).

Criticality can be extended to join preferences by assuming the degree of interest in their failure as being equal to 0. As a result, the property of decreasing degree of interest of a join as the length of the corresponding path increases transfers over to the degree of criticality as well. Unfortunately, the same does not hold for implicit selections: the degree of criticality of implicit selection preference c_S may be greater than the degree of criticality of any constituent join preference c_J, since c_S is the sum of two positive doi's. The following bound is derived by applying simple mathematics (not described here due to space constraints).

$$c_S \leq 2c_J \quad (8)$$

4. Preference Selection

The first step of the query personalization process deals with for the extraction of the top (most critical) K preferences related to a query. A preference may be related at a syntactic or semantic level. Our system currently supports the former level. A preference is syntactically related to a query, if it maps to a path attached to a relation included in the query. For example, an implicit preference related to a query about movies is:

```
MOVIE.mid=GENRE.mid and GENRE.genre='comedy'
```

Parameter K is specified with the use of some criterion. We distinguish two possibilities: *(a)* the criterion is based on the degree of criticality of preferences, e.g. it may specify that the top 5 preferences, or preferences with a degree of criticality above a threshold c_0, should be selected, or *(b)* the criterion is based on the desired doi in results, e.g. it may designate results of doi > 0.8.

4.1. Selection Based on Preference Criticality

Problem Formulation. Given a query Q and the personalization graph G_P corresponding to a user profile, we consider the set P_N of all paths P_i in G_P that are related to Q in decreasing order of criticality c_i, i.e.,

$P_N = \{P_i | \, i \in [1, N], \;\; c_{i-1} \geq c_i \;\}$

The set of preferences that may affect the query, based on some criterion $C(.)$ on the degrees of criticality, is the ordered subset $P_K = \{P_i | \, i \in [1, K], \;\; c_{i-1} \geq c_i \;\}$ of P_N such that: $K = max(\{ t | \, t \in [1, N]: C(P_t) \text{ holds } \})$.

Algorithms. A preference selection algorithm should gradually construct directed paths attached to query relations on the personalization graph G_P in decreasing order of criticality. Consider the personalization graph depicted in Figure 4. For simplicity, attributes and values involved in joins and selections are omitted. Each edge is labeled with the degree of criticality of the corresponding atomic preference. The property of decreasing degree of interest of a join as the length of the corresponding path increases gives the possibility of a best-first traversal of the personalization graph: AB being more critical than AE guarantees that ABD is more critical than AEF as well. Unfortunately, monotonicity is lost for the degree of criticality of implicit selection preferences. Indeed, $ABDs_1$ is *not* more critical than $AEFs_2$. Hence, a best-first traversal of the graph does not guarantee that implicit selections are generated in the proper order. For this reason, when an implicit selection preference is encountered, it is output provided that it is more important than the *most critical selection preference unseen* (*mcsu*). Based on Formula (8), the latter comprises the most critical join currently known followed by an atomic selection with the greatest degree of criticality, which equals to 2. Thus, an implicit selection preference may be safely output only if it *has a degree of criticality at least equal to the degree of criticality of the most critical known join multiplied by two*. Otherwise, the algorithm expands that join in order to examine longer paths. This algorithm is called *SPS* (Simple Preference Selection).

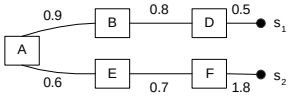

Figure 4. Example on the degree of criticality

Assuming that the most critical known join is followed by an atomic selection with a degree of criticality equal to 2 gives a worst-case estimate for *mcsu*. What the algorithm needs would be the real degree of criticality of the most critical selection preference following that join. For this purpose, a pre-processing step would be necessary: for each join edge, all subsequent paths should be visited in order to find the maximum degree of criticality among them. Then, this degree could be tagged on that join edge. However, neither this pre-processing step nor, mainte-

nance of that extra information is cheap. If the degree of criticality of some edge changes, or a new edge is added, then all join edges that expand to paths including this edge must be updated. A compromise between using a worst-case estimate and storing the real degree is the idea of keeping a *fake criticality fc*, as follows:

For every selection edge, *fc* is set to 1. For every join edge, *fc* is set to the maximum degree of criticality of all *edges* following this one. If one of those is a join, its degree of criticality is multiplied by 2.

Both creation and maintenance of fake criticalities are cheap. Then, a preference selection algorithm may treat each path with a degree of criticality *c* and a fake criticality *fc*, as if it were an implicit selection preference with criticality equal to $c*fc$ (instead of *c*). As a result, a best-first traversal of the personalization graph G_P based on the product $c*fc$ is now possible. Whenever a selection preference is constructed, it is immediately output. The algorithm, called *FakeCrit*, is presented in Figure 5. Experiments not presented in this paper for space constraints, have shown that it is more efficient than the simple *SPS* algorithm.

Preference Selection Algorithm-FakeCrit

Input: User profile *U*, user query *Q*, criterion *C*
Output: Set of preferences P_K

P_K ={}, *QP*={}, *K_Selected*=false
1. **Foreach** atomic preference $AC_i \in U$ *related* to *Q*
  1.1 **If** (AC_i does not *conflict* with *Q*) **Then** $QP \leftarrow AC_i$ **End if**
  End for
2. **While** (*QP* not empty) **and** (*K_Selected*=false)
  2.1 Get head *P* from *QP*
  2.2 **If** (*P* is selection) **Then**
    K_Selected=*C*($P_K \cup \{P\}$)
    If (*K_Selected*= false) **then** $P_K \leftarrow P$ **End if**
  End if
  2.3 **If** (*P* is join) **Then**
    K_Selected=*C*($P_K \cup \{P\}$)
    If (*K_Selected*= false) **then**
    Foreach atomic element $AC_i \in U$ *composable* with *P*
      If ((c_0>0) **and** ($fc_{PA}*c_{PA} \le c_0$) **Then** exit **For** **End if**
      If (AC_i does not join to relation $R \in P$ or $R \in Q$) **then**
      $QP \leftarrow P \wedge AC_i$ **End if**
    End for
  End if
  End if
End while

Figure 5. FakeCrit Preference Selection

A queue *QP* of preferences is kept in order of decreasing $c*fc$. Initially, it contains atomic preferences related to the query. In each round, the algorithm picks from *QP* the head *P*. If *P* is a selection satisfying the criterion $C(P_K \cup \{P\})$, then it is output. If *P* is a join satisfying the criterion $C(P_K \cup \{P\})$, then, it is *expanded* into longer paths which are added into *QP*. A new path $P \wedge AC_i$ is generated for each atomic preference AC_i that is composable with *P*. These atomic preferences are considered in order of decreasing $c*fc$. A new path is not inserted in *QP*: *(a)* if it expands to a relation included into *P* or *Q*, because a cycle is generated; *(b)* if the product of its degree of criticality and its fake degree of criticality ($c_{PA}*fc_{PA}$) is $< c_0$, provided that criterion *C* specifies that top *K* preferences must have a degree of criticality greater than $c_0 > 0$.

4.2. Selection Based on the Interest of Results

Selection of the top *K* preferences may be guided by a criterion on the desired doi in results. This criterion could be formulated with the use of a ranking formula, such as (1) or (2), calculating the doi in positive combinations. Compared to the previous case, this one presents certain particularities that need to be considered by a preference selection algorithm.

Example 5. Consider the following preferences.

(P_1) `doi(MOVIE.mid=GENRE.mid) = (1, 0 )`
(P_2) `doi(GENRE.genre='musical' = (-0.7, 0 )`
(P_3) `doi(GENRE.genre='adventure') = (0.9, 0 )`

Assume we are interested in movies with a doi higher than 0.8. The preference selection algorithm could select only P_3, since movies satisfying only this preference are interesting based on the criterion above. In practice, the personalized query executed will return movies that possibly satisfy some of the preferences ignored and do no satisfy some others. Using a ranking function for mixed combinations, we see that results not satisfying preferences with negative doi have a decreased doi than expected. For example, we see that movies satisfying P_3 but not P_2 are not desired, due to the negative doi of P_2; these should not appear in the answer. Consequently, whenever interested in personalized results with a minimum doi, negative preferences must be taken into consideration.

Problem Formulation. Given a query *Q* and the personalization graph G_P corresponding to a user profile, we consider again the set P_N of all paths P_i in G_P that are related to *Q* in decreasing order of their degree of criticality c_i, i.e., $P_N = \{P_i \mid i \in [1, N],\ c_{i-1} \ge c_i\ \}$

The set of preferences that must be satisfied so that tuples returned will have a minimum degree of interest equal to d_R, despite the fact that they may not satisfy preferences not selected, is the ordered subset $P_K = \{P_i \mid i \in [1, K], c_{i-1} \ge c_i\ \}$ of P_N such that:

$K = min(\{t \mid t \in [1, N]: r(d_1^+, \ldots d_t^+, d_{t+1}^-, \ldots d_N^-) \ge d_R\})$

(*r* is a ranking function for mixed combinations).

Algorithm. An exhaustive algorithm would enumerate all paths in the ordered P_N and repeat this calculation

$$r(d_1^+, d_2^+, \ldots d_t^+, d_{t+1}^-, \ldots d_N^-)\ \forall t = 1 \ldots N \qquad (9)$$

until it returns a doi greater than or equal to d_R.

A more efficient algorithm is built by appropriately extending *FakeCrit*. As before, a queue *QP* of candidate preferences is kept in order of decreasing $c*fc$. Initially, it contains all atomic preferences related to the query. In each round, the algorithm picks from *QP* the head *P*. In round *t*, *t* preferences have been selected. The problem is

how to compute Formula (9), without visiting the remaining *N-t* paths. Recall that the absolute doi in an implicit preference decreases as the length of the corresponding directed path increases. Then, the absolute doi d_i^- of any negative preference unseen can be at most equal to d_{worst}:

$d_i^- \le d_{worst}\ \forall P_i,\ i = t+1, \ldots N$

where d_{worst} can be computed by considering the doi's of all preferences known, i.e., currently in *QP*, as follows:

$d_{worst} = max(\{d_i^- \mid P_i \in QP \text{ and } P_i \text{ is selection}\} \cup \{d_j \mid d_j = \text{the doi in } P_j \in QP \text{ and } P_j \text{ is join}\})$

where, $d_i^- = |min(d_i^-(u))|$ of a preference P_i.

By considering the worst case scenario, i.e.

$d_i^- = d_{worst}\ \forall P_i,\ i = t+1, \ldots N,$

Formula (9) is written

$$r(d_1^+, d_2^+, \ldots d_t^+, -d_{worst}, \ldots -d_{worst}) \qquad (10)$$

where $-d_{worst}$ is repeated $N-t$ times.

At each iteration, the algorithm caches the doi in results satisfying *t* preferences given by $r(d_1^+, d_2^+, \ldots d_t^+)$ in order to re-use it in the next round. The problem is that without exhaustive enumeration of P_N, *N* is unknown. We may assume that *N* is equal to the number of preferences stored in the profile. Whether this estimate is close to the real value of *N* depends on the structure of the personalization graph. If its real value is much smaller, then Formula (10) assumes that there are more preferences to be examined than in reality. This may possibly result in enumerating all paths in P_N, which may be acceptable. Alternatively, for each join edge the number of paths that this edge expands to could be kept. This number may not be updated every time an edge is inserted or deleted from the graph. We have found that the selection algorithm can be effective relying only on periodic updates of this number.

5. Generation of Personalized Answers

Top *K* preferences are integrated into the user query and a personalized answer is generated. This should be:

(a) Interesting to the user. For this purpose, it should satisfy (at least) *L* from the top *K* preferences.

(b) Ranked based on the doi in the tuples returned.

(c) Self-explanatory. For each tuple returned, the preferences satisfied and/or not should be provided in order to justify its selection and ranking.

We describe two approaches for the generation of personalized answers. Elastic preferences are translated into appropriate range conditions using a set of rules before they can be inserted into a query. This is not discussed here any further, due to space limitations.

Simply Personalized Answers (**SPA**). One approach is to integrate the top *K* preferences into the initial query and build a new one, which is executed. We formulate the personalized query *as the union of a set of sub-queries*, each one mapping to one or more of the *K* preferences selected. Each sub-query is built by extending the initial query by an appropriate qualification involving the participating preferences. It also returns the positive degree of interest of the corresponding preference. If it contains an elastic preference, then the corresponding elastic function provides the doi in each tuple. This approach is adapted from [16], so that it can handle elastic and absence preferences, not captured in our previous work. We will give a representative example, without going into technical details.

Example 6. Suppose Al submitted this simple query

```
select title from movies
```

Assume that the following preferences have been selected, from which *L*=2 should be satisfied.

```
(P1)  MOVIE.mid=DIRECTED.mid and
      DIRECTED.did=DIRECTOR.did and
      DIRECTOR.name='W. Allen'     (presence)
(P2)  MOVIE.year<1980              (absence 1-1)
(P3)  MOVIE.mid=GENRE.mid and
      GENRE.genre='musical'
                                   (absence 1-n)
```

The kind of sub-query depends on the preference type.

A preference to be satisfied may be presence or absence preference. Moreover, we distinguish between 1–1 and 1–n absence preferences. The following sub-queries are built for each preference type.

(*Presence preferences*)

```
Q1: select title, 0.72 degree
    from   MOVIE M, DIRECTED D, DIRECTOR DI
    where  M.mid=D.mid and D.did=DI.did and
           DI.name='W. Allen'
```

(*1–1 absence preferences*) They are mapped to sub-queries in the same way as presence ones. The only difference is the change of the condition's operator.

```
Q2: select title, 0 degree
    from   MOVIE M
    where  M.year>=1980
```

(*1–n absence preferences*)

```
Q3: select title, 0.7 degree
    from   MOVIE M
    where  M.mid not in  (select M.mid
                 from MOVIES M, GENRE G
                 where M.mid=G.mid and
                 G.genre='musical')
```

The expected results are obtained by taking the union of the partial results of the sub-queries, grouping by the projected attributes of the initial query, and excluding all groups with less than *L* rows. Results are ranked based on the combination of preferences satisfied.

```
select   title,r(degree)
from    Q1 Union All Q2 Union All Q3 Union All
group by title
having   count(*) >= 2
order by  r(degree)
```

where *r* is a ranking function (implemented as a user-defined aggregate function), and each sub-query is replaced by Q_i for presentation purposes.

Although this approach is simple, it has certain disadvantages. It does not generate self-explanatory results. It cannot rank results based both on preferences from the *K* selected are satisfied and which are not. It may become

very inefficient when there are 1–n absence preferences. It does not allow for a progressive retrieval of tuples. Tuples are returned only after they have all been retrieved, merged, grouped and ordered.

Progressive Personalized Answers (**PPA**). This algorithm generates self-explanatory, ranked, personalized answers. It outputs results in a progressive fashion, and it handles 1–n absence preferences more efficiently.

Queries corresponding to presence and 1–1 absence preferences are constructed in the same way as the previous example showed. For convenience, let's call them presence queries. Let P_S be the set of presence and 1–1 absence preferences, and S the set of the corresponding queries in order of increasing selectivity. We use simple histograms to obtain this information. Queries corresponding to 1–n absence preferences, called absence queries, are now formulated as if they corresponded to presence preferences. Let P_A be the set of 1–n absence preferences, and A the set of the absence queries in order of increasing selectivity. The difference between presence and absence queries is that a tuple returned by the former satisfies the corresponding preference, whereas a tuple returned by the latter does not satisfy the corresponding preference. Each of these queries returns a tuple id, the table attribute and the value of the participating preference, and a doi. Presence queries return a positive doi, while absence preferences return a negative doi.

For each $S_i \in$ S, we build a parameterized query $Q_i^S(t)$ that is the union of all S_k following S_i in S. Likewise, for each $A_i \in$ A, we build a parameterized query $Q_i^A(t)$ that is the union of all A_k following A_i in A. In both cases, parameter t is a tuple id. The algorithm *PPA* is presented in Figure 6. It starts by executing presence queries. For each distinct tuple t returned by a query S_i, the algorithm executes the corresponding parameterized query $Q_i^S(t)$. This query returns zero or more occurrences of t, depending on the number of preferences that are contained in this query, and are *satisfied* by t. The algorithm records how many (*curL*) and which of these preferences are satisfied (*presSatisfied*). Then, it records which preferences were not satisfied (*presFailed*) by considering the difference of the set of satisfied preferences from the set P_S. It also executes the parameterized query $Q_1^A(t)$, in order to make the same with the absence preferences. However, this query returns zero or more occurrences of t, depending on the number of preferences contained in this query that *are not satisfied* by t. Then, the algorithm finds how many and which of the absence preferences are satisfied (*absSatisfied*) by taking the difference of the set of not satisfied absence preferences (*absFailed*) from the set of all absence preferences P_A. If the tuple t satisfies (at least) L preferences ($curL \geq L$), then its overall doi (*TupleDoi*) is calculated using some ranking function (r), and it is inserted in a list of results R in order of decreasing doi. Since, for every tuple t, the algorithm knows exactly which preferences are satisfied (*presSatisfied* ∪ *absSatisfied*) and which not (*presFailed* ∪ *absFailed*), ranking may be performed using any function r for positive, negative or mixed combinations. Then, the algorithm proceeds in the same way with the execution of absence queries. For each distinct tuple t returned by a query A_i, the algorithm executes the corresponding parameterized query $Q_i^A(t)$. It records how many and which of the absence preferences are satisfied, as described above, and inserts the tuple in R, provided that it satisfies L preferences. In addition, it keeps a list *Nids* of all tuple ids returned by absence queries, so that it may return any tuple of the initial query Q with id not in this list.

Progressive Algorithm- PPA

Input: number of preferences to satisfy L,
presence queries S, presence preferences P_S,
parameterized queries $\{Q_i^S(t) | Q_i^S(t)$ corresponds to $S_i\}$,
absence queries A, absence preferences P_A,
parameterized queries $\{Q_i^A(t) | Q_i^A(t)$ corresponds to A_i

$R=\{\}$, $MEDI=f(\{d_i^+ | i=1\ldots K\})$
1. **Foreach** $S_i \in$ S
  1.1 **If** rest of queries don't satisfy L prefs **Then** Exit For **End if**
  1.2 Execute S_i
  1.3 **Foreach** t returned by S_i not contained in R
    Set *presSatisfied*; Set *curl;*
    Execute $Q_i^S(t)$; Update *presSatisfied*; Update *curL*
    presFailed= P_S– *presSatisfied*
    Execute $Q_1^E(t)$; Update *absFailed*;
    absSatisfied = P_A– *absFailed*; Update *curL*
    prefsSatisfied = (*presSatisfied* ∪ *absSatisfied*)
    prefsFailed = (*presFailed* ∪ *absFailed*)
    If *curL* >=*L* **then**
     TupleDoi = r(*prefsSatisfied*, *prefsFailed*)
    $R\leftarrow$(t, *prefsSatisfied*, *prefsFailed*, *TupleDoi*) **End if**
    While ∃ *tuple* ∈ R not output, s.t. *TupleDoi* ≥ *MEDI*
     Output *tuple* **End While**
  End For
  1.4 Update *MEDI*
End For
2. **Foreach** $A_i \in$ A
  2.1 **If** rest of queries don't satisfy L prefs **Then** Exit For **End if**
  2.2 Execute A_i
  2.3 **Foreach** t returned by A_i
    Set *absFailed*; Execute $Q_i^A(t)$; Update *absFailed*
    absSatisfied = P_A– *absFailed*; Update *curL*
    prefsSatisfied = *absSatisfied; prefsFailed* = *absFailed*
    If *curL* >=*L* and t not contained in R **then**
     TupleDoi = r(*prefsSatisfied*, *prefsFailed*)
    $R\leftarrow$(t, *prefsSatisfied*, *prefsFailed*, *TupleDoi*)
    Nids$\leftarrow t$
    End if
    While ∃ *tuple* ∈ R not output, s.t. *TupleDoi* ≥ *MEDI*
     Output *tuple* **End While**
  End For
  2.4 Update *MEDI*
End For
3. $R=R\cup\{$*Allids-Nids*$\}$
4. Output remaining tuples of R

Figure 6. Progressive personalized answers

The algorithm terminates when the remaining, presence or absence, queries do not suffice for satisfying L preferences. At any point, if a tuple t already seen is encountered, it is ignored. In effect, output of results is possible, before retrieving the whole set of them. For this purpose, we maintain a *Maximum Estimated Degree of Interest*

(*MEDI*) that any unseen result can achieve. This is initially equal to the doi in satisfying the entire set of preferences. In each round, it is reduced to the degree of satisfying preferences corresponding to presence or absence queries not yet executed. Any of the ranking formulas presented may be used for calculating *MEDI*. The algorithm outputs tuples in *R*, with degree of interest greater than or equal to *MEDI*.

6. Experimental Results

Experiments were conducted using a system implemented on top of Oracle 9i. Our data comes from the Internet Movies Database [12] with information about over 340000 films. We conducted several experiments with various sets of profiles and queries. Due to space constraints, we discuss results of representative experiments concerning: *(a)* the efficiency of our algorithms, *(b)* the benefits of query personalization, and *(c)* the appropriateness of proposed ranking functions.

6.1. Efficiency of Personalization Algorithms

The parameters affecting execution time of our algorithms are: the number *K* of top preferences, and the number *L* of those that should be satisfied. Figure 7 shows execution times for: *(a) FakeCrit* (Preference Selection Time), *(b) SPA*, *(c) PPA*, plus *(d) PPA*'s first response time, for varying *K* positive presence preferences and *L* = 1. The purpose of considering only positive presence preferences was to see how efficient *SPA* and *PPA* are, when there are no time-consuming absence queries. We can see that the preference selection algorithm is very efficient, thus query personalization time may be considered equal to the time spent by *SPA* or *PPA*. *PPA* has a very good initial response time, and its overall execution time is better than *SPA*'s.

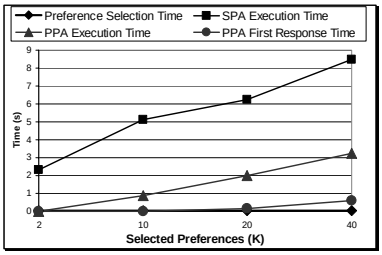

Figure 7. Execution times with K

Figure 8 shows execution times in *L*, for *K*=30 positive presence preferences. Preference selection time is not shown (it does not depend on *L*). *PPA*'s (overall and first response) times decrease in *L* increasing. This is due to the fact that *PPA* executes queries gradually; so, at any point, if the remaining queries do not suffice for satisfying *L* preferences, it stops. *SPA*'s time does not depend on *L*. In addition, *SPA* execution time is very high when there are absence queries. On the contrary, *PPA* is not affected by them as long as their number is below *L*. If not, its overall execution time gets worse but remains more efficient than *SPA*.

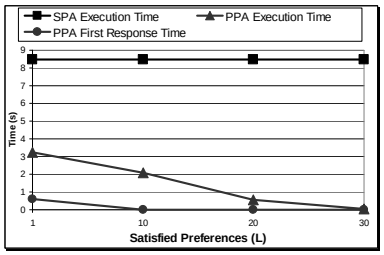

Figure 8. Execution times with L

We only mention here, since we have presented corresponding experimental results in our earlier work [16], that the overall overhead involved in supporting personalization is not significant.

6.2. Effectiveness of Personalized Queries

We conducted an empirical evaluation of our approach with 14 human subjects. 8 of them have a diploma in computer science (experts). The rest of them are simple users of computers. First, each user provided her preferences. Two trials were conducted using a web-based client developed for this purpose.

In the *first trial*, all subjects were given a set of 3 queries plus two additional ones that they would like to ask. Each user submitted the set of 5 queries twice in arbitrary order. Queries were executed once without personalization and once with personalization. This was also performed arbitrarily. Our intention was to let individuals judge the results unbiased by what happens to their query. Each user was asked to electronically evaluate each tuple returned by providing a score in the range [-10, 10] (*tuple interest*), as well as the overall answer to each query. For the latter a user provided three different scores: *(a)* an estimation of the difficulty to find something interesting (*degree of difficulty*), anything was found at all, *(b)* an estimation of how well the answer covered their need (*coverage*), and *(c)* an overall score of the results in the range [-10, 10] (*answer score*). As parameters for personalization, we chose *K* to be the number of preferences in a user profile, and *L*=2. We present some of the results, due to space considerations.

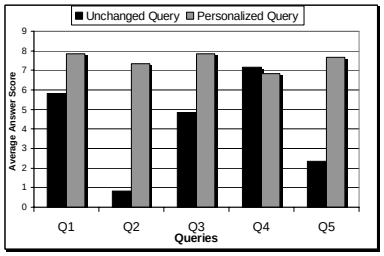

Figure 9. Average answer score (experts)

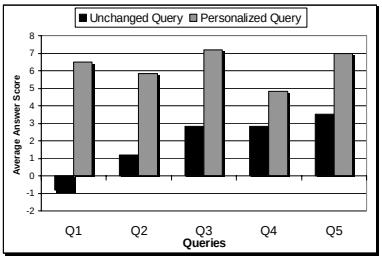

Figure 10. Average answer score (novice)

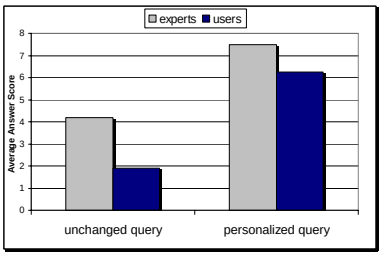

Figure 11. Average answer score per group

Figure 9 and Figure 10 present the average answer score reported when the query was executed unchanged and the average score reported when the query was personalized, for experts and novice, respectively. We see that personalized answers have higher scores. Figure 11 presents the average answer score per group over queries unchanged and queries personalized.

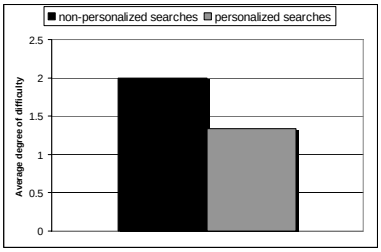

Figure 12. Average degree of difficulty

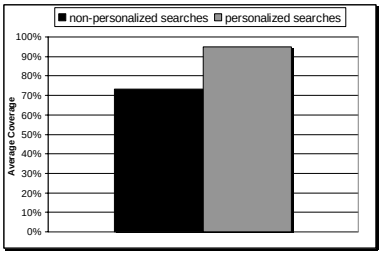

Figure 13. Average coverage

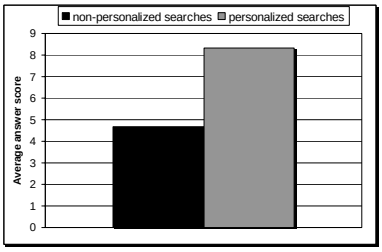

Figure 14. Average answer score

In the *second trial*, all users were asked to think of a specific need, e.g., finding a theatre to go or a DVD to rent. Queries submitted by half of them were not changed, while queries of the rest were personalized.

Figure 12 shows the average degree of difficulty reported by each group. Figure 13 shows the average coverage reported by each group and Figure 14 shows average scores. Overall, these experiments have shown that the benefits of personalized search can be significant in terms of the effort required by people -novices and experts alike- to find information.

6.3. Evaluation of Ranking Functions

In the experiments with human subjects (described in the previous subsection), users were asked to electronically state their interest in each tuple returned by personalized queries. We compared user interest (appropriately normalized) to the degree of interest returned by the three positive ranking functions described earlier.

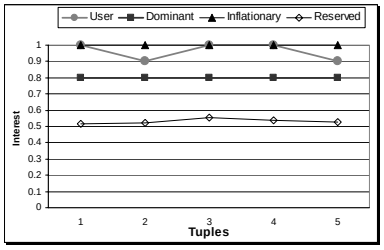

Figure 15. Tuple Interest close to Inflationary

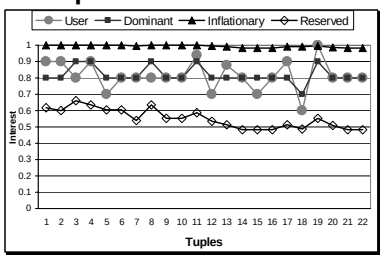

Figure 16. Tuple Interest close to Dominant

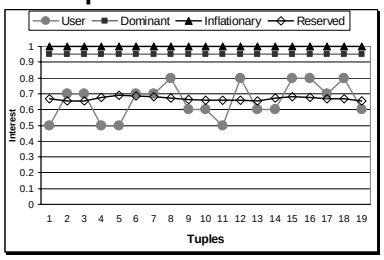

Figure 17. Tuple Interest close to Reserved

Figures 15, 16 and 17 show a user's interest over the tuples of a single query. In Figure 15, we observe that user interest is close to the doi of the inflationary function. In Figure 16, another user has ranked results of a different query following a rather dominant approach. Finally, in Figure 17, user behavior is best reflected by the reserved approach. Overall, experimental results have

indicated that the three ranking functions discussed here capture real users ranking philosophy. Therefore, it seems possible to learn the most appropriate ranking function per user. This information could be stored as part of the user's profile. Further experiments are needed towards the direction of producing psychological evidence regarding the conditions under different ranking approaches are followed by users.

7. Conclusions and Future Work

We presented an expressive preference model, efficient query personalization algorithms, ranking functions, and experimental results showing the efficiency of our algorithms, and the benefits of query personalization, and providing insight as to the appropriateness of the ranking functions. In ongoing work, we are concerned with how preferences expressed over a higher level model may be transparently mapped to an underlying database's schema, and we investigate how various profiling methods proposed in the literature may be adapted for (semi-) automatic construction of user profiles. We are also interested in combining personal preferences with other aspects of a query's context that call for query customization, such as user location, time, device, etc.

8. References

[1] Agrawal, R., Wimmers, E. A Framework for Expressing and Combining Preferences. In Proc. of ACM SIGMOD, 2000.

[2] André E., Rist, T. From adaptive hypertext to personalized web companions. *Comm. of the ACM*, 45(5), 43-46, 2002.

[3] Borzsonyi, S., Kossmann, D., Stocker, K. The Skyline Operator. In Proc. of ICDE, 421–430, 2001.

[4] Bruno, N., Chaudhuri, S., Gravano, L. Top- k Selection Queries over Relational Databases: Mapping Strategies and Performance Evaluation. *ACM TODS*, 27(2), 153-187, 2002.

[5] Chomicki, J. Preference Formulas in Relational Queries. *ACM TODS*, 28(4), 427–466, 2003.

[6] Collins, A., Quillian, M. Retrieval Time from Semantic Memory. *J. of Verbal Learning and Verbal Behaviour*, Vol 8, 240-247, 1969

[7] Cuppens, F. Demolombe, R. How to Recognize Interesting Topics to Provide Cooperative Answers. *Information Systems*, 14(2), 163-173, 1989.

[8] Fishburn, P. Preference Structures and Their Numerical Representations. *Theor. Comput. Sci*. 217, 359–383, 1999.

[9] Hansson, S. O. Preference Logic. *In Handbook of Philosophical Logic*, D. Gabbay, Ed. Vol. 8, 2001.

[10] Holland, S., Ester, M., Kießling, W. Preference Mining: A Novel Approach on Mining User Preferences for Personalized Applications. *PKDD, LNAI 2838*, 204–216, 2003.

[11] Ilyas, I., Shah, R. Aref, W., Vitter, J., Elmagarmid, A. Rank-aware Query Optimization. In Proc. of ACM SIGMOD, 2004.

[12] Internet Movies Database. Available at www.imdb.com

[13] Karypis, G. Evaluation of Item-Based Top-N Recommendation Algorithms. In Proc. of CIKM, 247-254, 2001.

[14] Kießling, W., Köstler, G. Preference SQL-Design, Implementation, Experiences. In Proc. of VLDB, 2002.

[15] Kießling, W. Foundations of preferences in database systems. In Proc. of. VLDB, 2002.

[16] Koutrika, G., Ioannidis, Y. Personalization of Queries in Database Systems. In Proc. of ICDE, 2004.

[17] Gaasterland, T., Godfrey, P. Minker, J. An overview of Cooperative Query Answering. Journal of Intelligent Information systems 1(2), 123-157, 1992.

[18] Liu F., Yu C., Meng W. Personalized Web Search by Mapping User Queries to Categories. In Proc. of ACM CIKM, 558-565, 2002.

[19] Papadias, D., Tao, Y., Fu, G., Seeger:, B. An Optimal and Progressive Algorithm for Skyline Queries. In Proc. of ACM SIGMOD, 467–478, 2003.

[20] Pitkow, J., Schutze, H., et al. Personalized Search. *Comm. of the ACM*, 45(9), 2002.

[21] Shahabi, C., Banaei-Kashani, F., Chen, Y., McLeod D. Yoda: An Accurate and Scalable Web-based Recommendation System. In Proc. of COOPIS., 2001.

[22] Wellman, M.P., Doyle, J. Preferential semantics for goals. Proc. of the National Conf. on AI, 698–703, 1991.

[23] Zhu, L., Meng W. Learning-Based Top-N Selection Query Evaluation over Relational Databases. In Proc. of WAIM, 2004.