ABSTRACT
Due to the growing size of the Web, exploiting user preferences and user profiles to provide personalized information access is becoming an increasingly important issue. Many web sites are supported by underlying database systems; hence there is a growing need for supporting personalization in this context as well. In this paper, we present a generalized model of user preferences for database information, including positive and negative as well as hard and soft preferences. Preference information is stored in user profiles and is used to enhance incoming queries and generate ranked, personalized answers. We describe several algorithms for such query personalization, including algorithms for selecting the most important and relevant preferences as well as algorithms for integrating those preferences into a given user query. Finally, we present the results of several experiments with a prototype personalized query answering system. These have been conducted with both synthetic and actual human profiles, measure both the efficiency of our algorithms as well as the overall effectiveness of our approach, and have given very encouraging results on both accounts.

1. INTRODUCTION
Imagine a buffet offering an extraordinary variety of delicacies. People, most with no particular knowledge of gastronomy, visit the buffet anytime with the intention to satisfy their varying feeding needs. Given the wealth and quantity of goods, finding what best fits one’s preferences and needs becomes a non-trivial task. This imaginary situation resembles information seeking on the Web. Increasing amounts of information are available. A rapidly growing class of untrained lay users “wanders around” trying to locate useful items; this procedure is often tedious and frustrating. Information providers have recognized problems stemming from information overload and have started looking into ways to alleviate them. Towards this direction, several approaches developed take advantage of information related to the user in order to provide customized, personalized answers. This information, such as preferences, is stored in profiles and is used for: results ranking, recommendations, information filtering, continuous querying etc.

Many web-based systems are supported by an underlying database management system, which leads to a growing need for handling user preferences and profiles in the context of database systems. Queries are explicitly or implicitly submitted to the underlying system. Following a personalized information access paradigm, the system would keep information related to each user in a profile. This information may be directly entered by the user or implicitly collected by the system by monitoring user interaction. When a new query arrives, relevant profile information is identified and integrated into it thus providing focused, personalized answers. What information is relevant to a specific request and how it is combined with it affecting the final answer is dynamically determined depending on the query, the profile and the personalization philosophy adopted. Results of a personalized query may be ranked depending on their estimated interest to the user. This approach of dynamically modifying a query by considering user information is called query personalization [15], a relatively new approach to the database and IR fields. It is different from other approaches like information filtering or continuous queries.

Focus of the paper and contributions. In order to make personalized scenarios as the one outlined above possible, user models and algorithms for query personalization are required. To this direction, the contributions of this paper are the following:

- Preference model. We focus on how user preferences on structured information are represented and stored in structured user profiles. Our preference model captures different flavors of preferences, such as likes and dislikes, hard and soft preferences, preferences for the presence or absence of attribute values. It may be used for different purposes, such as ranking of results of general searches, personalization of queries, filtering of information. In this paper, we focus on personalization and result ranking of queries.

- Personalization algorithms. Query personalization is a two-phase approach as described in [15]. Preference Selection deals with the identification of relevant preferences. Preference Integration deals with the construction of the suitable query for fetching the personalized results. We provide algorithms for preference selection and preference integration that cope with different types of preferences in our model. Negative preferences have been the key challenge in achieving the above.

- Personalized Query Execution. We describe a simple approach to fetch personalized results based on the preferences selected and we discuss its shortcomings. Then, we provide a more elaborate technique that allows the system to return more quickly results ranked, and moreover, justify its answers based on the preferences satisfied.

- Experiments. We have implemented a personalized query answering system and we have conducted synthetic experiments and experiments with users in order to evaluate the efficiency of the models and algorithms. (Semi-) automatic preference
elicitation and management, which are also important issues in a personalized setting, are out of the focus of this paper. In our experiments, users specify manually their preferences.

2. RELATED WORK

In this section, we present related work using the following axes: content selection approaches, user profiles, preferences and ranking.

Content selection approaches are roughly of three types. Traditional Database and IR systems follow a query-based approach, i.e., content is selected on the basis of a query issued. As a result, all users issuing the same query are provided with the same answer. User preferences are taken into consideration by the majority of IR systems for results ranking. Two recent lines of database research inspired from IR, namely keyword searches [2, 4, 11] and best-match query answering [1, 12, 9, 14] follow the same approach. In filter-based approaches, content selection is driven by a long-term information need stored in a user profile, which is considered as a form of continuously executing query [6, 16]. Personalized approaches that combine the query issued with user preferences and other aspects of the user’s context to focus the search have become to emerge [18, 15]. We build on the personalization framework presented in [15]. We present a generalized model for representing and storing several types of preferences in profiles and we provide algorithms for generation of personalized answers that are appropriate for manipulation of these new preference types.

User profiles have been broadly used for information filtering [5] of text-based data items and they typically represent user interests in terms of a single or multiple term vectors. Profiles have also been used for providing data management hints for preloading and pre-staging caches in distributed environments [17, 7]. In our work, we use profiles for storing user preferences and use them for individualizing an incoming user request based on the framework presented in previous work [15]. Preferences can be of different kinds [1, 9, 14]: positive, negative, hard, soft etc. Our preference model captures these, as well as preferences on associations of objects not explicitly defined within existing frameworks.

Ranking of results may be performed in several ways: results are ordered based on the number of joins they involve [2] or how closely they match a user’s soft preferences [1, 12]. In our work, we provide a mechanism for ranking results based on which preferences are satisfied and which are not. In addition, we provide an algorithm that generates results ranked based on the combination of different types of preferences captured by our preference model.

3. PREFERENCE MODEL

Without loss of generality, we focus on SPJ (Select-Project-Join) queries over relational databases. In particular, we focus on queries, whose qualification is a combination of disjunctions and conjunctions of atomic selection and join conditions, producing a result from atomic projections of attributes. In the following subsections, we provide a formal description of the preference model, starting from the simplest of conditions and building up.

We use $R, S, \ldots$ to denote relational tables, $RA, RB, \ldots$ to denote attributes, and $DR_A, DR_B$ their corresponding domains of values.

For our examples, we use a movies database described by the schema below; primary keys are underlined.

\[
\begin{align*}
&\text{TANDEM(tid, mid, region, ticket)} \\
&\text{PLAY(tid, mid, date)} \\
&\text{MOVIE(mid, title, year, duration)} \\
&\text{GENRE(mid, genre)} \\
&\text{CAST(mid, aid, award, role) \quad \text{ACTOR(aid, name)}} \\
&\text{DIRECTED(mid, did), \quad \text{DIRECTOR(did, name)}}
\end{align*}
\]

3.1 Stored Atomic Preferences

Our approach to personalization is based on maintaining, for every user, a user profile whose structure is intimately related to the features of the data and query models. In particular, we assume that user preferences are stored at the level of atomic elements of queries, i.e., atomic selection or join conditions, which are therefore called atomic user preferences.

Preferences on selection conditions are quite involved as they essentially express preferences on the values that may appear in the selections, giving rise to a wide range of possibilities. We present these by distinguishing three relevant dimensions.

(a) Positive/Negative Preferences: A user's interest in the values of an attribute is expressed in the form of a degree of interest (doi) function $f_{\text{doi}}: D_{RA} \rightarrow [-1, 1]$. A doi equal to 0 indicates lack of any user interest in the domain value (not stored in the profile), positive doi's indicate increasingly higher interest (doi=1 is for 'must-have' domain values), while negative doi's indicate increasing disinterest (doi=-1 is for 'most-unpleasant' domain values).

(b) Presence/Absence Preferences: With respect to preferences, a value $v$ in domain $D_{RA}$ may be seen in two roles: as an individual or as a member of a set. As defined, the doi function $f_{\text{doi}}(v)$ captures a user's interest in the presence of $v$, whether as the value of $RA$ (or any other single-valued path of the schema leading to $RA$) or in the result of traversing a multi-valued path of the schema. Users, however, have a distinct degree of interest in the absence of $v$ as well. This does not arise when values appear as individuals, as the absence of one is always due to the presence of another. As set members, however, values are mutually independent with respect to their presence or absence, e.g., N. Kidman's starring in a movie is not affected by the fact that J. Roberts is not in it. Furthermore, the doi in the absence of a value from a set is not derivable from the doi in its presence, e.g., strong interest in W. Allen could be combined with indifference or with strong negative interest in his absence. Hence, a second degree of interest function $f_{\text{fio}}: D_{RA} \rightarrow [-1, 1]$ is introduced to capture a user's doi in the absence of a domain value from sets, with the same interpretation and treatment of the values in [-1, 1] as before. As psychological evidence indicates [10], for normal users, the following holds:

\[
f_{\text{fio}}(v)f_{\text{doi}}(v) \leq 0, \quad \forall v \in D_{RA}
\]

As a matter of notation, we use $\langle P, f_{\text{fio}}, f_{\text{doi}} \rangle$ to denote the two degrees of interest in a selection condition $P$.

(c) Hard/Soft Preferences: In principle, none of the above depends on whether the domain $D_{RA}$ is categorical or numeric. Nevertheless, humans approach the two types differently. On one
hand, given the mutual independence of categorical values, preferences for those are considered hard and are either satisfied exactly or not at all. For example, preferences for ‘adventures’ and ‘comedy’ are two unrelated issues. On the other hand, preferences for numeric values may be smoothly continuous over their domain, in which case, they are considered soft and may be satisfied approximately. For instance, a preference on movies with duration 2 hours is soft, as movies of 122 or 115 minutes are probably of similar interest to the user.

A soft preference on an attribute $A$ is expressed in the form of the following pair of doi functions $(L (R))$ is monotonically increasing (decreasing) and upper semi-continuous).

$$f_p(u) = \begin{cases} L_p(u, d_p) & \forall u \in [a - m_1, a] \\ d_p & \forall u \in [a, b] \\ R_p(u, d_p) & \forall u \in [b, b + m_2] \end{cases}$$

$$f_a(u) = \begin{cases} L_a(u, d_a) & \forall u \in [a - m_1, a] \\ d_a & \forall u \in [a, b] \\ R_a(u, d_a) & \forall u \in [b, b + m_2] \end{cases}$$

D.o.i. functions for soft preferences resemble membership functions from fuzzy set theory [19]. There are many possible representations to express gradual set membership in fuzzy sets. We have adopted a parametric representation [3] and adapted it to provide a degree of interest in $u$ depending on its membership in a set of interesting values.

<table>
<thead>
<tr>
<th>Preferences</th>
<th>Positive/ Negative</th>
<th>Presence/ Absence</th>
<th>Hard/ Soft</th>
</tr>
</thead>
<tbody>
<tr>
<td>I like actor W. Allen</td>
<td>Positive</td>
<td>Presence</td>
<td>Hard</td>
</tr>
<tr>
<td>I don’t like theatres located downtown</td>
<td>Negative</td>
<td>Presence</td>
<td>Hard</td>
</tr>
<tr>
<td>I like movies without violence</td>
<td>Positive</td>
<td>Absence</td>
<td>Hard</td>
</tr>
<tr>
<td>I prefer films with a duration around 2h</td>
<td>Positive</td>
<td>Presence</td>
<td>Soft</td>
</tr>
</tbody>
</table>

Example. Clearly, all $(2^3 - 8)$ combinations of the above choices of preference types are valid. These combinations allow for the expression of various user preferences. Table 1 gives some characteristic examples. Our earlier work [15] only dealt with hard positive presence preferences, which constitute one of the $8$ combinations which are possible given the generalized preference model described above. Therefore, a user could only express preferences like the one recorded in the first row of the example.

Compared to selection conditions, preferences on join conditions are much simpler as they do not lend themselves to any of the variations mentioned above. They are bound to be non-negative presence hard preferences, as there are no numerical values or sets involved. Hence, a user’s interest in a join condition is expressed in the form of a degree of interest, which is a real number in the range $[0, 1]$, with the same interpretation and treatment of the doi values as before. For reasons that will become clear later, a user may express two separate degrees of interest in a join, one for each potential direction of traversal. As a matter of notation, for a directed join condition $P$, we use $<P, d>$ to denote a degree of interest $d$ in it.

Example. Julie has several preferences regarding movies and theatres. She is happy if the movie is not a thriller and she likes comedies very much. She prefers watching films with duration around 2h. She is a fan of actors Al Pacino and R. De Niro but she does not like J. Carey. She prefers only downtown theatres. She would pay around 6 Euros for watching a movie. These are expressed as degrees of interest in specific atomic selections. Moreover, Julie considers information about movies when inquiring on theatres more important than the other way around. She considers the director more important than the cast of a movie or the theatre playing it. The genre of a movie is almost as important as its director and so on. All of her preferences are stored in her profile, as depicted in Figure 1, with each condition followed by the corresponding $f_p$ value and then, if appropriate, the corresponding $f_a$ value. These preferences may evolve over time. Thus, Figure 1 illustrates an instance of Julie’s profile at a given point in time. The query personalization process is not affected by profile changes, since it automatically integrates recorded preferences in user requests.

```
[ THEATRE.tid=PLAY.tid, 1 ]
[ PLAY.tid=THEATRE.tid, 1 ]
[ PLAY.mid=MOVIE.mid, 1 ]
[ MOVIE.mid=PLAY.mid, 0.8 ]
[ MOVIE.mid=GENRE.mid, 0.9 ]
[ MOVIE.mid=DIRECTED.mid, 1 ]
[ DIRECTED.did=DIRECTOR.did, 1 ]
[ THEATRE.region='downtown', 0.7, -0.5 ]
[ THEATRE.ticket='6 Euros', fp(0.5), 0 ]
[ ACTOR.name='Al Pacino', 0.8 ]
[ ACTOR.name='J. Carey', -0.7, 0 ]
[ MOVIE.duration=120min, fp(0.7), fa(-0.5) ]
[ GENRE.genre='comedy', 0.9] 0]
[ GENRE.genre='thriller', -0.9, 0.7 ]
```

Figure 1. Part of the stored profile for Julie

Note that two distinct entries (rows 3 and 4) are stored in her profile for the same join between relations MOVIE and PLAY but with different degrees. The left part of a join corresponds to the relation already included in the query and the right one to the relation that will be added if the join is integrated into the query.

A particular user’s preferences over the contents of a database can be expressed on top of the personalization graph of the database [15]. This is a directed graph $G(V, E)$ ($V$ is the set of nodes and $E$ is the set of edges) that is an extension of the traditional schema graph. There are three types of nodes in $V$:

- **relation nodes**, one for each relation in the schema
- **attribute nodes**, one for each attribute of each relation in the schema
- **value nodes**, potentially one for each possible value of each attribute of each relation in the schema. In essence, only those that have any interest to a particular user need to be specified

Likewise, there are two types of edges in $E$:
• **selection edges**, from an attribute node to a value node; such an edge represents the potential selection condition connecting the corresponding attribute and value. Selection edges corresponding to hard preferences are depicted differently than those corresponding to soft preferences
• **join edges**, from an attribute node to another attribute node; such an edge represents the potential join condition between the corresponding attributes. These could be joins that arise naturally due to foreign key constraints, but could also be other joins that are meaningful to the designer. As indicated earlier, two attribute nodes could be connected through two different join edges, in the two possible directions

### 3.2 Transitive Preferences

By composing atomic user preferences that are adjacent in the personalization graph (hence, composable), one is able to build **transitive user preferences**, i.e., preferences expressed through relationships. Given the one-to-one mapping between edges in the personalization graph and atomic query elements, a transitive user preference is mapped to a directed path in the personalization graph. In particular:

- A **Transitive join preference** is mapped to a path in the personalization graph between two attribute nodes. The path is comprised of composable atomic join edges and represents the potential “implicit” join condition between the corresponding attributes.
- A **Transitive selection preference** is mapped to a path in the personalization graph from an attribute node to a value node. Such a path is comprised of n-1 atomic join edges and one selection edge and represents the potential “implicit” selection condition connecting the corresponding attribute and value. That is, a transitive selection is the combination of a transitive join and an atomic selection that are composable.

A transitive query element is defined as the conjunction of the constituent atomic ones. The degree of interest in a transitive preference should be a function \( f_p \) of the degrees of interest in the participating preferences. Consider a set of composable preferences comprised of \( n \) atomic joins and one atomic selection:

\[
(P_1, d_1) < (P_2, d_2) < \ldots < (P_m, d_m) < (P_n, f_p) >
\]

Then, the transitive join is defined as follows:

\[
(P_1 \ldots P_n, f_o(d_1, \ldots d_n))
\]

The transitive selection is defined as follows:

\[
(P_1 \ldots P_n, f_p(d_1, \ldots d_m, f_o))
\]

In principle, one may imagine several functions for \( f_o \). All of them, however, should satisfy the condition that the absolute doi in a transitive preference decreases as the length of the corresponding directed path increases, capturing human intuition and cognitive evidence [10]. Thus, we have chosen multiplication as the function \( f_o \):

\[
f_o(d) = d_1 d_2 \ldots (3)
\]

This function essentially approaches 0 as more and more preferences are added to the combination.

**Example.** Julie likes the actor R. De Niro, which is expressed as a preference for the selection:

\[
\text{MOVIE.mid} = \text{CAST.mid} \text{ and } \text{CAST.aid} = \text{ACTOR.aid} \text{ and } \text{ACTOR.name} = 'R. \text{ De Niro}'
\]

Then, she also likes movies starring the same actor, expressed as an implicit preference for the condition:

\[
\text{MOVIE.m} = \text{CAST.} \text{mid and CAST.aid} = \text{ACTOR.aid and ACTOR.name} = 'R. \text{ De Niro}'
\]

The degree of interest associated with the corresponding transitive preference is the product of the degrees of the constituent conditions, which based on her profile, gives \( 0.8 \times 0.7 = 0.56 \).

Note that any directed path in the personalization graph could map to a transitive preference. However, based on human intuition and cognitive evidence [10], we deal with acyclic paths only. It is rather unlikely and unnatural that a cyclic directed path would express a confirmed user preference. Moreover, cycles have termination problems.

### 3.3 Combination of Preferences

**Satisfaction** of an (atomic or transitive) preference \( <P_i, d_i > P_j > \) is equivalent to satisfaction of \( P_i \) if \( d_i \geq 0 \) or failure of \( P_j \) if \( d_j \geq 0 \).

**Failure** to satisfy a preference is, of course, the exact opposite. For example, the preference

\[
[ \text{ACTOR.name} = 'J. \text{ Carey}', -0.7, 0]
\]

is satisfied by movies in which J. Carey does not star. For convenience, we use \( d^+ \) to denote the degree of interest in the satisfaction of a preference \( P \) and \( d^- \) to denote the degree of interest in the failure of \( P \).

Consider a set \( P \) of \( N \) preferences and the set \( D \) of the corresponding satisfaction (non-negative) doi's:

\[
D = \{d^+ | d^-; \text{ degree of interest in } P \in P, i = 1 \ldots N\}
\]

The doi in the conjunction of preferences should be a function of the degrees of interest in the participating preferences. In principle, one may imagine several such functions, the appropriateness of each one being judged only by the philosophy of the approach taken towards personalization and, more importantly, by how closely it reflects human behavior. A parameter that appears pivotal in this issue is \( \text{max}(D) \). Around it, one may see three different philosophies:

- **Inflationary**: the doi in multiple preferences satisfied together increases with the number of these preferences, i.e., \( f^+(D) \geq \text{max}(D) \). One such example in this category is the following function:

\[
f^+_1 = 1 - \prod_{i=1}^{N} (1 - d^+_i) \quad (4)
\]

- **Dominant**: the doi in multiple preferences satisfied together is exactly equal to the doi of the most interesting of these preferences, i.e., \( f^+(D) = \text{max}(D) \). This is inspired by the treatment of conjunctions in fuzzy sets and captures a ‘winner-takes-all’ philosophy.

- **Reserved**: the doi in multiple preferences satisfied together is between the highest and the lowest degrees of interest among the original preferences, i.e., \( \text{min}(D) \leq f^+(D) \leq \text{max}(D) \). One such example in this category is the following function:

\[
f^+_2 = 1 - \prod_{i=1}^{N} (1 - d^+_i)^{1/N} \quad (5)
\]
Each one of the above approaches and functions may be appropriate for different situations or types of preferences. We have experimented with particular choices among them and the results shed some light as to intuitiveness of each one.

The above analysis addresses the degree of interest in satisfying multiple preferences. A similar issue arises with respect to the degree of interest in failing to satisfy multiple preferences, i.e., dealing with multiple non-positive dois in a set $D_c$. The two cases are symmetric with each other and may be treated in exactly the same fashion. The pivotal parameter is $\min(D_c)$ here and one may define inflationary, dominant, and reserved combination functions. We have worked with the counterparts of $f_1$ and $f_2$ above, which are exactly the same, only with an exchange of the ‘+’ and ‘-’ sign everywhere.

In the general case, one deals with satisfying a set $P_s$ of $N_c$ preferences and fails to satisfy a set $P_f$ of $N_f$ preferences. The doi in this case is a function of the degrees of interest in the two sets satisfying the followings conditions:

$$f^-(D_-) \leq f(D_s, D_-) \leq f^+(D_s) \tag{6}$$
$$f(d_p, d_a) = 0 \tag{7}$$

Among several possible examples of such functions, we have worked with the following:

$$f_1 = f^+ + f^- \tag{8}$$
$$f_2 = \frac{N_c f^+ + N_f f^-}{N_c + N_f} \tag{9}$$

Formula (9) proved to be more appropriate as it weighs the doi provided by each of the participating functions by the number of preferences that support this doi. In other words, it captures the intuition that the overall doi in some results should be affected not only by the doi in preferences satisfied and the doi in those failed but also by the number of preferences satisfied and preferences failed.

### 3.4 Preference Order

Given a set of selection preferences, it is useful to order them based on their importance or criticality with the intention of selecting which ones to satisfy and which ones not. Such ordering should take into account both degrees of interest $d_p$ and $d_a$ (for presence and absence, respectively). Intuitively, the most important/critical preference should be one whose satisfaction has a very positive doi, while its failure has a very negative doi. Accordingly, we define the degree of criticality $c$ of a preference $P$ as a real number in the range $[0, 2]$ given by the following formula:

$$c = |d_p| + |d_a| \tag{10}$$

Criticality can be extended to join preferences by assuming the doi in their failure as being equal to 0. In that sense, for joins, the property of the degree of interest always decreasing as the length of the corresponding path increases transfers over to the degree of criticality as well.

Unfortunately, the same does not hold for transitive selection preferences. Consider a set of composable preferences comprised of $n$ joins and one selection:

$$<P_1, d_1>, <P_2, d_2>, \ldots, <P_n, d_n>, <P_0, d_p, d_a>$$

Then the degree of criticality $c_j$ of the maximal join is

$$c_j = d_1^* d_2^* \ldots d_n^* \tag{11}$$

while the degree of criticality of the selection is

$$c_s = |d_1| \ldots |d_n| + |d_0^* d_1^* \ldots d_n^*| =
\leq d_1^* d_2^* \ldots d_n^* (|d_0^*| + |d_0|) \tag{12}$$

Since $|d_0^*| + |d_0| \geq 1$ could hold, monotonicity is lost for the degree of criticality of selections.

On the other hand, since $|d_0^*| + |d_0| \leq 2$, (12) gives us the following bound, which we use in our algorithms:

$$c_s \leq 2 * c_j \tag{13}$$

### 4. QUERY PERSONALIZATION

Given a query $Q$ and a user profile $U$, a personalized query is built using the personalization framework presented in [15]. A personalized query is built with the use of the following parameters: (i) the number $K$ of top preferences derived from the user profile that should affect the query, and (ii) the number $L$ ($L \leq K$) of those preferences that should at least be met by the results. Parameters $K$ and $L$ can be specified directly by the user or derived based on various criteria on the context, such as the time of the request, the user location, the device of access, etc. Analysis of aspects comprising the query context is out of the scope of this paper.

Query personalization proceeds in three stages: (Preference Selection) The top $K$ preferences recorded in the user profile are identified and extracted. (Preference Integration) The $K$ selected preferences are integrated into the query producing one that will return results satisfying $L$ of them. (Query Execution) The personalized query is executed and returns a ranked list of results based on their interest.

### 5. PREFERENCE SELECTION

The first step of the personalization process deals with the extraction of the top $K$ preferences from a profile, which are related to and not conflicting with the user query. A preference may be related or conflicting at a syntactic or semantic level. Our prototype system currently supports the former level. A preference is syntactically related to a query, if it maps to a path attached to the sub-graph representing the query. A query can be represented as a sub-graph on top of the personalization graph. This sub-graph includes all the nodes corresponding to relations that participate 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. An example of a transitive preference syntactically related to a query about movies is this one.

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

A preference is syntactically conflicting with a query, if it is conflicting with a condition already there.

For instance, the preference THEATRE.region='downtown' cannot be included in a query containing the condition
sub-graph on top of this graph, we consider the set

$\text{Problem Formulation}$

5.1 Selection Based on Criticality

$\text{Problem Formulation.}$ Given the personalization graph $G_P$ corresponding to a user profile and a query $Q$ represented as a sub-graph on top of this graph, we consider the set $P_N$ of all paths $P_i$ in $G_P$ that are related to but not conflicting with $Q$ in decreasing order of their degree of criticality, 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 selection criterion $C(.)$ on the degrees of criticality, is an ordered subset $P_K = \{ P_i \mid i \in [1, K], \ c_{i_1} \geq c_i \}$ of $P_N$ such that:

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

$\text{Algorithms.}$ Consider the personalization graph depicted in Figure 2. For simplicity, attributes involved in joins and selections are omitted. We are interested in finding the most critical transitive preference on $A$. Suppose we perform a best-first traversal of the graph on the basis of the degree of criticality starting from node $A$. Path $AB$ is more critical than $AE$ and it is expanded to include node $D$. Path $ABD$ is more critical than $AE$; then, it is more critical than $AEF$ as well, since the degree of criticality of a transitive join always increases as more join edges are included into it. This is not the case with transitive selections. Transitive selection $ABD_5$ is more critical than $AEF_5$ but it is less critical than $AEF_5$. Thus our naive best-first search would erroneously decide on $ABD_5$ as the most critical one.

Figure 3 presents a preference selection algorithm, $SPS$ that deals with this problem. The algorithm gradually constructs preferences related to a given query in decreasing order of their degrees of criticality. It outputs a selection preference only if its degree of criticality is at least equal to the degree of criticality of the most critical transitive join currently known multiplied by two. This product is the maximum degree of criticality that any selection path stemming from this transitive one may have.

More specifically, a queue $QP$ of candidate preferences is kept in order of decreasing degree of criticality. Initially, it contains all atomic preferences related to the query. This queue has two pointers; one pointing at the top element $P$ with degree of criticality $c_P$, and the other pointing at the top join $P_J$ with degree of criticality $c_J$. These may coincide.

$\text{Simple Preference Selection Algorithm-SPS}$

<table>
<thead>
<tr>
<th>Input</th>
<th>User profile $U$, user query $Q$, selection criterion $C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Set of preferences $P_K$</td>
</tr>
</tbody>
</table>

1. $\text{Foreach atomic preference } AC \in U \text{ related to } Q$

2. If $\text{(AC does not conflict with Q) then}$

3. $\text{QP} \leftarrow AC$

4. $\text{If } (P_J = \text{nil} \text{ and } (AC_i \text{ is join) then } P_J = AC_i \text{ End if}}$

5. $\text{End if}$

6. $\text{End for}$

7. $\text{While (QP not empty) and (K_Selected=false) then}$

8. $\text{Get head } P \text{ from QP}$

9. If $\text{(P is selection) Then}$

10. $\text{K_Selected=C(P_K U \{P\})}$

11. $\text{If (K_Selected=false) then}$

12. $\text{if } ((cP > 0) \text{ or } (P_J = \text{nil}) \text{ or } (c_J \geq 2* c_P)) \text{ then}$

13. $\text{QP} \leftarrow P$

14. $\text{Else set } P=P_J \text{ End if}$

15. $\text{End if}$

16. $\text{End if}$

17. $\text{End while}$

18. $\text{Output } P_K$

Figure 3. Simple Preference Selection algorithm

In each round, the algorithm picks from $QP$ the head $P$. Depending on the type of preference, we distinguish two cases:

$\text{P} \text{ is a selection satisfying the criterion } C(P_K U \{P\}). \text{ If the criterion specifies a minimum degree of criticality } c_P > 0, \text{ then the preference is output. The same occurs when there are no joins in } QP, \text{ or } c_F \geq 2* c_J \text{ holds. Otherwise, } P \text{ is kept in } QP \text{ for later re-examination and the top join } P_J \text{ becomes the next candidate preference to be examined } (P=P_J)$.

$\text{P} \text{ is a join satisfying the criterion } C(P_K U \{P\}). \text{ Then, it is expanded into longer paths which are added into } QP, \text{ as described below. A new path } P \land AC_i \text{ is generated for each atomic preference } AC_i \text{ that is composable with } P. \text{ These atomic preferences are considered in order of decreasing degree of criticality. A new path is pruned, i.e., not inserted in } QP, \text{ under the following conditions: } (i) \text{ it expands to a relation included into}$
Theorem 1: The algorithm SPS processes preferences related to a query \( Q \) in decreasing order of degree of criticality, and it generates the set \( P_K \) of the top \( K \) preferences according to the selection criterion.

<table>
<thead>
<tr>
<th>Preference Selection Algorithm-FakeCrit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
</tr>
<tr>
<td><strong>Output:</strong></td>
</tr>
</tbody>
</table>

\[
P_K = \{ \}, Q_P = \{ \}, K_{Selected} = false
\]

1. **Foreach** atomic preference \( AC \in U \) related to \( Q \)
   1.1 If \( (AC) \) does not conflict with \( Q \) Then \( QP \leftarrow AC \), 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
       \[ K_{Selected} \leftarrow C(P_K \cup \{P\}) \]
       If \( (K_{Selected} = false) \) then \( P_K \leftarrow P \), End if
       End if
   2.3 If \( (P \) is join) Then
       \[ K_{Selected} \leftarrow C(P_K \cup \{P\}) \]
       If \( (K_{Selected} = false) \) then
           **Foreach** atomic element \( AC \in U \) composable with \( P \)
           Prune=false
           If \( ((AC) \) joins to relation \( R \in P \) or \( R \in Q \)) or \( (P \land AC\), conflicts with \( Q \)) \) Then Prune=true, End if
           If \( (c_0 = 0) \) and \( (c_{PA} c_{PA} < c_0) \) Then exit For End if
           If Prune=false then \( QP \leftarrow P \land AC \), End if
           End for
       End if
       End if
   End while

3. **Output** \( P_K \)

Figure 4. FakeCrit Preference Selection algorithm

We now re-examine the way SPS decides whether output of a transitive selection is performed or postponed. The algorithm makes an estimate of the degree of criticality of the most critical preference unseen and compares it to the degree of the preference under consideration. The former is comprised of the most critical transitive join currently known followed by an atomic selection with a degree of criticality equal to two, thus it has a degree of criticality double the degree of criticality of this join. Consequently, SPS relies on a worst-case estimate. This results in conservative decisions on output of preferences. The algorithm for preference selection would be more efficient, if the real maximum degree of criticality of all paths stemming from each join was known. This degree would be kept as extra information on every join edge of the personalization graph. For this purpose, a preprocessing step is needed: for each join edge, all subsequent paths should be visited in order to find the maximum degree of criticality. This process is not very cheap. In addition, 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. For this reason, instead of keeping the exact maximum degree of criticality, we keep an estimate of it, called fake degree of criticality \( c_f \), defined as follows:

If the edge is a selection, then \( c_f \) is set to 1 by default. If it is a join, then \( c_f \) equals to the maximum degree of criticality of all edges that are adjacent to it. If one of those is a join, then consider its degree of criticality multiplied by 2.

Based on this approximation, preprocessing and updating are more efficient.

In order to find the most critical path, we calculate for each join edge the product \( c^*c_f \). The maximum product corresponds to the most critical preference unseen. The algorithm FakeCrit that exploits the fake degree of criticality is presented in Figure 4. Again, there is a queue \( QP \) of candidate preferences. However, it is kept in order of decreasing product of degree of criticality and fake degree of criticality. In each round, the algorithm picks the head \( P \) from \( QP \). If \( P \) is an atomic or transitive selection and it satisfies the criterion \( C(P_K \cup \{P\}) \), it is immediately expanded. If \( P \) is a join then it is expanded. A new path \( P \land AC \) is pruned under the same circumstances (i)-(ii) as in SPS. In addition, if preferences selected must have a degree of criticality greater than \( c_0 \), then the new path is pruned when the product of its degree of criticality and its fake degree of criticality \((c_{PA}c_f c_{PA}) < c_0 \). In this case, the algorithm stops expansion of \( P \). A theorem analogous to the previous ones holds for this algorithm as well.

5.2 Selection Based on the d.o.i. of Results

Consider a user profile consisting of these preferences.

We are interested in movies with an estimated doi at least equal to 0.8. Then, we can choose to satisfy only the preference on J. Nicholson and ignore the rest. Movies with J. Nicholson and D. Keaton have an estimated doi higher than 0.8. However, movies with J. Nicholson and A. Sandler have an estimated doi below 0.8. According to our criterion on movies, these should not have been retrieved. Consequently, if we are interested in results with a desired doi greater than a certain threshold, then we must take into consideration preferences with negative degrees of interest. An appropriate preference selection algorithm should select \( K \) preferences, such that any preference ignored will not lower the doi of results below the desired threshold.

Problem Formulation. Given the personalization graph \( G_P \) corresponding to a user profile and a query \( Q \) represented as a sub-graph of this graph, we consider the set \( P_K \) of all paths \( P_i \) in \( G_P \) that are related to but not conflicting with \( Q \) in decreasing order of their degree of criticality, i.e.,

\[
P_K = \{ P_i \mid i \in [1, N], \ c_{PA} \geq c_i \}
\]
The set of all preferences that must be satisfied so that any tuple satisfying those will have a degree of interest greater than or equal to a threshold $d_k$, is an ordered subset $P_k = \{P_i | i \in [1, K], c_{i, t} \geq c_i \}$ of $P_s$ such that:

$$K = \min \{t \mid t \in [1, N]: f(d_1^t, d_2^t, \ldots d_t^t, d_{t+1}^t, \ldots d_N^t) \geq d_k \}.$$ 

**Algorithms.** An exhaustive algorithm would enumerate all paths in $P_k$ and repeat the calculation $f(d_1^t, d_2^t, \ldots d_t^t, d_{t+1}^t, \ldots d_N^t)$ for each $t = 1 \ldots N$ until this function returns a degree greater than or equal to $d_k$. This is clearly inefficient. We describe an algorithm that is built on the algorithms presented in the previous subsection for selecting the most critical preferences and avoids both exhaustive enumeration of all paths as well as repetitive calculations using $f$.

At each step, $t$ preferences are selected. The problem is how to compute $f(d_1^t, d_2^t, \ldots d_t^t, d_{t+1}^t, \ldots d_N^t)$, without visiting the remaining $N-t$ paths. In this case, the exact number $N$ and the doi of the preferences not yet seen would be unknown. Recall that any of the degrees of interest associated with a path decreases in value as the length of the path increases. Then the maximum doi of any preference unseen is estimated based on the following:

$$d_w = \max(\{|d_t^i| \text{ s.t. } P_t \in QP \text{ and } P_i \text{ is selection}\} \cup \{d_t \text{ s.t. } P_t \in QP \text{ and } P_i \text{ is join \}})$$

$$|d_t^i| \leq d_w \forall P_t, \quad k = t+1, \ldots N$$

Then, for the estimation of the doi’s of the $N-t$ preferences unseen, we may consider the worst case scenario, that is:

$$d_k^t = d_w \forall P_t, \quad k = t+1, \ldots N$$

In addition, at each step, the doi in results satisfying $t$ preferences calculated by $f(d_1^t, d_2^t, \ldots d_t^t, d_{t+1}^t, \ldots d_N^t)$ is cached and re-used in the next round. As far as the number $N$ is concerned, this may be considered equal to the size of profile, i.e., $N = \text{size}(U)$. Whether this estimate is close to the real value of $N$ depends on the structure of the personalization graph. If $N << \text{size}(U)$, then the algorithm may enumerate all $N$ paths. This is not very time consuming, since $N$ is relatively small compared to the profile size.

The algorithm outlined above selects more preferences than actually needed for retrieving results with a degree of interest greater than a certain value. Its advantage is that it is cheap and needs no extra pre-processing.

### 6. GENERATION OF PERSONALIZED ANSWERS

Preferences selected from the previous step are integrated into the query initially submitted by a user in order to generate results personalized. These should be:

- **Interesting** to the user, i.e., they should satisfy at least $L$ preferences from the $K$ selected
- **Ranked** based on their estimated degree of interest, so that the most interesting results should come first followed by these that are less interesting to the user
- **Self-explanatory** with respect to the reasons justifying their retrieval and ranking. For this purpose, each returned tuple should be accompanied by the preferences it satisfies.

Preferences selected may be hard or soft. In the latter case, these must be translated into appropriate range conditions that will retrieve approximate results with respect to the values preferred. We will explain how this translation is performed using a set of three preferences. $P_1$ and $P_2$ are hard preferences with degrees of interest that are equal to 0.9 and 0.6, respectively; $P_3$ is a soft preference with a maximum doi equal to 0.7. This is illustrated in Figure 5. $P_1$ is translated into a range condition mapping to the gray region. This condition returns values that are at least as interesting as those satisfying the least interesting preference $P_3$.

Analogous mappings are performed when a minimum degree of criticality has been provided for preference selection, and when we are interested in results with a doi greater than a certain threshold.

![Figure 5. Translation of soft preferences](image)

We now proceed into the description of two approaches for generation of personalized results. For purposes of illustration, consider that a user has issued the following simple query:

```sql
select title
from MOVIE
```

**Construction of a single query.** This approach divides preferences into groups. Each group is integrated into a single sub-query, as it will be described shortly. The personalized results are obtained by taking the union of the partial results, grouping by the projected attributes, and excluding all groups containing less than $L$ rows. Preferences are grouped as follows.

- **Presence preferences comprised of a common transitive join and atomic selection conditions on the same table are grouped together and integrated into a single sub-query.** This sub-query is built on the initial query which is extended by including a complex qualification. This consists of the common transitive join and a disjunction of all atomic selection conditions of the participating preferences. If these are conflicting to each other, then a tuple may be returned by this sub-query at most once. Otherwise, a tuple may be returned as many times as the number of individual preferences. As an example, consider these preferences that are related to the afore-mentioned initial query:

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

These are combined into the following sub-query:

```sql
select title
from MOVIE M, GENRE G
where M.mid=G.mid and
(G.genre='comedy' or G.genre='action')
```

- A combination of conflicting presence and absence preferences comprised of a common transitive join and atomic selection conditions on the same attribute are grouped together
and integrated into a single sub-query. Absence conditions are inserted as presence conditions by changing the operator specified. If a presence preference returns values that are also returned by an absence one, then the former may be omitted from the qualification of the query and considered only for purposes of ranking. As an example, consider these preferences

```
MOVIE.year=2003
Not(MOVIE.year<1980)
```

These are combined into the following sub-query

```
select title
from MOVIE
where Year >= 1980
```

- Each absence preference corresponding to a path with at least one join that is to-many in the direction of the selection is inserted into a separate sub-query. The following preference

```
Not(MOVIE.mid=CAST.mid and CAST.aid=ACTOR.aid and ACTOR.name='W. Allen')
```

is integrated into the following sub-query

```
select title
from MOVIE
where title not in
(select title from MOVIE M, CAST C, ACTOR AC
where M.mid=C.mid and
C.aid=AC.aid and AC.name='W. Allen')
```

The sub-queries constructed as described above are combined into the final query. This is the personalized query that will be executed. Using the example sub-queries constructed above and assuming $L=2$, the personalized query that will be executed instead of the initial one in our example is the following

```
select title
from
(select title from MOVIE M, CAST C, ACTOR AC
where M.mid=C.mid and
C.aid=AC.aid and AC.name='W. Allen')
union all
(select title from MOVIE
where Year >=1980)
union all
(select title from MOVIE
where title not in
(select title from MOVIE M, GENRE G
where M.mid=G.mid and
G.genre='comedy' or G.genre='action'))
```

Ranking of results based on the preferences satisfied is performed by inserting an appropriate `order by` clause, provided that each tuple returned is accompanied by its respective degree of interest. If a sub-query includes a hard preference, then each tuple returned by this one will have the same doi. If a sub-query contains a soft preference, then the doi of each tuple will be provided by the corresponding function of the degree of interest.

Although this approach is simple, it has certain disadvantages: (i) it generates results that are not self-explanatory; (ii) it cannot rank results based on which preferences from the $K$ selected are satisfied and which are not; (iii) it is very inefficient when there are absence preferences, such as the one above referring to W. Allen; (iv) it does not allow for a progressive retrieval of tuples. Tuples are returned only after they have all been retrieved, merged, grouped and ordered. We will refer to this approach as SGPA (Simple Generation of Personalized Answers).

**Progressive Algorithm.** We now describe an algorithm for progressive generation of personalized answers, called PGPA (Progressive Generation of Personalized Answers). Its characteristics are: (i) it generates results ranked and self-explanatory; (ii) it permits ranking of the results based on which preferences from those selected are satisfied and which are not, as well as, based only on the former; (iii) it outputs results as soon as they are available, thus it demonstrates a better initial response time; (iv) it handles more efficiently absence preferences.

```
Progressive Algorithm- PGPA
Input: minimum number of preferences to satisfy $L$, set of simple queries $\{S_i | i=1...N_S\}$, set of exclusion queries $\{E_i | i=1...N_E\}$, set of parameterized queries $\{Q(t) | i=1...N_T-1\}$, set of parameterized queries $\{Q(t) | i=1...N_T-1\}$,
Output: Set of personalized tuples $R$
```

1. $\text{MEDI}=f(\{L_i | i=1...K\})$
2. While remaining sub-queries can satisfy $L$ preferences and $\exists S_i$
   2.1 Execute $S_i$
   2.2 For each $i$ returned by $S_i$ not contained in $R$
      Set $\text{prefsSatisfied}=\text{Set} \; \text{current}_L=\text{Set} \; \text{degreesSatisfied}
      \text{Execute} \; Q(t)$
      Update $\text{prefsSatisfied}$; Update $\text{current}_L$; Update $\text{degreesSatisfied}
      \text{Execute} \; Q(t)$
      Update $\text{prefsSatisfied}$; Update $\text{current}_L$; Update $\text{degreesSatisfied}$
      TuleDegree=\$\text{degreesSatisfied}$
      If $\text{current}_L \geq L$, then
      $R \leftarrow (t, \text{prefsSatisfied}, \text{TupleDegree})$ End if
      While $\exists t \in R$ not yet output, with $\text{TupleDegree} \geq \text{MEDI}$
      Output $t$
      End While
      End For
      Update MEDI
      End While
3. While remaining sub-queries can satisfy $L$ preferences and $\exists E_i$
   3.1 Execute $E_i$
   3.2 For each $i$ returned by $E_i$ not contained in $R$
      Set $\text{prefsSatisfied}=\text{Set} \; \text{current}_L=\text{Set} \; \text{degreesSatisfied}
      \text{Execute} \; Q(t)$
      Update $\text{prefsSatisfied}$; Update $\text{current}_L$; Update $\text{degreesSatisfied}$
      TupleDegree=\$\text{degreesSatisfied}$
      If $\text{current}_L \geq L$, then
      $R \leftarrow (t, \text{prefsSatisfied}, \text{TupleDegree})$ End if
      While $\exists t \in R$ not yet output, with $\text{TupleDegree} \geq \text{MEDI}$
      Output $t$
      End While
      End For
5. $R=R \cup \{(\text{Allids}, \text{Nids})\}$

---

**Figure 6.** Progressive generation of personalized answers
The algorithm is described in Figure 6. Preferences are combined into sub-queries as described in the previous subsection. These are placed in order of increasing selectivity. We use simple histograms to obtain this information. Sub-queries that include a to-many absence preference are denoted $E$. Let $N_S$ be the number of them. The remaining sub-queries, which we call simple, are denoted $S$. Let $N_S$ be the number of them. Each of these sub-queries returns a tuple id, and the table attribute and the value involved in a preference. A sub-query $E$ is constructed in exactly the same way as if it were a presence preference. In that way, we avoid the overhead of the time-consuming absence sub-queries constructed in the SGP method. The difference between $S$’s and $E$’s is that a tuple returned by the former satisfies the associated preference, while a tuple returned by the latter does not satisfy the associated preference. Our algorithm relies on this difference in order to generate the personalized tuples.

For each sub-query $S$, we construct a parameterized query $Q^S_i(t)$ that is the union of sub-queries $S_j$, $j>i$. Parameter $t$ is a tuple id. Similarly, for each $E$, we construct a parameterized query $Q^E_i(t)$ that is the union of sub-queries $E_j$, $j>i$. A list $R$ of retrieved tuples is kept in order of decreasing degree of interest.

In each round, the algorithm executes the sub-query $S$ with the lower selectivity from those not yet executed. For each distinct tuple returned, we keep (i) the preferences satisfied based on $S$, $\text{prefsSatisfied}$, (ii) their corresponding degrees $\text{degreesSatisfied}$, and (iii) their number $\text{currentL}$. The algorithm executes the parameterized queries $Q^S_i(t)$ and $Q^E_i(t)$ and updates the aforementioned structures accordingly taking into account the difference between these queries mentioned previously. If this tuple satisfies at least $L$ preferences, then its overall degree of interest $\text{TupleDegree}$ is calculated using any of the formulas mentioned in the beginning of the paper, and the tuple is placed in the list $R$ along with the preferences it satisfies and its degree of interest.

The algorithm executes sub-queries $E$ following a similar procedure like the one described above for simple queries. In addition, it keeps a list $\text{Nids}$ of all tuple ids returned by these queries. All tuples whose ids are not included in this list are also returned by the algorithm.

In effect, the algorithm can start producing results 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 sub-queries not yet executed. Each time a new tuple is inserted into $R$, the algorithm checks if there are tuples there that can be output. A tuple is output if its doi is greater than or equal to MEDI. The algorithm stops execution of sub-queries if the remaining ones can satisfy less than $L$ preferences.

7. EXPERIMENTAL RESULTS

Our data primarily comes from the Internet Movies Database [13] and contains information about over 340000 films played in theatres or in DVDs. We conducted two series of experiments, one with synthetic profiles in order to evaluate the efficiency of our personalization algorithms and one with human subjects to evaluate empirically the effectiveness of our approach.

7.1 Efficiency

In this series of experiments, we used sets of random profiles and sets of random queries. The number of preferences selected from a profile is $K$ and the minimum number of those that should be met by results is $L$.

![Figure 7: Preference selection times](image)

Preference Selection. We compared the performance of the two algorithms for preference selection, SPS and FakeCrit on the execution time. We measured the preference selection time for each combination of query and profile for varying $K$. Figure 7 shows results for a set of 20 random queries run over a set of 50 random profiles containing 100 atomic selection preferences. FakeCrit displays improved performance. As $K$ approaches the maximum number of selections possibly extracted from a profile, the two lines converge. This is expected because at this point they both end to read the whole profile. The maximum number read is smaller than the actual number of preferences stored per profile. That is why the two lines converge for $K$ less than the size of the profile (approximately 70 preferences for this experiment).

![Figure 8: Execution times with K](image)

![Figure 9: Execution times with L](image)
Generation of personalized answers. We compared the performance of the two algorithms for generation of personalized answers, SGPA and PGPA. We conducted several experiments using a set of 50 random profiles and different sets of queries with varying result sizes ranging from 20 to 5500 rows. Figure 8 summarizes results for varying K and L=2 and for the case that selected preferences include no absence preferences. PGPA displays a very improved initial response time, while the overall execution time is a little worse than SGPA’s execution time. When L=1 or the query contains absence preferences, then the overall execution times of PGPA queries are better than SGPA’s. Due to space limitations, we present only Figure 9 which shows the comparison of execution times for different values of L and K=35.

7.2 Effectiveness

We conducted an empirical evaluation of our approach with 12 human subjects. 6 of them have a diploma in computer science (experts). The rest of them are simple users of computers. Each one of them manually defined their profiles, which were stored in our system. Two trials were conducted, each one using a web-based client developed for this purpose.

In the first trial, all subjects were given a set of 3 specific queries and in addition, they were asked to think of two additional ones they would like to ask the system. Each person submitted the set of 5 queries twice in arbitrary order. Each query was 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 the query they submitted. In any case, if a query returned no results, it was replaced by another one that was submitted instead. Each user was asked to electronically: (i) evaluate each tuple returned by a query by giving a tuple degree of interest in the range [-10, 10]. Value -10 indicated that they completely rejected the tuple and value 10 indicated that they strongly favored it, and (ii) evaluate the overall answer to a query by providing:

- an estimation of the difficulty to find something interesting (degree of difficulty), if they found anything.
- an estimation of how well the answer covered their request (coverage).
- 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 of each individual and L=2. Below we present only the results for the answer score, as the other two measures show similar behavior.

Average tuple degree of interest. We consider only queries personalized. We are interested in comparing the degrees of interest per tuple provided by the user to the degrees of interest estimated by our preference combination formulas. The degrees of interest estimated by our formulas are multiplied by 10.

Figure 10 shows how each tuple returned by a particular query was evaluated by some user and by our formulas. Est. Tuple Interest 1 is calculated by formula (4) which is based on the inflationary approach. Est. Tuple Interest 2 is calculated by formula (5) which is based on the reserved approach. We see that the user interest per tuple lies between the degrees calculated by the two formulas. Figure 11 provides a summary per user over all queries. Avg. Est. Tuple Interest 1 is calculated by formula (4) while Avg. Est. Tuple Interest 2 is calculated by formula (5). Interestingly, the average degree of interest per query estimated by formula (5) follows very closely the average user interest.

Answer score. For each query, we compare the average answer score reported by each individual when the query was executed unchanged to the average score reported when the query was personalized.

Figure 12 presents results for the experts group. We observed that personalized answers have higher scores. Figure 13 presents similar results for the users group.

Figure 14 presents the average answer score per group of people over queries unchanged and queries personalized. We observe that no-experts give lower scores to queries unchanged and higher scores to personalized ones than experts.
In the second trial, all human subjects were asked to think of a specific need, e.g., to find a theatre to go tonight or a DVD to rent. Queries submitted by half of them were left unchanged by the system, while queries of the rest were personalized.

Figure 15 shows the average degree of difficulty reported by each group. Figure 16 shows the average coverage reported by each group and Figure 17 shows average scores. We observed that personalized queries led into more effective searches.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we build on the personalization framework presented in earlier work. We provide a generalized model for representing and storing preferences in profiles. This model captures different types of preferences not considered earlier, such as soft preferences and dislikes or negative preferences. These new types impose new challenges for query personalization; thus we provide new algorithms for generation of personalized answers. We presented many results from experiments with both synthetic and actual human profiles for measuring both the efficiency of our algorithms as well as the overall effectiveness of our approach. We are interested in 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, and see what kind of statistics may be additionally needed and how they are obtained. We are also interested in investigating ways of optimizing personalized queries in terms of running time and result size constraints.

9. REFERENCES


