Personalization of Queries Based on User Preferences

Georgia Koutrika , Yannis Ioannidis

University of Athens,
Department of Informatics and Telecommunications,
Athens, Greece
{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 algorithms for the selection of preferences related to a query and the progressive generation of personalized results, which are ranked based on user interest.

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. Thus, they may expect different answers to the same query. Consider a simple case: two users, Jon and Julie, access a web-based movies database both searching for comedies. Jon 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 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 [18]. Personalized results for Jon 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

Dagstuhl Seminar Proceedings 04271
Preferences: Specification, Inference, Applications
http://drops.dagstuhl.de/opus/volltexte/2006/403

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 [14, 18, 16]. We are concerned with query personalization in the context of databases. Essential ingredients of query personalization are *(a)* a model for storing preferences in user profiles, *(b)* a query personalization framework that specifies what kind of personalized answer is generated given a query and a user profile, and *(c)* query personalization algorithms.

We adopt the query personalization framework presented in our earlier work [14]. Based on that, given a query and a profile, a personalized answer is built by specifying the number K of top preferences from the user profile that should be considered, and the number L (L≤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. Query personalization proceeds in two phases: (Preference Selection) Top K preferences are derived from the user profile. (Personalized Answer) These are combined with the query, and a personalized answer is returned satisfying (at least) L of the K preferences.

Outline. In this paper, we present the following:

- *A Preference Model*. We present a preference model that combines expressivity and concision. In particular, we model a set of dimensions along which several types of preferences may be formulated.
- *Preference Selection Algorithms*. 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.
- *Generation of Personalized Answers*. 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 [14]. 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 estimated user interest.
- *Ranking Functions*. Results may be ranked based on which preferences are satisfied or not. Several classes of ranking functions are described.
- *Experimental Results*. We have conducted experiments testing the efficiency of the proposed query personalization algorithms, the appropriateness of the ranking functions, and the effectiveness of personalized search. We will provide an overview of the experimental results in order to give insight so as to the appropriateness of ranking functions, and the effectiveness of personalized search.

2. Related Work

Preference is a fundamental notion in applied mathematics [7], philosophy [8], AI [20], and databases [15]. However, only recently this area has attracted a broader

interest in the database community. 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 [12, 13]. 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 [12, 13]). Skylines [3, 17] 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, 21, 9].

Our earlier model [14] 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 [14] 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, 13] do not capture preferences expressed on relationships between entities and implicit preferences. Besides, [13] defines specific preference constructors, thus not considering the possibility of having arbitrary constraints in preferences. [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, 13] 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 the 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*' matches. We do not, yet, support skylines, and conditional preferences.

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 [19, 11] to intelligent agents [2], query personalization approaches in IR [18,

16] and databases [14] are just emerging.

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. Preferences for *relationships* indicate to what degree, if any, entities related depend on each other (in particular by preferences on each other).

Example 1. Jon's preferences include the following.

(p_1) He likes director W. Allen very much. (p_2) He does not like movies released before 1980. (p_3) He prefers a ticket price around 6 Euros. (p_4) He prefers movies of duration around 2h. (p_5) He is happy if the movie is not a 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 cares a lot about the movie genre. (p_9) He cares less about the theatres showing a movie. (p_{10}) He cares a lot about 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 queries over relational databases. Nevertheless, our approach is applicable to any graph model capturing 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.

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), which is 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 $d_T(u) * d_F(u) \leq 0$.

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 (degree 1 is for 'must-have' domain values); a negative degree indicates increasing dislike (degree –1 is for 'most-unpleasant' values); 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, in which case, they 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. Fig. 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 (min.) degree this function returns, depending on its form, (see Fig. 1). We have experimented with functions of the form of Fig. 1 (a). Using a set of elastic doi functions, a system may support fuzzy operators, such as 'around', for the expression of elastic preferences by users.

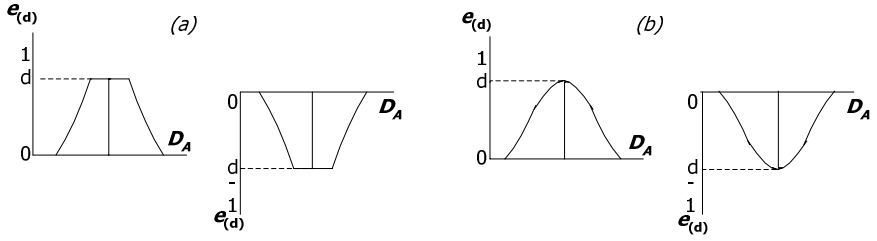

Fig. 1. Example forms of elastic functions

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 Jon's preferences. Regarding valence, p_1 is an instance of a positive preference, and p_2 is an instance of a negative one. Regarding concern, one may be concerned for the presence (absence) of a value, while one is indifferent for the opposite case. These are simple preferences. E.g., Jon 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_2 are instances of exact preferences. However, Jon's preference for movies with duration around 2 hours is elastic, as movies of 122 or 115 minutes are close matches probably of similar interest to him. Thus, p_4 is an elastic preference.

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 Jon 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. Fig.2 shows how Jon's profile may look like.

```
< DIRECTOR.name='W. Allen',          0.8,          0 >
< MOVIE.year<1980,                  -0.7,          0 >
< THEATRE.ticket='6Euros',          e(0.5),        0 >
< MOVIE.duration='2h',              e(0.7),     e(-0.5)>
< GENRE.genre='musical',            -0.9,        0.7 >
< THEATRE.region='downtown'          0.7,       -0.5 >
< MOVIE.mid=DIRECTED.mid,               1           >
< DIRECTED.did=DIRECTOR.did,          0.9           >
< MOVIE.mid=GENRE.mid,                0.8           >
< MOVIE.mid=PLAY.mid,                 0.7           >
< PLAY.tid=THEATRE.tid,                 1           >
< THEATRE.tid=PLAY.tid,                 1           >
< PLAY.mid=MOVIE.mid,                   1           >
```

Fig. 2. Example user profile

A user's preferences over the contents of a database can be expressed on top of a *personalization graph* [14]. 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 a particular user. Likewise, edges in E are *(a) selection edges,* from an attribute node to a value node; such an edge represents the potential selection condition connecting the attribute and the value, and *(b) join edges*, from an attribute node to another attribute node; such an edge represents the 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-to-1 mapping between edges in the graph and atomic preferences, degrees of interest are placed as labels on the edges. Part of the personalization graph corresponding to Jon's profile is illustrated in Fig. 3.

As a matter of notation, we use <q, $d_T(u)$, $d_F(u)$ > to denote a selection preference p, and <q, d > to denote a join preference p. For simplicity, we may omit parameter u from the doi of selection preferences.

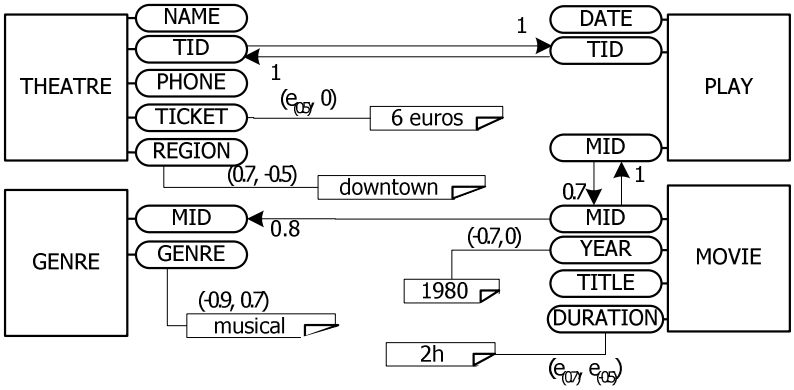

Fig. 3. Example personalization graph

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 one-to-one 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 in the personalization graph between two attribute nodes. It is comprised of composable join edges and represents the "implicit" join condition between the corresponding attributes. An *implicit selection preference* is mapped to a path in the personalization graph from an attribute node to a value node. It is comprised of 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. The degree of interest in an implicit preference is a function of the degrees of interest in the participating atomic preferences. In principle, one may imagine several functions. All of them, however, should satisfy the condition that the absolute degree of interest 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. These preferences from Jon's profile

```
< MOVIE.mid=DIRECTED.mid,              1          >
< DIRECTED.did=DIRECTOR.did,         0.9          >
< DIRECTOR.name='W. Allen',          0.8        0 >
```

are composed into this implicit preference for movies directed by W. Allen:

```
< MOVIE.mid=DIRECTED.mid and
  DIRECTED.did=DIRECTOR.did and
  DIRECTOR.name='W. Allen',        0.72,        0 >
```

Note that 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.

3.3 Combinations of Preferences

Satisfaction of an atomic or implicit selection preference $<q, d_T, d_F>$ 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 degree of interest in the failure is $d^-(u) = min(d_T(u), d_F(u))$.

Example 3. Consider these preferences of Jon.

```
< DIRECTOR.name='W. Allen',           0.8,      0 >
< GENRE.genre='musical',             -0.9,    0.7 >
```

The first one is satisfied by tuples that satisfy the corresponding condition, e.g., movies directed by W. Allen. The second one is satisfied by tuples that do not satisfy the corresponding condition, e.g., theatres that do not play musicals.

The overall degree of interest 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 (for simplicity, we omit u):

$D^+ = \{d_i^+ \mid d_i^+$: doi in $p_i \in P_+, i = 1 \ldots N_+\}$

The degree of interest 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. 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 the better'. The function proposed in [14] 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 exactly equal to the degree of interest 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 degree of interest 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 giving insight as to the appropriateness and intuitiveness of each one of them.

Negative Combinations. A similar issue arises with respect to the degree of interest in multiple preferences not satisfied, i.e., dealing with multiple non-positive degrees 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 degree of interest 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) is more appropriate, as it captures the intuition that the overall degree of interest should be affected not only by the degrees of interest 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

The notion of degree of criticality is introduced for ordering preferences and selecting the top K of them. Intuitively, the most important or critical preference is the one with the highest d^+, and the lowest d^-.

The *degree of criticality* c of a preference $<q, d_T(u), d_F(u)>$ is defined as follows

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

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

Based on the degree of criticality, preferences are ordered as the following example shows.

Example 4. These preferences from Jon's profile

p_1: < DIRECTOR.name='W. Allen', 0.8, 0 >
p_4: < MOVIE.duration='2h', e(0.7), e(-0.5)>
p_5: < GENRE.genre='musical', -0.9, 0.7 >

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. For joins, the property of decreasing degree of interest 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. Consider an implicit selection preference with degree of criticality c_S. For any constituent implicit join with a degree of criticality c_J, the following bound is derived by applying simple mathematics

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

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 the query graph. This is a sub-graph on top of the personalization graph and includes all the nodes corresponding to relations involved in the query (possibly replicated if multiple tuple variables range over them) and all the selection and join edges corresponding to the atomic conditions of the query qualification. For example, in Fig. 4 the query

```
select   title
from     MOVIE M, PLAY P
where    M.mid=P.mid and P.date='28/07/2004'
```

is depicted as a sub-graph in grey color on top of the personalization graph corresponding to Jon's profile.

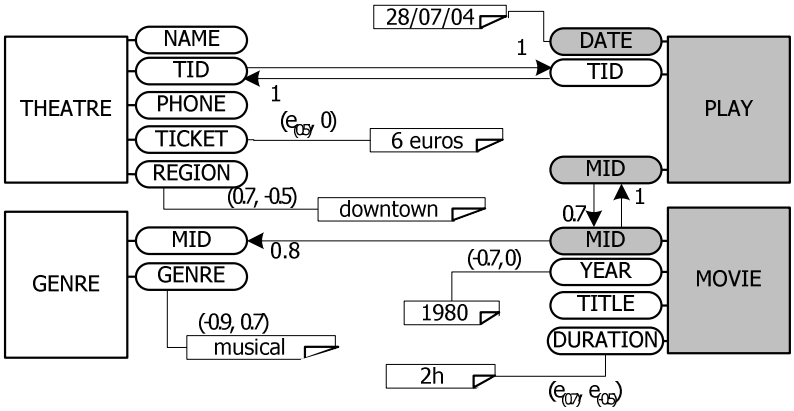

Fig. 4. A query on top of a personalization graph

An implicit preference related to this query is:

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

Parameter K is specified with the use of some criterion. For example, a criterion based on the degree of criticality of preferences, may specify that the top 5 preferences, or preferences with a degree of criticality above a threshold c_0, should be selected; a criterion based on the desired degree of interest in results, may designate results of degree > 0.8.

Problem Formulation. Given the personalization graph G corresponding to a user profile and the query sub-graph on top of this graph representing a query Q, we consider the set P_N of all paths p_i in G that are related to Q in decreasing order of criticality c_i, i.e.,

$$P_N = \{p_i \mid i \in [1, N],\ c_{i-1} \geq c_i\}$$

The set of preferences that may affect the query, based on some criterion $C(.)$ on the degrees of criticality, is the ordered subset $P_K = \{p_i \mid i \in [1, K],\ c_{i-1} \geq c_i\}$ of P_N such that:

$$K = max(\{t \mid t \in [1, N]: C(P_t) \text{ holds}\}).$$

Algorithms. A preference selection algorithm should gradually construct directed paths attached to the query sub-graph on the personalization graph G in decreasing order of criticality. Consider the personalization graph depicted in Fig. 5. 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.

Implicit joins have the property of decreasing degree of criticality as the length of the corresponding path increases. This gives the possibility of a best-first traversal of the personalization graph `G`. In Fig. 5, AB is more critical than AE and, due to the abovementioned property, it is guaranteed that ABD is more critical than AEF.

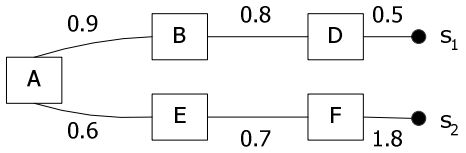

Fig. 5. Example graph with degrees of criticality

Unfortunately, monotonicity is lost for the degree of criticality of implicit selection preferences. Hence, a best-first traversal of the graph does not guarantee that implicit selections are generated in the proper order. Indeed, $ABDs_1$ is *not* more critical than $AEFs_2$.

An implicit selection preference may be safely output only if it is more important than the *most critical selection preference unseen* (*mcsu*). Based on Formula (8), the latter is comprised of the most critical implicit join currently known followed by an atomic selection with a degree of criticality equal 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 that join multiplied by two*. Otherwise, the algorithm should expand that join and examines longer paths stemming from it.

Assuming that the most critical implicit join currently known 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, maintenance 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 cheap alternative is keeping a *fake criticality* `fc`, defined 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` based on the product `c*fc` is now possible. Whenever a selection preference is constructed, it is output immediately. The algorithm, called *FakeCrit*, is outlined below.

It generates the set P_K of top `K` preferences based on some criterion `C`(.). 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 $c_0 > 0$. The algorithm terminates when all top K preferences have been derived.

5. Generation of Personalized Answers

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

(a) Interesting to the user, i.e. it should satisfy (at least) L from the top K preferences.

(b) Ranked based on the degree of interest.

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

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. This sub-query also returns the positive degree of interest of the corresponding preference. If it contains an elastic preference, then the corresponding elastic function provides the degree of interest in each tuple. This approach is adapted from [14], so that it can handle elastic and absence preferences. We will give a representative example, without going into technical details.

Example 6. Suppose Jon submitted a 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 built 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 case.

(*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
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**).We present an algorithm for generation of progressive, personalized, self-explanatory, ranked results, which handles 1–to–many absence preferences more efficiently.

The basic idea is as follows. Preferences are integrated into sub-queries as described in the SPA methodology. 1–to–many absence preferences are integrated as if they were presence ones. Hence, two sets of sub-queries are formed: a set of sub-queries involving presence and 1–to–1 absence preferences, $\mathbf{Q_s}$, and a set of sub-queries involving 1–to–many absence preferences, $\mathbf{Q_a}$. We consider both sets ordered in increasing selectivity, and we use simple histograms to obtain this information.

```
PPA (In: Q, preferences P_K, criterion for L, Out: personalized results)
Begin
1.  R := {}; P_active := P_K; MEDI := r+(P_active); Build Q_S; Build Q_a
2.  Foreach q_i ∈ Q_S
    2.1.  If other preferences in Q_S, Q_a do not satisfy criterion for L then
          2.1.1.  output tuples from R and stop
          end if
    2.2.  execute q_i
    2.3.  Foreach t returned by q_i, t ∉ R
          2.3.1.  execute Q_i^s(t)
          2.3.2.  P_Satisfied:={prefs satisfied by t in results by q_i and Q_i^s }
          2.3.3.  execute Q_1^a(t)
          2.3.4.  A_Satisfied :={preferences satisfied by t in results by Q_1^a }
          2.3.5.  Prefs_Satisfied:=P_Satisfied ∪ A_Satisfied;
          2.3.6.  Prefs_NotSatisfied := P_K - Prefs_Satisfied
          2.3.7.  If t satisfies the criterion for L then
                     calculate d_t;
                     R := add(R, t, Prefs_Satisfied, Prefs_NotSatisfied, d_t)
                  end if
          end for
    2.4.  output all t ∈ R not yet output with d_t ≥ MEDI
    2.5.  P_active := P_active − {preferences in q_i }; MEDI := r+( P_active)
    End for
3.  Foreach q_i ∈ Q_a
    3.1.  If rest of preferences in Q_a do not satisfy criterion for L then
          3.1.1.  output tuples from R and stop
          end if
    3.2.  execute q_i
    3.3.  Foreach t returned by q_i, t ∉ R
          3.3.1.  execute Q_i^a(t); Ids_A:= add(Ids_A, t)
          3.3.2.  Prefs_Satisfied :={prefs satisfied by t in results by Q_i^a }
          3.3.3.  Prefs_NotSatisfied := P_K - Prefs_Satisfied
          3.3.4.  If t satisfies the criterion for L then
                     calculate d_t;
                     R := add(R, t, Prefs_Satisfied, Prefs_NotSatisfied, d_t)
                  end if
          end for
    3.4.  output all t ∈ R not yet output with d_t ≥ MEDI
    3.5.  P_active := P_active − {preferences in q_i }; MEDI := r+( P_active)
    end for
4.  If preferences in queries in Q_a satisfy the criterion for L then
    4.1.  execute Q
    4.2.  Foreach t returned by Q, t ∉ R, t ∉ Ids_A
          4.2.1.  Prefs_Satisfied := {all 1-n absence queries}
          4.2.2.  If t satisfies the criterion for L then
                     calculate d_t;
                     R := add(R, t, Prefs_Satisfied, Prefs_NotSatisfied, d_t)
                  end if
          end for
5.  output remaining tuples from R
End
```

Fig. 6. Algorithm PPA

Sub-queries are executed sequentially starting from the queries in $\mathbf{Q_s}$. A tuple id returned by a sub-query may satisfy one or more preferences, depending on its frequency in the results. For each tuple id, we check whether it is also returned by other sub-queries, thus it satisfies more preferences. We assemble all occurrences of a tuple id and record the preferences satisfied, their number, and the degree of interest in this tuple. A tuple that satisfies the criterion on L, is output based on evidence that no coming tuple could have a degree of interest greater than the degree of the former.

The algorithm is called PPA and is provided in Fig. 6. Its inputs are: the initial query Q, the set of preferences selected from the previous step of query personalization, and an explicit or implicit specification for L. Results of Q that satisfy at least L preferences are accumulated in a list R ordered in decreasing degree of interest. From there, tuples are progressively output when their degree of interest is greater than the degree of any future tuple.

More specifically, consider the two sets of sub-queries formed:

- Each sub-query $q_i \in \mathbf{Q_s}$ identifies tuples that *satisfy* a presence or 1–to–1 absence preference, and returns the tuple id t, the relation attribute and value satisfying the corresponding preference and the associated positive degree of interest.
- Each sub-query $q_j \in \mathbf{Q_a}$ identifies tuples that *do not* satisfy a 1–to–many absence preference, and returns the tuple id t, the relation attribute and value referred in the corresponding preference and the associated negative degree of interest.

First, queries in $\mathbf{Q_s}$ are sequentially executed. For each distinct tuple id t returned by a query $q_i \in \mathbf{Q_s}$, we check whether t is also returned by queries following q_i in $\mathbf{Q_s}$. For this purpose, a parameterized query $Q_i^s(t)$ that is the union of these queries is executed with input parameter the tuple id t. Each occurrence of t in the results returned by $Q_i^s(t)$ and q_i satisfies some (presence or 1–to–1 absence) preference. The relation attribute – value pair returned in each occurrence of t describes the preference satisfied. From them, we build the set of presence or 1–to–1 absence preferences satisfied by t, $P_{Satisfied}$. In addition, for the same tuple id t, a parameterized query $Q_1^a(t)$ that is the union of all queries $q_j \in \mathbf{Q_a}$ is executed. Each occurrence of t in the results of this query corresponds to a 1– to–many absence preference that *is not* satisfied. Since the set of 1– to–many absence preferences is known, we can easily find which of them are satisfied by t. This is the set $A_{Satisfied}$.

Thus, the preferences satisfied by t are the set $Prefs_{Satisfied} := P_{Satisfied} \cup A_{Satisfied}$. The set of preferences not satisfied is easily found: $Prefs_{NotSatisfied} := P_K - \{P_{Satisfied} \cup A_{Satisfied}\}$. Then, the degree of interest d_t in it may be calculated using any ranking function for positive, negative or mixed combinations of preferences.

Parameter L may be specified explicitly or implicitly by providing a minimum degree of interest in the results. If t satisfies the criterion on L, then a result tuple is produced with the following elements:

$\{t, Prefs_{Satisfied}, Prefs_{NotSatisfied}, d_t\}$.

This is placed in the list of results R.

When all queries in $\mathbf{Q_s}$ have been executed, queries in $\mathbf{Q_a}$ are sequentially executed following the same logic as before. For each distinct tuple id t returned by a query $q_i \in \mathbf{Q_a}$, if not already in R, a parameterized query that is the union of all queries

following q_i in $\mathbf{Q_a}$ is executed. A tuple $\{t, Prefs_{Satisfied}, Prefs_{NotSatisfied}, d_t\}$ is constructed and placed in the list of results if it satisfies the criterion on L. In addition, the algorithm keeps a list Ids_A of all tuple ids returned by absence queries. At the end, any tuple of the initial query Q with id not in this list is also part of the results returned by the algorithm.

The list of results, R, is ordered in decreasing degree of interest. A tuple from this list may be output based on evidence that no coming tuple could have a degree of interest greater than the degree of the former. Such evidence is provided in the form of a maximum estimated degree of interest (MEDI) that any unseen result can achieve. This is the degree of interest of the maximal set of preferences that may be satisfied at each point of the algorithm. Initially, this is the whole set of preferences. When the algorithm proceeds with the next sub-query in sequence, the preferences involved in the previous one are not applicable any more, thus they are removed from this set of preferences.

Tuples in R whose degree of interest becomes greater than or equal to MEDI are output. However, they are not removed from R. In this way, the algorithm always knows which tuples have already been encountered and does not create duplicates. Since R is ordered in decreasing degree of interest, there is a pointer to the first tuple there not output yet. As tuples are progressively output this pointer moves towards the bottom of the list.

The algorithm terminates when the preferences involved in the remaining sub-queries do not suffice for satisfying the criterion on L, and it outputs any remaining tuples in R.

6. Experimental Results

Experiments were conducted using a system implemented on top of Oracle 9i. Our data comes from the Internet Movies Database [10] with information about over 340000 films. We conducted several experiments with various sets of profiles and queries. We discuss results of experiments concerning *(a)* the appropriateness of the described ranking functions, and *(b)* the benefits of query personalization.

We conducted an empirical evaluation of our approach with human subjects. Almost half of them have a diploma in computer science, the rest of them being simple users of computers. First, each user provided his 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 queries. Each user submitted these 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. In the second trial, all users were asked to think of a specific need, e.g., to find 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.

Each user was asked to electronically evaluate each tuple returned by a query, and the overall answer to a query. We compared user interest in each tuple to the degree of interest returned by the three positive ranking functions described earlier. Overall

evaluation of the answer was performed by providing: *(a)* an estimation of the degree of difficulty to find something interesting, if they found anything, *(b)* an estimation of how well the answer covered their need, and *(c)* an overall score of the results.

Regarding ranking functions, our results indicated that the inflationary function is not appropriate, because it approaches 1 very quickly, and when it is already close to 1, adding more preferences causes only slight changes to the degree calculated. Almost half of our users followed a dominant philosophy when evaluating the results, while the rest of them followed a reserved philosophy. These results are indicative as to the appropriateness and intuitiveness of the described ranking functions, and have shown that it may be possible to learn for each user the most appropriate ranking function, and store this information in the user profile.

Regarding the effectiveness of personalized queries, 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.

7. Conclusions and Future Work

We have focused on query personalization and we have presented a preference model, efficient query personalization algorithms, ranking functions, and experimental results.

Personalized database information access opens the door to a new set of challenges and opportunities for the future. Combining personal preferences with other aspects of a query's context that call for query customization, such as time of day, user location, device used for querying, etc is certainly an outstanding research challenge in the near future. Furthermore, since query personalization alters the search experience, the user interface needs to provide a way to explain what the system is doing to personalize the experience as well as to undo the personalization. Therefore, an interesting research direction is towards design of user interfaces that allow users to control the extent of the personalization, and can help alleviate inaccurate personalization. Other interesting issues are the expression of preferences over a higher level model that may be transparently mapped to an underlying database's schema, and algorithms for (semi-) automatic construction of user profiles.

8. References

[1] Agrawal, R., Wimmers, E. A Framework for Expressing and Combining Preferences. 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. 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] Fishburn, P. Preference Structures and Their Numerical Representations. *Theor. Comput. Sci*. 217, 359–383, 1999.
[8] Hansson, S. O. Preference Logic. *In Handbook of Philosophical Logic*, D. Gabbay, Ed. Vol. 8, 2001.
[9] Ilyas, I., Shah, R. Aref, W., Vitter, J., Elmagarmid, A. Rank-aware Query Optimization. Proc. of ACM SIGMOD, 2004.
[10] Internet Movies Database. Available at www.imdb.com
[11] Karypis, G. Evaluation of Item-Based Top-N Recommendation Algorithms. Proc. of CIKM, 247-254, 2001.
[12] Kießling, W., Köstler, G.. Preference SQL-Design, Implementation, Experiences. Proc. of VLDB, 2002.
[13] Kießling, W. Foundations of preferences in database systems. Proc. of. VLDB, 2002.
[14] Koutrika, G., Ioannidis, Y. Personalization of Queries in Database Systems. Proc. of ICDE, 2004.
[15] Lacroix, M., Lavency, P. Preferences: Putting More Knowledge into Queries. Proc. of VLDB, 217–225, 1987.
[16] Liu F., Yu C., Meng W. Personalized Web Search by Mapping User Queries to Categories. Proc. of ACM CIKM, 558-565, 2002.
[17] Papadias, D., Tao, Y., Fu, G., Seeger, B. An Optimal and Progressive Algorithm for Skyline Queries. Proc. of ACM SIGMOD, 467–478, 2003.
[18] Pitkow, J., Schutze, H., et al. Personalized Search. *Comm. of the ACM*, 45(9), 2002.
[19] Shahabi, C., Banaei-Kashani, F., Chen, Y., McLeod D. Yoda: An Accurate and Scalable Web-based Recommendation System. Proc. of COOPIS, 2001.
[20] Wellman,M.P., Doyle, J. Preferential semantics for goals. Proc. of the National Conf. on AI, 698–703, 1991.
[21] Zhu, L., Meng W. Learning-Based Top-N Selection Query Evaluation over Relational Databases. Proc. of WAIM, 2004.