Personalizing Queries based on Networks of Composite Preferences

Georgia Koutrika
Stanford University, USA
and
Yannis Ioannidis
University of Athens, Greece

People's preferences are expressed at varying levels of granularity and detail as a result of partial or imperfect knowledge. One may have some preference for a general class of entities, e.g., liking comedies, and another one for a fine-grained, specific class, e.g., disliking recent thrillers with Al Pacino that are intended for families. In this paper, we are interested in capturing such complex, multi-granular preferences to personalize database queries and in studying their impact on query results. In particular, we organize the collection of one's preferences in a *preference network* (a directed acyclic graph), where each node refers to a subclass of the entities that its parent refers to, and whenever they both apply, more specific preferences override more generic ones. We study query personalization based on networks of preferences and provide efficient algorithms for identifying relevant preferences, modifying queries accordingly, and processing these queries to obtain personalized answers. Finally, we present results of both synthetic and real-user experiments, which (a) demonstrate the efficiency of our algorithms, (b) provide insight as to the appropriateness of the proposed preference model and (c) show the benefits of query personalization based on composite preferences compared to simpler preference representations.

Categories and Subject Descriptors: H.1 [**Models and principles**]: ; H.2.8 [**Database Applications**]:

General Terms: Theory, Algorithms, Experimentation

Additional Key Words and Phrases: preference modeling, preference hierarchies, personalization

1. INTRODUCTION

Personalization has come about as a result of a long evolutionary process accelerated by the rapid development of the World Wide Web and personal communication technologies. The Web has enabled people with varied goals and characteristics to access an ever-growing amount of information. The emergence of hand-held electronic devices, such as palmtops and cellular phones, has increased the possibilities for information access from anywhere, anytime. In this context, finding the "right information" remains a difficult problem accentuated by the growing rate at which new information becomes available and the heterogeneity of people searching for information. To help users cope with information overload, personalization methods, such as recommendations (e.g., [Balabanovic and Shoham 1997; Das et al. 2007; Linden et al. 2003]) and query personalization (e.g., [Koutrika and Ioannidis 2004; Liu et al. 2004; Pitkow et al. 2002]), have been proposed both by industry and academia.

Query personalization is based on the observation that "different users may find different answers relevant when searching" because of different preferences and goals [Pitkow et al. 2002]. It generates personalized results by dynamically enhancing a query with related preferences stored in a user profile and changing the order and possibly the size of results. Which preferences are related to a request and how these affect the final answer is dynamically determined based on the query, the profile and the personalization logic applied. For instance, preferences for dramas are not related to a search for comedies and preferences for actors may not be related to searches regarding movie directors. A personalized answer is always ranked according to the user's preferences; depending on the query personalization logic, it may contain all results that match the query or only a subset that satisfies some of the related preferences.

In this paper, we are interested in modeling and using user preferences for query personalization in the context of databases. Our interest in databases stems from the fact that databases comprise an important part of the (deep) Web, as witnessed by the proliferation of database-driven web sites (e.g., e-commerce and social bookmarking sites), where different notions of user information are very important.

Modeling preferences and ranking query results based on preferences hide many challenges. Existing works have studied various aspects of these problems, such as qualitative (e.g., [Chomicki 2003; Kiessling 2002]) vs. quantitative preference formulations (e.g., [Agrawal and Wimmers 2000; Koutrika and Ioannidis 2004]), different types of preferences (e.g., likes and dislikes [Kiessling 2002; Koutrika and Ioannidis 2005b]), context and preference combinations for result ranking (e.g., [Holland and Kiessling 2004; Stefanidis et al. 2007]), and so forth. In this paper, we study preference modeling at various levels of granularity and result ranking that is aware of and respects the preference importance as well as the natural relationships between multi-granular preferences.

Multi-granular, composite user preferences and ranking. In an ideal world, preferences could exist *for every object* in a domain of interest, e.g., for every movie, for every book, etc. Such elaborate preferences would yield a perfectly fine-grained ranking of alternatives and highly personalized content. In another ideal world, one could express preferences *for general but well-defined, disjoint sets* partitioning the objects of discourse. Unfortunately, neither of these two extremes is found in practice often. User preferences are typically incomplete and, furthermore, our knowledge of them is imperfect and partial. Different factors may contribute to this phenomenon, as we explain below.

People tend to have a mix of general and specific preferences. This mix may indicate lack of knowledge, lack of elaborate taste, or inability to identify those properties of objects that determine their preferences. For instance, one may have a general liking for adventures but a finer-grained taste only for a subset of them, such as those directed by S. Spielberg, which are characterized as favorites. On the other hand, one may have liked some particular books but may be unable to identify a common characteristic that made them attractive. Implicit collection of user preferences, for instance by observing user behavior in the system, suffers from similar problems. For instance, consider a system that builds user profiles from user ratings for movies to provide personalized content. Suppose that a user has given high ratings to all comedies she has seen so far. This knowledge allows the system to conclude that the user likes comedies but does not suffice to differentiate among comedies. Suppose that the user continues to interact with the system and provides some low scores for comedies starring Jack Nicholson. Then, the system may enrich its knowledge of this user's preferences by discriminating against comedies starring Jack Nicholson, which will be ranked lower than other comedies. On the other hand, what the system knows about the user may not suffice for making generalizations. For instance, assume that another user may have liked all comedies with Adam Sandler so far; this fact does not necessarily imply, however, that the user should like all comedies or all movies with Adam Sandler independently.

Paper focus and contribution. Given that general and specific preferences can peacefully coexist, in this paper, we study *query personalization based on multi-granular preferences*. We represent preferences as degrees of interest in query conditions following a similar philosophy to previous works [Koutrika and Ioannidis 2004; Stefanidis and Pitoura 2008]. The key difference from earlier work, however, is that we allow a profile to contain any mix of general and specific preferences, where the latter are explicitly stated rather than being implicitly calculated from the former.

For personalizing the results of a query, we adopt the query personalization framework presented in earlier work [Koutrika and Ioannidis 2005b], which is based on the following principles:

- A preference is related to a query if the former can be integrated into the latter based on the syntactic characteristics of both.
- A personalized answer should satisfy at least l of the top k user preferences related to a query, in order of decreasing degree of interest.

Parameters l and k provide a quantitative way to describe the desired answer. Parameter k determines which of the related user preferences should be considered for application to a particular query. It also provides a way to control the cost of query personalization, as the fewer the preferences integrated into a query, the more efficient that query typically is. Parameter l captures the minimum number of user criteria (i.e., preferences) that an answer should meet, thereby offering a degree of flexibility to personalizing a query answer. We discuss possible ways to specify parameters l and k in Section 4.1.

At query time, a mix of generic and specific preferences may be related to a query but these may not be freely combinable to determine the ranking of the query results. For example, a preference for comedies can be combined with a preference for actor Adam Sandler but they are both overridden by a preference for 'comedies with Adam Sandler'. We formally define preference overriding relationships based on containment mappings. Given a set of preferences related to a query, we automatically identify such relationships and we organize them in a *preference network* (a directed acyclic graph). Each node in the network refers to a subclass of the entities that its parent refers to. Whenever they both apply, the more specific preference (e.g., a preference for comedies with Adam Sandler) overrides the more generic one (e.g., a preference for comedies), whether the former represents a stronger or a weaker preference than the latter. To the best of our knowledge, this is the first systematic study of multi-granular preferences and preference relationships in a quantitative way.

Handling multi-granular preferences and preference relationships can lead to increased complexity and, thus, higher execution times. One approach would be to build an appropriate set of queries that expand the initial query with combinations of preferences that are allowable based on the preference relationships and return ranked results. For example, we could execute a query that returns comedies without Adam Sandler and another query that returns comedies with Adam Sandler. However, such queries may be quite complex and time-consuming even for a few preferences with no relationships among them [Koutrika and Ioannidis 2005a]. Organizing the preferences in a network helps preference processing in the following ways. It helps keep track of the preference relationships and the preferences to use for ranking each tuple. In addition, our personalized query answering algorithms can work directly on the network. They exploit the containment mappings implied by the network to decide how to traverse the network, i.e., which preferences to process and in what order resulting in reduced preference processing. Exploiting containment mappings for optimizing performance and in general exploiting the characteristics of preferences (not only the characteristics of the queries) to optimize queries with preferences has not been considered in previous works so far.

We evaluate our framework and algorithms both for effectiveness and efficiency. We study the effect that the increased expressive power and freedom offered by our framework has to users through a user evaluation. Naturally, we raise the question of whether increased expressiveness is achieved at the expense of performance and we answer it in the negative through an appropriate experimental performance evaluation.

In summary, our contributions are the following:

- We introduce a framework for multi-granular preference modeling that is based on the concept of networks of preferences, which allow the representation of both generic and specific preferences and their relationships in a concise and flexible way (Section 3).
- We present a system architecture for query personalization based on preference networks (Section 4).
- We describe efficient algorithms that, given a query and a user profile, derive the top k preferences that are related to the query, identify the relationships among them and organize them in a network (Section 5).
- We use the preference network to support algorithms for computing the personalized results of a query. Our algorithms process preferences in an efficient way by exploiting the containment mappings captured in the network (Section 6).
- We study the overall impact of the increased expressiveness allowed by our framework through both synthetic and real-user experiments that evaluate the preference model and the algorithms. We show that more accurate, finer-grained result rankings can be achieved without losing in performance (Section 7).

2. RELATED WORK

Preference is a fundamental notion in applied mathematics [Fishburn 1999], philosophy [Hansson 2001] and AI [Wellman and Doyle 1991]. In Databases, preferences have been used for cooperative query answering, i.e., for providing answers with extra or alternative information that may be meaningful to the user [Cuppens and Demolombe 1989; Gaasterland et al. 1992]. Recently, preferences have attracted renewed interest in the database community. This has been triggered by the observation that the strict Boolean information access model assumed in databases is some times restrictive. Query criteria are considered as *hard* by default, and a non-empty answer is returned only if it satisfies all criteria. On the contrary, in Information Retrieval, query criteria (terms) are considered *soft* and an answer contains results ranked according to how well they match the query. Recent years have witnessed the emergence of approaches aiming at the introduction of soft criteria or preferences in database queries, resulting to *preference queries*, with some early efforts going back to eighties [Lacroix and Lavency 1987]. These approaches are divided into the following categories:

Qualitative approaches aim at a relative formulation of preferences, such as a user prefers comedies over westerns. Such a formulation is natural for a human and results in partial orders of results. Absolute specification of the significance of a preference is not possible. Preferences between tuples in the answer to a query are specified directly, using binary preference relations [Borzsonyi et al. 2001; Chomicki 2003; 2004; Kiessling 2002; Kiessling and Kostler 2002]. Two frameworks have been independently proposed in which preference relations are defined using logical formulas [Chomicki 2003] or special preference constructors [Kiessling 2002]. Preference relations are embedded into relational query languages through a relational operator that selects from its input the set of the most preferred tuples (winnow [Chomicki 2003], BMO [Kiessling 2002]).

Quantitative approaches aim at an absolute formulation of preferences, such as a user likes comedies very much and westerns to a lesser degree. Such a formulation allows for total ordering of results and the straightforward selection of those that match user preferences. Preferences in queries are specified indirectly using scoring functions that associate a numeric score with every tuple of the query answer [Agrawal and Wimmers 2000; Hristidis et al. 2001].

There is extensive work on executing and optimizing preference queries, in particular:

- *skyline queries* [Borzsonyi et al. 2001; Chomicki 2004; Kossmann et al. 2002; Papadias et al. 2003; Pei et al. 2005], which are a special case of qualitative preference queries. Semantic optimization techniques have been proposed [Chomicki 2004]. Skyline queries have been also studied over uncertain data (e.g., [Pei et al. 2007]), streams (e.g., [Sarkas et al. 2008]) and in P2P environments (e.g., [Vlachou et al. 2007]).
- *top-n queries* [Bruno et al. 2002; Chang and Hwang 2002; Tao et al. 2007], i.e., queries that retrieve the best n objects that minimize a specific function. Core rank-aware query operators [Ilyas et al. 2004] and special structures, such as the P-Cube [Xin and Han 2008], have been proposed. Special cases of top-n queries have been studied, such as spatial top-n queries [Yiu et al. 2007], which rank objects based on the qualities of features in their spatial neighborhood.

Preference Applications. Research on preference queries was mainly inspired by the need to adopt an IR-like answer model, where an answer to a query contains results ranked according to how well they match the query, as opposed to the strict Boolean model traditionally assumed in databases. Independent of the answer model assumed, the answer of a query may be the same for all users. Recently, the need to personalize answers to fit different user preferences has been acknowledged [Pitkow et al. 2002]. User preferences may comprise an ephemeral or long-term profile and are used to tailor system answers. The trend for personalized answers emerged in the IR community. Personalized IR systems use different representations of preferences, such as bags of words [Joachims et al. 1997], vectors of terms [Balabanovic and Shoham 1997] and concept hierarchies [Liu et al. 2004], to provide information filtering, recommendations or personalized results.

The idea of using preferences to customize system answers has been transferred to databases and applications of preferences include recommendations [Satzger et al. 2006], query personalization [Koutrika and Ioannidis 2004; Koutrika and Ioannidis 2005a; Koutrika and Ioannidis 2005b; Stefanidis et al. 2007] and collaborative query results [Koutrika 2006]. In addition to finding interesting results for a query, a related problem is finding interesting attributes that characterize a set of results [Miah et al. 2008; Wong et al. 2007]. Preference modeling is central to all these approaches. For instance, Satzger et al. [2006]

adopt the preference model presented in [Kiessling 2002]. Koutrika [2006; 2005a; 2005b; 2004] and Stefanidis et al. [2007] represent preferences as query conditions associated with a degree of interest for personalizing queries. Personalizing queries based on preferences that hold only in specific contexts has been also studied [Agrawal et al. 2006; Holland and Kiessling 2004; Stefanidis and Pitoura 2008; Stefanidis et al. 2007; van Bunningen et al. 2006].

Preference modeling is only one of the challenges in preference applications. Other issues include dynamically identifying the preferences that are related to a request, selecting the most appropriate ones for the specific query and context and determining their effect on the final answer. The personalized answer may contain all results of the initial query ranked based on all the preferences that are appropriate under a specific context [Stefanidis et al. 2007]. Alternatively, a personalized answer may be shaped following an l out of k personalization logic: a personalized answer should satisfy at least l out of k preferences from the user profile that are related to the user query [Koutrika and Ioannidis 2005b; Koutrika and Ioannidis 2004]. A different approach is to view query personalization as an optimization problem using as parameters the query cost, the answer size and the interest in the answer and dynamically decide the appropriate personalized answer optimizing w.r.t. one parameter given constraints on the others [Koutrika and Ioannidis 2005a]. Acquiring user preferences is another great challenge. In the context of IR and Databases, preferences are either entered explicitly through a query interface or acquired at query time using a preference elicitation mechanism [Balke et al. 2007; Lee et al. 2008]. Long-term preferences can be learnt based on user feedback as well. There are studies for learning the order of objects that reflects user preferences [Cohen et al. 1998; Joachims 2002] or for learning preferences in object attributes (such as preferences formulated as described in [Kiessling 2002]) [Holland et al. 2003; Jiang et al. 2008].

Comparison to related work. Our work is closer to [Koutrika and Ioannidis 2004; Koutrika and Ioannidis 2005b; Stefanidis et al. 2007] in that we share the idea of representing preferences as query conditions associated with a degree of interest and that we adopt the "l out of the top k preferences" query personalization logic. These efforts focus on problems of preference representation, such as differentiating preferences based on their intensity [Koutrika and Ioannidis 2005b] and studying context-dependencies [Stefanidis et al. 2007]. However, they formulate structurally (and semantically) atomic preferences, i.e., over a single attribute. For example, they allow to capture a preference for Woody Allen and a preference for comedies but would not allow an explicit preference statement for comedies with Woody Allen. For such complex preferences, they provide mechanisms to derive them based on their simpler, constituent ones. However, a preference for comedies with Woody Allen may not be derivable in any natural way by a preference for comedies and a preference for Woody Allen. One may like both comedies and Woody Allen but one may like or dislike comedies with Woody Allen. As a result, the personalized results generated may not always accurately reflect the user preferences.

In this work, we are studying *complex* preferences and their implications. We allow complex preferences to be explicitly stated (not be solely derivatives of other preferences) and hence we can generate more accurate results. Furthermore, the central assumption in existing approaches is that preferences hold independently of each other and hence they can be freely combined for result ranking. Allowing preferences of different granularity makes this somewhat unrealistic assumption no longer necessary. Having removed this independence assumption, our framework allows the formulation of multi-granular preferences and captures preference relationships using preference networks, so that interrelated preferences (e.g., one for comedies and one for comedies with Woody Allen) can be treated appropriately (Section 3).

Our work is also different from existing approaches on executing and optimizing queries with preferences, such as skyline and top-n preference queries, in several aspects. First, we deal with different types of preferences than skylines. Skylines are formulated in the context of multidimensional Euclidean spaces [Borzsonyi et al. 2001]. The attributes of a relation are partitioned into DIFF, MAX and MIN attributes. Only tuples with identical values of all DIFF attributes are comparable; among those, MAX attribute values are maximized and MIN values are minimized. Our approach is also different from top-n queries (e.g., [Bruno et al. 2002; Tao et al. 2007]), where a single scoring function maps tuples to scores. We describe preferences irrespective of the attribute domain by specifying conditions that tuples must satisfy and assigning a degree of interest in them. Then, tuples are ranked depending on which of the conditions they satisfy.

Given the above differences in problem formulation, our work addresses several new challenges related to preference integration with queries: identifying preference relationships, constructing a network of related preferences for a query, and using this network to generate personalized answers that respect the semantics of query personalization and the relationships between the preferences of the network. Given any two preferences, we are able to determine their relationship based on conjunctive query containment mappings [A. Aho and Ullman 1979; Chekuri and Rajaraman 1997]. Preference mappings are simpler because they are defined on the basis of mappings of the atomic conditions in the preferences considered. When generating personalized results, our algorithms take into account the way preferences are related in the preference network to rank the output tuples.

By enforcing preference independence, previous works may produce less accurate results, which may be still acceptable given that query personalization using independent preferences is more straightforward and efficient. Clearly, there is a tradeoff between efficiency and expressivity in allowing multi-granular preferences. We experimentally compare our algorithms with their simpler counterparts, which assume preference independence, both for efficiency and answer accuracy, and we show that personalized answers that capture user preferences more accurately are feasible without losing in efficiency.

3. FRAMEWORK

We consider that for every user there is a profile storing user preferences for personalizing queries. We consider SPJ queries over relational databases. In this section, we present our multi-granular preference model (Section 3.1). Then, we define preference relationships and we introduce preference networks that capture a set of preferences and their relationships (Section 3.2). In order to rank the results of a query, we need to "estimate" the user's preference in each tuple in the results taking into account the way preferences are interrelated (Section 3.3).

3.1 Preference Formulation

A database comprises a set of relations. A relation R (denoted R_i when more than one relation is implied) has a set $\mathbf{A}$ of attributes. We will use $R.A_j$ to refer to an attribute in $\mathbf{A}$ or simply A_j when R is understood.

MOVIE($\underline{\text{mid}}$, title, year, duration)	GENRE($\underline{\text{mid}}$,$\underline{\text{genre}}$)
CAST($\underline{\text{mid}}$,$\underline{\text{aid}}$)	ACTOR($\underline{\text{aid}}$, name)
DIRECTED($\underline{\text{mid}}$,$\underline{\text{did}}$)	DIRECTOR($\underline{\text{did}}$, name, nationality)

Fig. 1. A database

A *preference* for a set of tuples of a relation R in the database is expressed as a degree of interest $d \in [0,1]$ in a condition q that describes this set of tuples, and is denoted:

$$R \ : \ (q,d)$$

Depending on the value of d, we capture weaker or stronger preferences, with $d = 0$ indicating no interest in the qualifying tuples and $d = 1$ indicating extreme interest. Depending on the form of condition q, we distinguish the following types of preferences:

—*Atomic preference*. If q is a single, atomic, selection or join condition, then a preference for q is called atomic.

—*Composite preference*. If q is a conjunction of multiple atomic conditions, then a preference for q is called composite.

—*Selection preference*. If q is a conjunction of atomic selections involving a set $\mathbf{A}$ of attributes and atomic joins transitively connecting these attributes to R on the database graph, then a preference for q is called a selection preference.

—*Join preference*. If q is a conjunction of atomic join conditions representing the transitive join of relations R and R_j on the database graph, then a preference for q is called a join preference.

A selection preference indicates a user's preference for tuples from R that satisfy the transitive selections in q. A join preference indicates the degree in which preferences for tuples in R are influenced by preferences on related, i.e., joining, tuples. Since a preference for a set of tuples is essentially captured as a preference for a condition that describes this set, in what follows, we consider the terms "preference for tuples" and "preference for a condition" equivalent and we use them interchangeably. We will use the symbol p (or p_i, if a set of preferences is discussed) to refer to a preference. We will use $doi(p)$ to refer to the degree of interest for a preference p or simply d if p is understood.

Example. To illustrate preferences, we consider the small database shown in Figure 1. A user, named Julie, has preferences for movies captured in her profile, part of which is depicted in Figure 2. For instance, she prefers movies released after 1990 (p_1) and she likes comedies (p_2) followed by adventures (p_3). She particularly likes family adventures (p_4). Preferences p_1, p_2 and p_3 are atomic selection preferences expressed on different attribute values. We also observe that preference p_1 is stronger than p_2 and p_3. Preference p_4 is a composite selection preference. Furthermore, Julie considers the director of a movie more important than the genre (p_5 and p_6). p_5 is a composite join preference and p_6 is an atomic join preference. Her favorite director is Alfred Hitchcock (p_7). She also likes sci-fi movies by Steven Spielberg captured by the composite selection preference p_8. $\square$

Preference Inference. From the preferences stored in a user profile, one can compose implicit preferences, i.e., preferences that are not explicitly stored in the profile but can be implied by those actually stored. Implicit preferences can be also derived by combining

(p_1)	MOVIE :	MOVIE.year > 1990,	0.9
(p_2)	GENRE :	GENRE.genre = “comedy”,	0.85
(p_3)	GENRE :	GENRE.genre = “adventure”,	0.7
(p_4)	MOVIE :	MOVIE.mid = GENRE1.mid and GENRE1.genre = “family” and MOVIE.mid = GENRE2.mid and GENRE2.genre = “‘adventure”	0.8
(p_5)	MOVIE :	MOVIE.mid = DIRECTED.mid and DIRECTED.did = DIRECTOR.did,	1.0
(p_6)	MOVIE :	MOVIE.mid = GENRE.mid,	0.7
(p_7)	DIRECTOR :	DIRECTOR.name = “A.Hitchcock”,	0.7
(p_8)	MOVIE :	MOVIE.mid = GENRE.mid and GENRE.genre = “scifi” and MOVIE.mid = DIRECTED.mid and DIRECTED.did = DIRECTOR.did and DIRECTOR.name = “S.Spielberg”	0.65

Fig. 2. Example preferences.

base and/or other implicit preferences. In the general case, implicit preferences can be derived from composeable preferences.

Two preferences, p_i and p_j, are *composeable*, iff

- p_i is a join preference, $R_i\ :\ (q_i, d_i)$, connecting R_i to a relation R_j, and
- p_j is a join or selection preference on R_j, i.e., of the form $R_j\ :\ (q_j, d_j)$,

We now consider a set of preferences: $R_1\ :\ (q_1, d_1)$, $R_2\ :\ (q_2, d_2)$, ..., $R_m\ :\ (q_m, d_m)$ such that, $\forall 1 \leq i < m$, preferences $R_i\ :\ (q_i, d_i)$, $R_{i+1}\ :\ (q_{i+1}, d_{i+1})$ are composeable. Then, we can compose the *implicit preference* $R\ :\ (q, d)$, as follows:

- $R \equiv R_1$,
- q is the conjunction of the conditions $q_1, q_2, ..., q_m$ and
- d is some function of the degrees of interest of the base preferences, i.e., $d = f(d_1, d_2, ..., d_m)$.

Inference Assumption: The degree of interest in an implicit preference is a non-increasing function f of the degrees of interest, $d_1, d_2, ..., d_m$, of its constituent preferences, i.e.,:

$$f(d_1, d_2, ..., d_m) \leq \min(\{d_1, d_2, ..., d_m\}) \tag{1}$$

Example functions for computing the degree of interest in implicit preferences include product and minimum. One advantage of product is that it is intuitive: the more the preferences required to derive an implicit preference, the weaker this preference is.

Example. From the preferences stored in Julie's profile (Figure 2), we can derive implicit preferences. For instance, since Julie likes director Alfred Hitchcock (p_7) and her movie preferences are affected by who is the director of a movie (p_5), we can assume that (in lack of any other evidence) Julie would like to some extent movies directed by Alfred Hitchcock by composing p_5 and p_7 into the following preference:

$(p_{5,7})$	MOVIE :	MOVIE.mid = DIRECTED.mid and DIRECTED.did = DIRECTOR.did and DIRECTOR.name = “A.Hitchcock”,	0.7

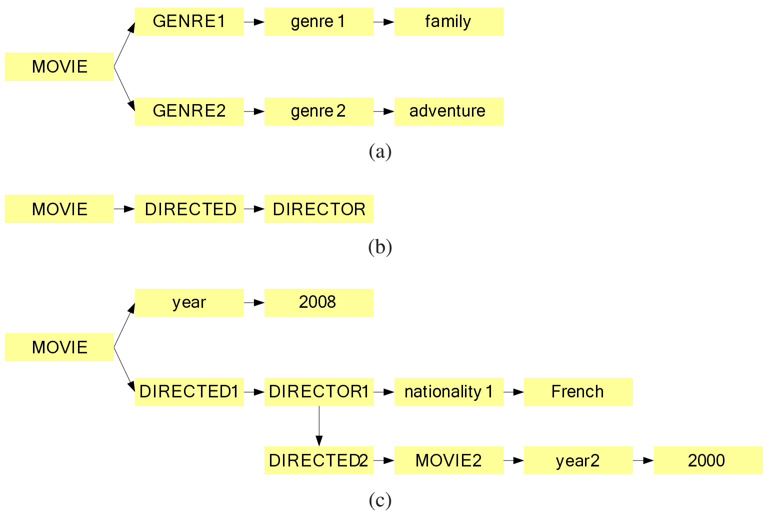


Fig. 3. Example preference diagrams.

Similarly, we can derive an implicit preference for movies that are adventures by combining preferences p_3 and p_6 into this preference:

$(p_{6,3})$	MOVIE :	MOVIE.mid = GENRE.mid and GENRE.genre = “adventure”,	0.49

For the sake of illustration, in both preferences, we have taken the product of the partial degrees of interest as the degree of interest in the derived preference. Of course, we could have used any other non-increasing function, such as minimum.

We observe that both base selection preferences p_3 and p_7 have the same degree of interest, i.e., 0.7. However, when combined with their composeable join preferences, the degrees of interest in the resulting implicit preferences, $p_{6,3}$ and $p_{5,7}$, are different. It is join preferences that determine to what extent preferences for tuples in one relation are influenced by preferences on related, i.e., joining, tuples. In this case, preference p_5 indicates that directors are more important for movies than genres (preference p_6). Consequently, preferences for directors will count more than preferences for genres when determining movie preferences. □

Preference diagram. A preference $R\,:\,(q,d)$ can be graphically represented as a rooted directed acyclic graph $g(V,E)$. The *preference diagram* $g(V,E)$ can be thought of as an extension of the traditional schema graph and has the following characteristics. Nodes in V map to relations, attributes, and values in q, and can be possibly replicated if the corresponding relation, attribute or value is used more than once in q. Edges in E connect (a) a relation node to another relation node, representing a join condition in q or (b) an attribute to its container relation or (c) an attribute to a value, representing a selection in q. Join edges are tagged with the corresponding joining attributes, and selection edges are tagged with the operator used in the selection condition. R maps to the graph's root. For selection preferences, the leaves are always values.

A preference diagram can take either of two forms. It can contain one single path mapping to a join or selection preference or it can contain a set of paths connecting its root to a set of values capturing a composite multi-value selection preference. As we will see in Section 5.2, we can determine if there is an overriding relationship between a pair of preferences by comparing their corresponding preference diagrams.

Example. Figure 3 illustrates some example preference diagrams. Figure 3(a) depicts the preference diagram for p_4. Figure 3(b) shows the preference diagram for p_5, which comprises a single path in contrast to the diagram for p_4, which maps to a set of paths from the root to the value nodes. For the sake of presentation, we do not show tags on the edges, since for our small database they are understood. Repeated relations and attributes are mapped to different nodes in the preference diagram. As a notational convention, different occurrences of a relation (mapping to different nodes in the diagram) are written as a concatenation of the relation name and a sequential number. To distinguish when an attribute is used with a different occurrence of the same relation, the attribute takes the relation's sequential number in their representation. □

Preference diagrams can "unfold" complex conditions, such as self-joins, and map them to distinctive paths on the diagram. To illustrate this, consider the following, rather non-typical, preference for movies released in 2008 and directed by French directors who have also directed movies released in 2000. The respective preference diagram is shown in Figure 3(c).

```
MOVIE :     MOVIE.year = 2008 and
            MOVIE.mid = DIRECTED1.mid and
            DIRECTED1.did = DIRECTOR1.did and
            DIRECTOR1.nationality = "French" and
            DIRECTOR1.did = DIRECTED2.did and
            DIRECTED2.mid = MOVIE2.mid and
            MOVIE2.year = 2000
```

3.2 Preference Networks

Depending on the form of condition q, selection preferences can be defined on a relation R at varying levels of granularity, ranging from quite generic (e.g., an atomic selection on R) to very specific that combine multiple (atomic or implicit) selections. Intuitively, one may view a selection preference as defining a *class* of entities with some particular features. Given preferences $R : (q_i, d_i)$ and $R : (q_j, d_j)$, q_i and q_j may define two different classes of entities from R. For example, preferences p_1 and $p_{5,7}$ describe two independent classes of movies: the class of movies released after 1990 and the class of movies directed by Alfred Hitchcock. A movie can belong to both classes, hence, these preferences can *independently hold* for the same movie. However, preference p_4 explicitly refers to a *subclass* of the entities that $p_{6,3}$ refers to, i.e., adventures that are also family movies. Consequently, whenever they both apply, the more specific one, i.e., the one defining the subclass, *overrides* the more generic one. In what follows, we formally define preference relationships: preference overriding and independence.

We first start with some observations. We can view a preference $R_i : (q_i, d_i)$ as a possible conjunctive query, which selects tuples from R_i that satisfy q_i. On the basis of this correspondence, we can define preference overriding through conjunctive query containment and containment mappings: Given two preferences, $R_i : (q_i, d_i)$ and $R_j : (q_j, d_j)$,

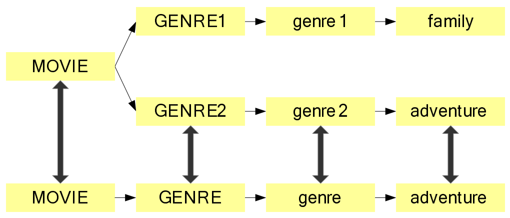


Fig. 4. Preference overriding.

we consider the corresponding conjunctive queries Q_i and Q_j. We say that p_i *is overridden by* p_j if Q_j is contained in Q_i ($Q_j \subseteq Q_i$). Q_j is contained in Q_i iff there is a containment mapping from Q_i to Q_j [A. Aho and Ullman 1979; Chekuri and Rajaraman 1997]. A containment mapping is defined on the basis of mapping the predicates involved in the two queries. In what follows, we formally define preference overriding on the basis of how one preference can be mapped to another:

Preference Relationships. Given a set **P** of selection preferences and two preferences $R_i : (q_i, d_i)$ and $R_j : (q_j, d_j)$ (denoted p_i and p_j, respectively) from **P**, we distinguish the following possible relationships between them:

—*Preference Overriding*. If p_i can be mapped to p_j, such that (a) $R_i \equiv R_j$ and (b) each atomic condition in q_i is mapped to an atomic condition in q_j with the same relations and attributes, then p_i *is overridden by* p_j (p_j *overrides* p_i), denoted $p_i \sqsubset p_j$. In this relationship, p_j is *specific*, p_i is *generic*, and the following degree of interest correspondences hold (denoted by $\leftharpoondown$):

$$doi(p_i \wedge p_j) \leftharpoondown doi(p_j) \tag{2}$$

$$doi(p_i \wedge \neg p_j) \leftharpoondown doi(p_i) \tag{3}$$

i.e., the generic preference p_i holds for a tuple only if the specific preference p_j does not hold for the same tuple.
Furthermore, if $\nexists p \in \mathbf{P}$ such that $p_i \sqsubset p \sqsubset p_j$, i.e., p_j is the most generic specialization of p_i in **P**, then p_i is *minimally overridden* by p_j, denoted $p_i \sqsubseteq p_j$.

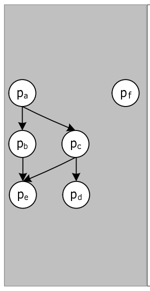
—*Preference Independence*. If $R_i \equiv R_j$ but $p_i \not\sqsubset p_j$ and $p_j \not\sqsubset p_i$, then p_i and p_j are *independent* and the following holds:

$$doi(p_i \wedge p_j) \leftharpoondown h(doi(p_i), doi(p_j)) \tag{4}$$

i.e., both preferences can hold for the same tuple and the overall degree of interest is the combination of their degrees of interest.

In other words, a preference p_i is overridden by a preference p_j if both of them are expressed over the same relation and there is a "to-1" mapping of the atomic conditions from one preference to the other. Figure 4 shows an example.

Preference Network. Given a set **P** of selection preferences over the same relation R, we can map **P** to a network, where nodes map to preferences and edges represent preference relationships. Each node in the network refers to a subclass of the entities that its parent refers to, and whenever they both apply, the more specific preference overrides the more generic one. More formally:



(p_a)	MOVIE.year = 2000,	0.75
(p_b)	MOVIE.year = 2000 and MOVIE.mid = GENRE.mid and GENRE .genre = “comedy”,	0.8
(p_c)	MOVIE.year = 2000 and MOVIE.mid = CAST.mid and CAST .aid = ACTOR.aid and ACTOR.name = “Jack Nicholson”,	0.85
(p_d)	MOVIE.year = 2000 and MOVIE.mid = CAST 1.mid and CAST 1.aid = ACTOR 1.aid and ACTOR 1.name =”Jack Nicholson” and MOVIE.mid = CAST2.mid and CAST2.aid = ACTOR2.aid and ACTOR 2.name = “Adam Sandler”,	0.7
(p_e)	MOVIE.year = 2000 and MOVIE.mid=GENRE.mid and GENRE .genre = “comedy” and MOVIE.mid = CAST.mid and CAST .aid = ACTOR.aid and ACTOR .name = “Jack Nicholson”,	0.4
(p_f)	MOVIE.mid = GENRE.mid and GENRE .genre = “adventure”,	0.95

Fig. 5. A set of preferences as a network.

A *preference network* $G_H(V_H, E_H)$ corresponding to a set of preferences **P** is a directed acyclic graph. Nodes in V_H map to preferences in **P**. Given two nodes v_i and v_j in V_H, mapping to preferences p_i, $p_j \in \mathbf{P}$, respectively, an edge in E_H from v_i to v_j, $e(v_i, v_j)$, denotes that p_i is minimally overridden by p_j. When a preference is not overridden by any other preference, it is called a *leaf.* When it does not override any preferences, it is called a *root*.

Example. Figure 5 illustrates a preference network (on the left side) corresponding to a set of preferences (on the right side). p_a is a root, p_d and p_e are leafs, and p_f is both a root and a leaf. We observe that the degrees of interest at one level of the network are not derived from the degrees of interest at the higher or lower levels of the network. For instance, p_b is a strong preference, whereas the more specific p_e, which differentiates a subset of tuples satisfying p_b, is a weak preference.

If there is no directed path in the network that connects two preferences, then these preferences are independent. For instance, p_b, p_d and p_f are independent. □

3.3 Combining Preferences

To compute the degree of interest in a particular tuple, we combine the degrees of interest of the preferences that are known for this tuple. For instance, given that Julie likes movies by S. Spielberg and action movies, we can compute a preference for the movie "Minority Report" that satisfies both preferences.

Consider a set **P** of selection preferences for the same relation R. The degree of interest in a tuple $t \in R$ satisfying the preferences in **P** is performed with the help of a ranking function h as follows:

$$doi(t) = doi(\wedge_{p_i \in \mathbf{P}'} p_i) = h(\{doi(p_i) | p_i \in \mathbf{P}'\}) \tag{5}$$

with $\mathbf{P}' \subseteq \mathbf{P}$ such that:

- $\forall p_i, p_j \in \mathbf{P}'$, p_i, p_j are independent and
- $\forall p_i \in \mathbf{P}'$, $\nexists\, p_j \in \mathbf{P}$ that is more specific from p_i.

In other words, given a set of preferences that are satisfied by a tuple t, only the most specific, independent preferences will determine the degree of interest in t.

There are many possible ranking functions that could be used, e.g., the maximum or the average, depending on the application, the objects of interest, and so forth. For instance, for choosing movies, ranking them based on the best preference they meet may suffice, whereas when buying a car, where many of the car features matter, a function such as the average of preferences may be more appropriate. A classification of ranking functions can

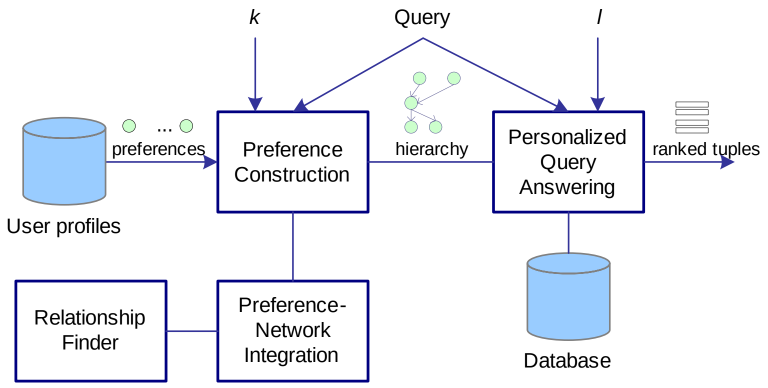


Fig. 6. System architecture.

be found in [Koutrika and Ioannidis 2005b].

Example. Consider the preferences shown in Figure 5 and a movie m that is a 2000 comedy with J. Nicholson. This movie, in principle, satisfies preferences p_a, p_b, p_c and p_e. Taking into account their relationships, not all preferences count and the degree of interest in this movie is computed as follows:

$$\begin{aligned} doi(m) = doi(p_a \wedge p_b \wedge p_c \wedge p_e) &\overset{p_a \sqsubset p_b}{=} doi((p_a \wedge p_b) \wedge p_c \wedge p_e) \overset{Formula\ (2)}{=} \\ doi(p_b \wedge p_c \wedge p_e) &\overset{p_b \sqsubset p_e}{=} doi((p_b \wedge p_e) \wedge p_c) \overset{Formula\ (2)}{=} \\ doi(p_e \wedge p_c) &\overset{p_c \sqsubset p_e}{=} doi(p_e) = 0.4 \end{aligned}$$

That is, since p_e overrides p_a, p_b and p_c, it alone determines the degree of interest in the movie. □

4. QUERY PERSONALIZATION

4.1 Query Personalization Logic

We consider that a set of preferences for a particular user are stored in the *user profile* and query personalization is performed using this profile. In order to personalize the results of a query, one has to identify the preferences from the user profile that are related to the request and determine which of them are the most appropriate ones for the specific query and how they will affect the final answer. Obviously, not all preferences are “valid” for all queries. For instance, preferences for dramas are not related to a search for comedies and preferences for actors may not be related to searches regarding movie directors. Moreover, user preferences may have different effects on the results of a query. For instance, the system may return all results that match a query ranked according to the user’s preferences or only a subset of them that exactly match some preferences.

We consider that only selection (explicit or implicit) preferences related to and not in conflict with the query are valid for modifying its results. We define preference relatedness and conflicts on the basis of the syntactic characteristics of the query and the preferences in the spirit of the work by Koutrika and Ioannidis [2004], extended to our multi-granular preference model.

Related Preference. Preference $R : (q,d)$ is related to query Q, if R is referred to in Q.

Conflicting Preference. Preference $R : (q,d)$ is conflicting with a query Q, if (a) it is related to the query; (b) when q is inserted into the original query qualification, replacing any part of the latter that coincides with it, i.e., joins or selections on a common attribute, the resulting query qualification and the original one contain at least one common implicit selection; and (c) this selection is specified over a single-value attribute of the query results.

Example. Consider the set of preferences shown in Figure 5, which are all related to movies, and a query about family movies released in 2008. Then, preferences p_a-p_e are clearly conflicting with the query since they are about movies released in 2000. Preference p_f is valid because a movie can have more than one genre. □

We adopt the query personalization framework introduced by Koutrika and Ioannidis [2005b] for determining the effect of query personalization. Given a preference $R : (q,d)$, d shows the user interest in considering q for personalizing the results of a query involving R. Hence, preferences that are related to and not conflicting with a query can be ordered based on their degree of interest. We can describe the desired personalized answer with the help of two parameters as follows:

A personalized answer for a query Q and a user profile U is the set of results of Q that satisfy at least l of the top k user preferences related to Q and are ranked according to their degree of interest.

Parameters k and l together offer a flexible mechanism to capture the desired level of personalization for the results. For example, some special cases are:

- $\mathsf{l} = 0$: the personalized answer contains all the results matching the query ranked based on the k preferences.
- $\mathsf{l} = \mathsf{k}$: the personalized answer contains only the query results that satisfy all k preferences.

Parameters l and k can be specified in various ways. A user can explicitly put constraints on the acceptable personalized answer by specifying desired values for l and k. The system can also automatically determine the appropriate values.

Parameter k controls the desired *extent* of personalization, i.e., how many of the top user preferences should participate in personalizing the query results. For instance, the user may be willing to put no bound to k and see results that satisfy any of her related preferences (i.e., $\mathsf{k}\equiv$'all'). On the other hand, if the results are too many or the user is very selective, returning results that satisfy a small number k of preferences may be more appropriate. For example, a user interested in dinner after work may want to see only restaurants that satisfy her top 3 preferences.

Parameter l controls the desired *forcefulness* of personalization, i.e., how many of the k live preferences should the query results be forced to simultaneously satisfy. For instance, when buying a car a user may definitely want offers that meet all her preferences but, when selecting a movie, she may be more flexible and may accept movie recommendations that meet just two of her preferences.

Parameters k and l also provide a way for the system to control the size of the personalized answer and the cost of query personalization and the system could automatically determine appropriate values for them. The system may restrict k since considering a smaller set of preferences leads to smaller answers and may be more efficient. For example, if the user is browsing results through a cell phone, then the system may pick few preferences from the user profile in order to provide results on the go. If the user uses a laptop and a high-bandwidth connection, then a more flexible k or no bound for k may be selected. For the case of k=l, determining automatically appropriate values of k has been studied in the past [Koutrika and Ioannidis 2005a]. In that work, query personalization is formulated as a constrained optimization problem, where constraints are expressed as an upper bound on execution time of the final query and/or a lower or upper bound on its result size.

In a similar vein, the system may automatically determine the right values for l. For instance, if the initial query is very general and returns many results, then a larger number of preferences (i.e., $l \rightarrow k$) may be applied on the query in order to produce an answer of manageable size. Likewise, if the initial query is overly specific and returns very few results, then a smaller number of simultaneous preferences (i.e., $l \rightarrow 0$) may be chosen to avoid an empty answer. The preference network may also serve as the basis for choosing appropriate values of l. For instance, if the top preferences are highly correlated, i.e., they form a chain, then the system should automatically determine that l can only be equal to 1 and return results that satisfy exactly one of the k preferences.

The system may also learn l and k for each user by mining user logs. These may contain both explicit choices of l and k by users as well as other user actions, i.e., paying attention to specific tuples, that offer implicit indications for the desirable values of these parameters. For example, if the user always examines results that satisfy up to the top 3 of her preferences and most of the times her final picks satisfy at least 2 of those preferences, then we can provide more targeted results by using k=3 and l=2 in in future interactions of the user with the system.

Finally, if personalization is an interactive process and the user can specify the values of k and l, then there are potentially several cases where freely-selected values for k and l may not make sense, such as: (a) the top k preferences are highly correlated, i.e., there are fewer than k independent preferences, (b) there are not as many as k preferences related to the current query or (c) there are no results that satisfy as many as l preferences. In these cases, the system should guide the user in selecting meaningful values for k and l.

4.2 Overall Approach

Given a query Q, a user profile U and values for k and l, query personalization proceeds as follows.

The top k preferences that are related to a query are extracted from the user profile. Given that these may comprise a mix of generic and more specific preferences, we automatically identify the relationships among them and organize them into a network that captures these relationships. Subsequently, we use the network to guide preference processing for the computation and ranking of the personalized results. We propose two algorithms for personalized query answering that apply different traversal strategies on the network, exploit the containment mappings captured in it to process the preferences, and rank the results accordingly.

The query personalization system we have implemented is shown in Figure 6. *Preference Construction* extracts the top k preferences from the user profile that are related to a query. *Preference-Network Integration* is responsible for organizing them in a network that captures the relationships among them (if any). This module places each preference found from the first module in the right "position" in the network. It collaborates with the *Relationship Finder* that determines the relationship between a pair of preferences. *Person-*

```
Algorithm PFC
Input:   query $Q$, a user profile $U$, k
Output:  a preference network $G_H(V_H, E_H)$
Begin
1.      $V_H := \varnothing$ , $E_H := \varnothing$
2.      $QP := \{(p_i, P_i) \mid p_i \in U$ related to $Q$ $\}$
3.      While $QP <> \varnothing$ and k$> 0$                          {
3.1.      remove head $(p, P)$ from $QP$
3.2.      If $p$ is a selection preference           {
3.2.1.      add $p$ to $V_H$; k $:=$ k$-1$
3.2.2.      $G_H :=$ PNET$(G_H, (p, P))$              }
3.3.      Else                                                   {
3.3.1.      $s :=$ the unique path in $P$
3.3.2.      Foreach $p_j \in U$ composeable with $p$                 {
              If $(p \wedge p_j)$ is not conflicting with $Q$          {
                $P' := \varnothing$
                Foreach path $s_i \in P_j$                       {
                  $s' := concatenate(s, s_i)$
                  add $s'$ in $P'$                               }
              add $(p \wedge p_j, P')$ to $QP$                         } } } }
4.      output $G_H(V_H, E_H)$
End
```

Fig. 7. Building a network of top k preferences related to a query.

alized Query Answering takes into account the network of related preferences and returns results that meet the initial query and at least l from the top k preferences.

Sections 5 and 6 describe in detail the above modules and the algorithms and techniques employed in them.

5. PREFERENCE CONSTRUCTION

Algorithm PFC (preference-construction) returns the top k preferences from the user profile U that are related to and not conflicting with a query Q. These preferences are organized in a network, which is the output of the algorithm.

Preferences that are related to a given query include explicit preferences stored in the user profile but also implicit ones that can be derived by composing stored preferences. In order to select the top k preferences, the algorithm starts from the preferences that are stored in the user profile and are related to the query and iteratively considers additional preferences that are composeable with those already known to derive implicit ones that are also related to the query. The set of preferences that are related to the query is kept ordered in decreasing degree of interest. When k selection preferences have been inserted into the network, the *Inference Assumption* guarantees that these are the top k preferences related to the query. Hence, an exhaustive enumeration of all related preferences is avoided.

The algorithm is presented in Figure 7. *QP* is the set of preferences related to the query. Initially, it contains all related preferences explicitly stored in the profile (ln:2). At each round, the most significant preference is processed based on its type.

- A selection preference is added to the preference network that comprises the final output of the algorithm (ln: 3.2). This process is performed by PNET, which will be described in Section 5.1.

- A join preference is composed with other preferences (ln: 3.3). The algorithm considers all stored preferences that are composeable with it to infer new preferences that are related to and not conflicting with the query. These are inserted into *QP*.

We will use the *Inference Assumption* to prove the correctness of the algorithm.

Theorem 1: The algorithm is correct, i.e., it finds the top k preferences related to a query.

Proof: It is sufficient to show that the algorithm produces preferences related to the query Q in decreasing order of their degree of interest. It keeps an ordered list *QP* of preferences. At each round, the head p of the queue (which has the highest degree of interest) is picked and processed. If p is a selection preference with degree of interest d, we will show that: (a) d is greater than (or equal to) the degree of interest of any other selection preference in *QP*, and (b) d is greater than (or equal to) the degree of interest of any other selection preference that has not been seen yet (i.e., not in *QP*).

Since *QP* is ordered, (a) is self-evident. For (b), we observe that any unseen selection preference can be only derived from the join preferences currently in *QP*. Based on the *Inference Assumption*, for each join preference p_j in *QP* with degree of interest d_j, any selection preference that contains p_j will have a degree of interest at most equal to d_j. Since it is $d \geq d_j$ (due to (a)), (b) also holds. Consequently, we have shown that the algorithm produces preferences related to the query Q in decreasing order of their degree of interest. ■

Example. Let us assume that we want to find the top 3 preferences related to a query about movies given the user profile of Figure 2. (How the preference network is progressively constructed is the focus of the next section.) Figure 8 shows the status of *QP* and the top preferences found at the end of each round of the algorithm. Newly inserted preferences in both lists are shown instantly in different color. When three selection preferences have been found, the algorithm stops. It is guaranteed that these are the top 3 preferences related to a movie query. □

In addition, for each preference p, algorithm PFC constructs its preference diagram. In particular, it constructs the set P of all the root-to-leaf paths on the preference diagram. Each path is encoded as a string for the purposes of finding relationships between pairs of preferences, as we will discuss in Section 5.2. We distinguish two cases:

—If a preference is stored in a profile, then we have also stored all the root-to-leaf paths on its preference diagram. This step is done off-line. The algorithm PATHS is sketched in Figure 5 and is described below.

—If it is an implicit preference composed of other preferences: Algorithm PFC builds a new preference by composing a join preference p with another preference p_j (ln:3.3.2). Then, the set P' of paths for the new preference is generated by concatenating each of the paths that map to p_j (the set of paths for p_j is P_j) with the paths in P that maps to p. Note that since p is a join preference, its set P of paths is singleton.

We now describe algorithm PATHS. The algorithm is used for off-line processing of the preferences in a user profile. For a given preference p, it builds its preference diagram and

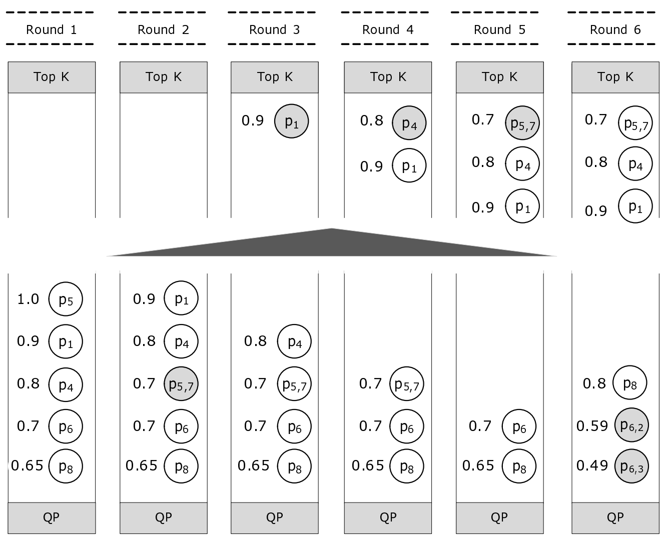


Fig. 8. Example of composing top k related preferences.

generates the set P of root-to-leaf paths on the diagram, which are then stored in the user profile. That pre-processing saves time for the on-line algorithm PFC. A path is represented as a string (for comparison purposes as we will see in Section 5.2.)

The algorithm first constructs the preference diagram $g(V,E)$ that corresponds to the preference p (ln:2-3). The diagram edges are considered undirected at this stage. The direction is determined when we construct the paths that lead from the root to the leaves. The root is the relation for which the preference is defined and the leaves are the value nodes (in case of selection preferences) or the most distant relation from the root (in case of join preferences). The algorithm computes the set P of paths on the preference diagram from the root relation to all leaf nodes performing a breadth-first traversal of the diagram (ln:4).

5.1 Preference-Network Integration

Algorithm PFC builds the network of the top k preferences that are related to a query proceeding in a step-wise manner: for each new top-k preference, it establishes its relationships with the preferences found in the previous rounds. In this section, we focus on this problem, i.e., the integration of a preference p into a network $G_H(V_H, E_H)$ of related preferences.

Problem statement. Given a preference p and a preference network $G_H(V_H, E_H)$, the result of their integration is a new network $G'_H(V'_H, E'_H)$, such that:

- $V'_H = V_H \cup \{p\}$,
- $\forall p_i, p_j \in V'_H$ with $p_i \sqsubseteq p_j, \exists e(p_i, p_j) \in E'_H$.

In other words, the result of integrating a preference into a preference network is a network, i.e., containing only minimally overriding relationships, that captures all such relationships that exist between the new preference and the preferences already in the network.

Algorithm PATHS

```
Input:    a preference $p$
Output:   its set of root-to-leaf paths $P$
Begin
1.   $V := \varnothing$ , $E := \varnothing$
2.   Foreach join condition $q_i$ in $p$ between relations $R$ and $R'$   {
2.1.     If there is no node in $V$ mapping to $R$   { add $R$ to $V$ }
2.2.     If there is no node in $V$ mapping to $R'$   { add $R'$ to $V$ }
2.3.     add edge $e(R,R')$ in $E$                                     }
3.   Foreach selection condition $q_i$ in $p$ on relation $R$            {
3.1.     If there is no node in $V$ mapping to $R$   { add $R$ to $V$ }
3.2.     for the attribute $A$ in $q_i$: add $A$ to $V$
3.3.     for the value $v$ in $q_i$: add $v$ to $V$
3.4.     add edge $e(R,A)$ in $E$
3.5.     add edge $e(A,v)$ in $E$                                      }
4.   $P :=$ BFS( $V$, $E$)
5.   output $P$
End
```

Fig. 9. Finding the root-to-leaf paths on a preference diagram.

Algorithm PNET. Integrating a preference into a network comprises two problems: (*a*) how to identify a $\sqsubseteq$ relationship that involves this preference and some other preference in the network and (*b*) how to traverse the network in order to find all such relationships between the preference under consideration and the preferences in the network.

In order to find the $\sqsubseteq$-relationships between a new preference p and the preferences in a network $G_H(V_H, E_H)$, algorithm PNET employs a set of patterns, each one identifying one or more $\sqsubseteq$-relationship using $\sqsubset$-relationships.

- *root pattern*: (p_r is a root in G_H with $p \sqsubset p_r) \Longrightarrow p \sqsubseteq p_r$
- *leaf pattern*: (p_i is a node in G_H with $p_i \sqsubset p$ and p_i is a leaf or all its children are independent with $p) \Longrightarrow p_i \sqsubseteq p$
- *intermediate node pattern*: (p_i, p_j are nodes in G_H with $p_i \sqsubset p \sqsubset p_j$ and $p_i \sqsubseteq p_j) \Longrightarrow p_i \sqsubseteq p \sqsubseteq p_j$

The first pattern shows the case of p taking over the role of root from an earlier root. The second pattern shows the case of p becoming a leaf in the network. The last pattern captures the case of p being added between two other nodes. Based on this pattern, given two preferences p_i and p_j in the network with p_i being minimally overridden by p_j, if the new preference p overrides p_i and is overridden by p_j, then the edge between p_i and p_j is replaced by two edges that essentially place p between the two preferences. Clearly, as algorithm PNET may be called for a sequence of preferences, relationships in the network are added or replaced in light of each new preference considered. If a new preference does not have any relationships with the other preferences, then it is independent from all preferences currently in the network and becomes a new root-leaf in the network.

In order to establish all relationships between preference p and the preferences of the network, all preferences and edges in the network need potentially to be examined. A network may contain multiple roots, i.e., preferences that do not override others (Section

```
Algorithm PNET
Input:    a preference network $G_H(V_H, E_H)$, preference $p$
Output:   a preference network $G_H(V_H, E_H)$
Begin
1.      $is\_root$ := true
2.      $RQ := \varnothing$
3.      While $\exists$ root preference $p_r \in V_H$ not examined                    {
3.1.      $Relationship$ := FIND_REL($p, p_r$)
3.2.      If $Relationship$ = '$p$ is overridden by $p_r$'              {
3.2.1.      create edge $e(p, p_r)$ in $E_H$
3.2.2.      unmark $p_r$                                          }
3.3.      If $Relationship$ = '$p_r$ is overridden by $p$'              {
3.3.1.      $is\_root$ := false
3.3.2.      If $\nexists$ visited edge $e(p_r, *)$ in $E_H$     {
              create edge $e(p_r, p)$ in $E_H$       }
            Else                              {
              Foreach not visited edge $e(p_r, p_i)$ in $E_H$
                add $(p_r, p_i)$ in $RQ$
              add $(p_r, -)$ in $RQ$
              $inserted$ := false                    }              }  }
3.4.      While $RQ <> \varnothing$                                   {
3.4.1.      get head element $(p_s, p_i)$ from $RQ$
3.4.2.      If $p_i = -$ and $inserted$ = false                    {
              create edge $e(p_s, p)$ in $E_H$
              $inserted$ := true                              }
            Else                                  {
              $Relationship$ := FIND_REL($p, p_i$)
              If $Relationship$ = '$p$ is overridden by $p_i$'   {
                remove edge $e(p_s, p_i)$ from $E_H$
                create edge $e(p_s, p)$ in $E_H$
                create edge $e(p, p_i)$ in $E_H$
                $inserted$ := true                          }
              If $Relationship$ = '$p_i$ is overridden by $p$'   {
                If $\nexists$ visited edge $e(p_i, *)$ in $E_H$   {
                  create edge $e(p_i, p)$ in $E_H$
                  $inserted$ := true                   }
                Else                        {
                  Foreach not visited edge $e(p_i, p_j)$ in $E_H$
                    add $(p_i, p_j)$ in $RQ$
                  add $(p_i, -)$ in $RQ$                  }         }   }   }
4.      If $is\_root$ = true
4.1       { mark node $p$ as a root  }
5.      output $G_H(V_H, E_H)$
End
```

Fig. 10. Integrating a preference into a network.

3.2). Hence, preferences that are minimally overridden ($\sqsubseteq$) by p, or vice versa, may be found at different places in the network, under the same or different roots. Taking this into

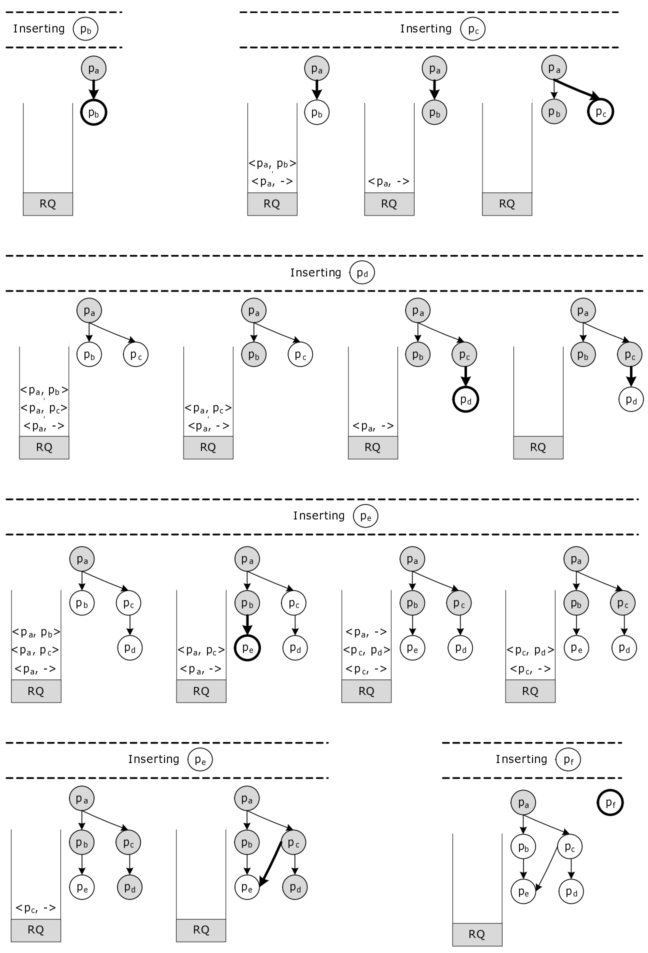


Fig. 11. Examples of preference-network integration.

account, algorithm PNET starts from each root and performs a breadth-first traversal of the corresponding subgraph. To avoid an exhaustive traversal of the network, two pruning rules are used:

- Subgraphs with different roots may overlap. If an edge is already visited then the underlying subgraph is already explored and it can be safely pruned.
- The subgraph starting from any node in the network is not explored if the preference mapping to this node is independent from or overrides p.

Based on the patterns presented, which capture all cases of p's position w.r.t. other preferences in the network and the traversal strategy, algorithm PNET correctly places a preference in a network by establishing *all* $\sqsubseteq$-relationships with the existing preferences in the network and the result of the integration is a network.

The algorithm is presented in Figure 10. It integrates a preference p into a preference network $G_H(V_H, E_H)$. A queue RQ keeps edges to be examined. These are edges that have not been visited before. New edges are always added in *RQ*'s tail. FIND_REL is responsible for finding the relationship of a pair of preferences and it will be described in Section 5.2.

The algorithm examines the relationship of p with each root p_r in the network (ln: 3.1). It first tries the root pattern (ln: 3.2). If p is overridden by p_r, then the property of being root is transferred from p_r to p. If p overrides p_r, then the algorithm tries the leaf pattern (ln: 3.3.2): if there are no outgoing edges from p_r, then p becomes a leaf under p_r. Otherwise, all edges from p_r to its children are added in the queue along with a dummy edge $(p_r, -)$ and the algorithm will investigate p's position w.r.t. the other preferences in p_r's underlying subgraph.

The algorithm goes down a subgraph as long as there is an edge $e(p_s, p_i)$ in *RQ* (ln: 3.4). If this is a dummy edge, $e(p_s, -)$, then the algorithm tries to apply the leaf pattern: if p has been found independent from p_s's children (indicated by *inserted* = false), then p becomes p_s's new child. Hence, a dummy edge is used in combination with the flag *inserted* to show when p should become a new child of an intermediate node in the network. If $e(p_s, p_i)$ is an actual edge in the network, then:

- If p is overridden by p_i (intermediate node pattern), then the algorithm 'breaks' the edge between p_i and its predecessor, and creates two edges, one connecting the predecessor to p and one from p to p_i. Having found the position of p in the subgraph, *inserted* becomes true and no outgoing edges from p_i are added in *RQ*, i.e., the subgraph under p_i is pruned.
- If p_i is overridden by p, then this node's outgoing edges are added in *RQ* along with a dummy edge $(p_i, -)$. If there are no outgoing edges, then p becomes p_i's child and again *inserted* becomes true.

Finally, if p does not override any other preference (*is_root* = true), it becomes a root.

Example. Figure 11 presents the construction of the preference network depicted in Figure 5. Preferences p_a to p_f are presented to the algorithm in that order. Each block in the figure shows the steps required for inserting one preference in the network. Each step is described by the contents of *RQ* and the status of the network, with nodes examined already indicated with gray background. For instance, the first step for inserting p_c visits the root p_a, which is overridden by p_c. Hence, its outgoing edges plus a dummy edge are placed in *RQ*. In the second step, the edge going to p_b is obtained from *RQ* and as a result p_b is visited. This one is independent from p_c, hence the algorithm will not search below this preference. Finally, pulling out the dummy edge $< p_a, - >$ marks the end of p_a'children examination and since p_c has been found independent from all of them, it is connected to p_a.

Preference p_e provides an example of how the algorithm moves until it establishes all relationships that p_e participates. The first relationship (i.e., with preference p_b) is found in the second step, but the algorithm goes on exploring the subgraphs starting from p_b's siblings, and ends in discovering the second relationship (i.e., with preference p_c). Finally, observe how p_f becomes a root. □

5.2 Relationship Finder

A preference is overridden by another preference if they are both defined over the same relation and there is a mapping of the atomic conditions from one preference to the other

(Section 3.2). Preferences are represented as graphs. Our approach for identifying the relationship between two selection preferences, p and p', defined over the same relation R, is the following. We consider the sets P and P' of all root-to-leaf paths on their preference diagrams, respectively. $|P|$ and $|P'|$ are the sizes of these sets. For efficient path counting and comparison, we adopt a string representation of a path (string encoding of graphs in general has been proposed in [Zaki 2005].) To generate a path representation, we concatenate the names of nodes in the path. For instance, the string representation of p_a in Figure 5, which comprises a single path, is "Myear2000". Two paths $s_i \in P$ and $s'_i \in P'$ *match* iff their string representations are the same. Then, the relationship of p and p' can be determined by counting the number M of pairs (s_i, s'_i) of matching paths. The following cases are distinguished:

- If $M = |P|$ it is $p \sqsubset p'$.
- If $M = |P'|$ then $p' \sqsubset p$.
- If none of the above holds, p' and p are independent.

This process is captured in algorithm FIND_REL, shown in Figure 12. In case of selections containing inequalities, the process is slightly different: it matches paths without considering the selection values, and performs an additional check for the atomic selection conditions to determine their relationship. For presentation purposes, in subsequent algorithms, we assume selections with equalities.

The correctness of the algorithm stems directly from the definitions of preference diagram (Section 3.1) and preference overriding (Section 3.2). p is overridden by p' if each atomic condition in p is mapped to an atomic condition in p' with the same relations and attributes. Since selection preferences map to rooted graphs, where the root is a relation and the leaves are always values, the problem of finding the relationship of two selection preferences is translated to a mapping of the paths that connect the root to the leaves in the two preference diagrams.

5.3 Preference Construction Analysis

Preference construction involves the following tasks: extraction of the top k preferences from the user profile that are related to and not conflicting with a given query, identification of the relationships that exist among these preferences, and construction of the network that captures the preferences and their relationships.

Algorithm FIND_REL compares two preferences by comparing their respective sets of paths. We store sets of paths in memory as hash tables. The algorithm starts with the "smaller" preference, i.e., the one that has the fewest paths (ln:2, Figure 12) and it looks in the hash table of the larger set of paths for matching paths. Due to the use of hash tables, the matching cost per path is $O(1)$. We consider that P_o is the maximum number of paths of any preference in the user profile. Hence, the cost for one invocation of FIND_REL is $O(P_o)$. Naturally, we do not expect P_o to take very large values. In the cases we have examined, we found P_o ranging from 1 to 4.

Algorithm PNET compares a preference to a set of preferences, for which their relationships are already known. It is called k times from PFC, once for each of the top k preferences produced. When PNET examines the j^{th} preference ($j = 1$ to k), it performs a maximum of $j - 1$ comparisons. Each comparison is performed with the help of algorithm

```
Algorithm FIND_REL
Input:    preference $p$, preference $p'$
Output:   relationship
Begin
1.  read the sets $P$ and $P'$ of root-to-leaf paths for $p$, $p'$, resp.
2.  If  $|P| \leq |P'|$       {relationship := MATCH($P$,$P'$) }
    Else                      {relationship := MATCH($P'$, $P$) }
3.  output <$p$ relationship $p'$>
End
```

```
procedure MATCH
Input:    set $P_1$, set $P_2$
Output:   relationship
Begin
1.  $M$ := 0; found := true
2.  While found = true and $\exists$ unexamined path $s_i \in P_1$     {
2.1   If $s_i$ matches some $s'_i \in P_2$     {$M$ := $M$+1  }
2.2   Else                                     {found := false }      }
3.  If $M = |P_1|$
3.1    If $M = |P_2|$      {relationship := ‘is the same to’ }
3.2    Else                {relationship := ‘is overridden by’ }
4.  Else    {relationship := ‘is independent from’ }
4.  output relationship
End
```

Fig. 12. Identifying preference relationships.

FIND_REL. Hence, it is $O(j * P_o)$.

Algorithm PFC extracts the top k preferences from a user profile. The algorithm first extracts all related preferences to Q that are found in the profile U (ln:2). If relations are not repeated in Q, then it should be $O(|Q|+|U|)$, where $|Q|$ is the size of query Q in terms of relations and $|U|$ is the size of the profile in terms of preferences. Assuming the profile is hashed on the relation for which each preference is defined, then for every query relation the algorithm finds the list of related preferences in $O(1)$ time and then inserts into QP all preferences of that relation. Duplicate insertion can be avoided by maintaining a hash table of all relations examined so far. Hence, there are $|Q|$ probes in the two hash tables and at most $|U|$ preferences inserted into QP (if all preferences are related to the query). If we use $|U_Q|$ to signify the subset of $|U|$ with the related and not conflicting preferences then the complexity is exactly $O(|Q|+|U_Q|)$.

The main loop in PFC (ln:3) will be executed as many times as there are preferences added to QP, which are bound by $|U|$ again, or more precisely, by the size of the transitively composeable subset of it $|U_Q^*|$. For k of these iterations, the cost of each one will be that of ln: 3.2.1, which is $O(1)$ and the cost of a call to PNET, which we have analyzed above. For the remaining iterations, which should be $O(|U_Q^*|)$ in number, making the reasonable assumption that k< $|U_Q^*|$, the cost of each one is the sum of:

—the cost for testing for conflicts, which can be considered equal to $O(P_o)$. Recall that P_o is the maximum number of paths of any preference in the user profile. Since all

Algorithm EXCLUDE_COMBINE

```
Input:    query $Q$, a constraint $\mathfrak{l}$
          a preference network $G_H(V_H, E_H)$
Output:   personalized results $Results$
Begin
1.  $Results := \varnothing$
2.  Foreach $p_i$ in $V_H$                                {
2.1.   $Results(p_i) := execute\_query(Q \wedge p_i)$             }
3.  If $\mathfrak{l} = 0$                          {
3.1.    $Results(Q) := execute\_query(Q)$
3.2.    $\mathfrak{l} := 1$
3.3.    $\mathfrak{l}_{copy} := 0$                          }
4.  Foreach not visited $e(p_i, p)$ in $E_H$                  {
4.1.    $Results(p_i) := Results(p_i) - Results(p)$
4.2.    mark $e$ visited                                 }
5.  While at least $\mathfrak{l}$ non-empty preference result sets exist          {
5.1.   remove the head with the highest $tid$ among all result sets
5.2.   If $tid$ satisfies at least $\mathfrak{l}$ preferences          {
5.2.1.     $doi(tid) := h(\{d_i \mid d_i = doi(tid \in Results(p_i)),$
                               $\forall Results(p_i)$ having $tid\})$
5.2.2.     add $(tid, doi(tid))$ in $Results$              }              }
6.  If $\mathfrak{l}_{copy} = 0$   { $Results := Results \cup Results(Q)$ }
7.  output $Results$
End
```

Fig. 13. Generating personalized results - EXCLUDE_COMBINE.

implicit preferences are generated by composing a join preference (that is always a single path) to another join or selection preference, P_o is the maximum number of paths in any preference, original or generated.

—the cost for generating the path combinations, which for each iteration is $O(P_o)$, since a join preference from which we compose other preferences has only one path.

Hence, the total cost of PFC is $O(|Q| + |U_Q| + \mathrm{k} * cost(\text{PNET}) + |U_Q^*| * P_o)$, which becomes $O(|Q| + |U_Q| + \mathrm{k}^2 * P_o + |U_Q^*| * P_o)$ and finally $O(|Q| + (\mathrm{k}^2 + |U_Q^*|) * P_o)$.

6. PERSONALIZED QUERY ANSWERING

Given a query Q and a set of k related preferences organized in a network, the last step of personalization is responsible for returning *all* query results that (*a*) satisfy at least some of the preferences, i.e., $\mathfrak{l} \in [0..\mathrm{k}]$ preferences, (*b*) respect the preference relationships, and (*c*) are ranked with the help of a ranking function h (Section 3.3). When $\mathfrak{l} = 0$ then all the results of the initial query are returned whereas when $\mathfrak{l} > 0$, the personalized answer may be smaller.

Relying directly on SQL to capture these semantics can lead to complex and time-consuming queries. The disadvantages of such approaches to query personalization have been studied in the literature [Koutrika and Ioannidis 2005b], and hence are not discussed any further. Here, we describe two new approaches. These approaches work directly on the preference network. They take into account the containment mappings captured in the

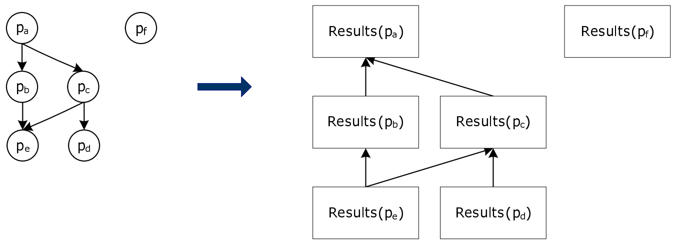


Fig. 14. Mapping a preference network to a network of result sets.

network in order to generate and rank results with respect to the preference relationships. The first algorithm (EXCLUDE_COMBINE) is based on the observation that an edge from p_i to p_j in the preference network implies a difference operation $Results(p_i) - Results(p_j)$ on the respective result sets in order to find the results that satisfy p_i but not p_j (i.e., w.r.t. the overriding relationship of p_i and p_j). Hence, instead of executing complex queries, it executes a set of simpler queries that map to the preferences in the network and capture these 'local' difference operations and combines the partial results to find those that satisfy at least l preferences.

Algorithm EXCLUDE_COMBINE. The algorithm executes a number of simple queries. Each query Q_i is a combination of the initial query Q and a preference $p_i (i = 1...k)$ in the current network and returns the set $Results(p_i)$ of Q's results that match p_i. In this way, we go from a network of preferences to a virtual "network of partial result sets", where nodes and edges have the following meaning:

- each node $Results(p_i)$ represents the results that satisfy preference p_i (without taking into consideration any preference relationships)
- for each edge from p_i to p_j in the preference network, there is an edge with the opposite direction in the network of result sets, i.e., from node $Results(p_j)$ to node $Results(p_i)$, which implies a difference operation $Results(p_i) - Results(p_j)$ on the respective result sets in order to find the results that satisfy p_i but not p_j (i.e., w.r.t. the overriding relationship of p_i and p_j).

Then, for each edge from p_i to p_j, the algorithm removes from $Results(p_i)$ any results found also in $Results(p_j)$. These difference operations are sufficient to ensure that the tuples remaining in each result set will be those that satisfy the respective preference (i.e., the preference that initially generated this set) but not any preference more specific than that. The reason is that, by definition, $Results(p_j)$ contains all results corresponding to preferences more specific than p_j. Hence by removing $Results(p_j)$'s tuples from $Results(p_i)$, we satisfy all preference relationships between p_i and any preferences more specific than p_j. In the end, all tuples that occur in at least l sets are those that satisfy at least l preferences, and comprise the final answer, which is ranked based on the preferences satisfied using a specified ranking function h.

Algorithm EXCLUDE_COMBINE, presented in Figure 13 implements this idea and for a query Q and a network $G_H(V_H, E_H)$ of related preferences, it generates all the tuples that satisfy at least l preferences w.r.t. the preference relationships in G_H. The algorithm proceeds as follows.

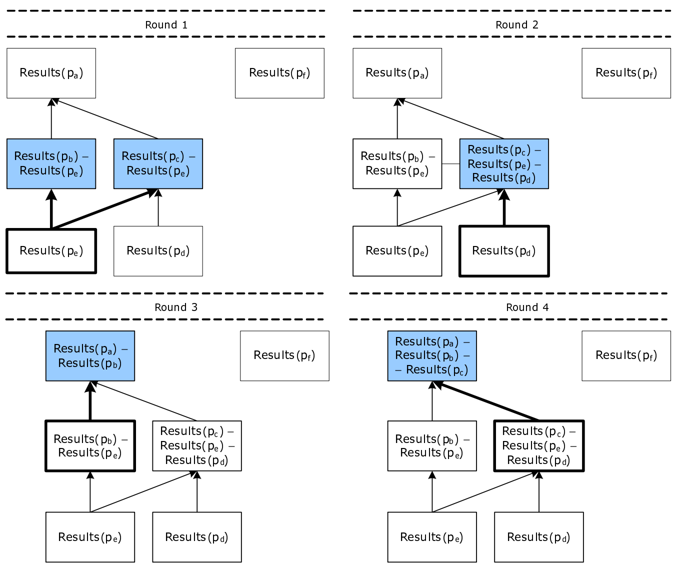

Round 1
Round 2
Round 3
Round 4
Results(pa)
Results(pf)
Results(pb) – Results(pe)
Results(pc) – Results(pe)
Results(pe)
Results(pd)
Results(pc) – Results(pe) – Results(pd)
Results(pa) – Results(pb)
Results(pa) – Results(pb) – Results(pc)

Fig. 15. Excluding results that satisfy more specific preferences.

- (*Exclude*) It generates the partial result sets, each one satisfying a single preference (ln: 2). Each set is ordered on the tuple id *tid*. Then, it traverses the network and excludes from each set all tuples that satisfy other, more specific, preferences (ln: 4).
- (*Combine*) As long as there are $\mathfrak{l}$ non-empty sets, the algorithm removes the greatest *tid*. If it satisfies at least $\mathfrak{l}$ preferences, then its degree of interest $doi(tid)$ is computed from the preferences corresponding to these sets and $(tid, doi(t))$ is added in the *Results*, which is kept ordered on the degree of interest (ln: 5).

When $\mathfrak{l} = 0$, all the results of the initial query are returned and are ranked based on the preferences in the network. In that case, the algorithm executes query Q (ln: 3.1). The tuples returned, $Results(Q)$, have default degree equal to zero (since at this point, we do not know if they satisfy any preferences.) The algorithm also sets $\mathfrak{l}$ equal to 1 (ln: 3.2). This makes sure that the combine step (ln: 5) will be executed only if there is at least one preference satisfied, i.e., if there is at least one non-empty preference result set $Results(p_i)$. Then, any results of Q that that have not been ranked during the combine step are added to the results with degree of interest equal to 0 (ln: 6).

Example. We consider the preference network of Figure 5. The algorithm first generates the partial result sets, each one satisfying a single preference and maps the preference network to a network of results sets, as shown in Figure 14. On this network, we can for instance observe that taking into consideration the edge from $Results(p_d)$ to $Results(p_c)$, $Results(p_c) - Results(p_d)$ is the 'actual' set of results for which p_c should hold because the more specific p_d does not hold for them. Then, the algorithm proceeds as shown in Figure 15 excluding from each set the non-qualifying tuples, i.e., those that satisfy a more specific preference. □

Algorithm EXCLUDE_COMBINE executes all possible queries that map to the nodes of the preference network. The second algorithm (REPLICATE_DIFFUSE) that we present

Algorithm REPLICATE_DIFFUSE

```
Input:      query $Q$, a constraint $\mathfrak{l}$
            a preference network $G_H(V_H, E_H)$
Output:     personalized results $Results$
Begin
1.  $Results := \varnothing$
2.  If $\mathfrak{l} = 0$    { $Results(Q) := execute\_query(Q)$ }
3.  Foreach root $p_{rt} \in V_H$ in order of selectivity    {
3.1.    $Results(p_{rt}) := execute\_query(Q \wedge p_{rt})$
3.2.    Foreach $tid \in Results(p_{rt})$ with $tid \notin Results$    {
3.2.1.      $P := \{p_{rt}\}$
3.2.2.      $execute\_query(Q_{rt}(tid))$
3.2.3.      add to $P$ all roots satisfied by $tid$
3.2.4.      While $\exists p \in P$ with $e(p, p_i) \in E_H$    {
                overridden := false
                Foreach $e(p, p_i) \in E_H$ with $p_i \notin P$    {
                    If $execute\_query(Q_i(tid)) \neq \varnothing$    {
                        overridden := true
                        add $p_i$ in $P$    }    }
                    If overridden = true    {
                        remove $p$ from $P$    }    }
3.2.5.      If there are at least $\mathfrak{l}$ preferences in $P$    {
                $doi(tid) := h(P)$
                add $(tid, doi(tid))$ in $Results$    }
3.2.6.      remove $tid$ from $Results(Q)$    }    }
4.  If $\mathfrak{l} = 0$    { $Results := Results \cup Results(Q)$ }
5.  output $Results$
End
```

Fig. 16. Generating personalized results - REPLICATE_DIFFUSE.

below tries to minimize the number of such queries based on the following observation: given a preference network, any tuple that satisfies any preference at any level in the network satisfies (at least) one of the root preferences. Hence, we can execute only the queries corresponding to the roots of the network and then perform 'look-ups' to find the specific preferences satisfied by each retrieved tuple. In addition, the algorithm works with a hopefully smaller set of tuples than EXCLUDE_COMBINE.

Algorithm REPLICATE_DIFFUSE. This algorithm is based on the idea of visualizing a preference network as a system of pipes with preference nodes acting as safety valves. When a tuple enters the system at some node (as a result of satisfying the corresponding preference), it rolls down the pipes (edges) starting from this node as long as there is a safety valve that can be "opened". A safety valve will remain closed to a tuple, if the latter satisfies the preference corresponding to the valve, but the valve leads to no other preferences that can be satisfied. Moreover, a tuple satisfying a preference at any level of a network satisfies its ancestors too. This means that any tuple satisfying at least one preference in the network will "enter" the system from the network's roots and roll down following edges from general to more specific preferences, until it is collected by valves mapping to the most specific preferences satisfied by it.

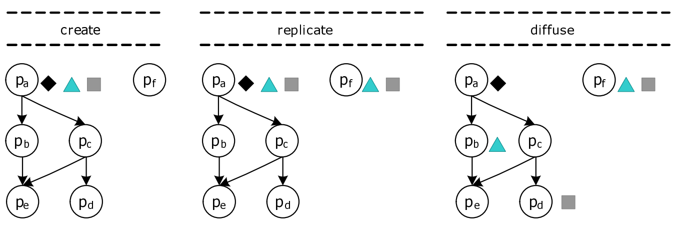


Fig. 17. Example of replicate-diffuse steps.

Algorithm REPLICATE_DIFFUSE implements this idea in three steps, repeated for each root of a given network. It creates a set of queries, each one combining the user query Q with a root preference, in order of increasing selectivity.

- (*Create*) For each root, the algorithm first executes the respective query and creates the set of results that satisfy the root preference (ln: 3.1).
- (*Replicate*) For each tuple in this set, it finds which root preferences following the current one in order of selectivity are also satisfied (ln: 3.2.2).
- (*Diffuse*) Then, it "lets" tuples satisfying the root preferences to roll down the network. In the end, all tuples are found only in nodes corresponding to the most specific, independent preferences they satisfy (ln: 3.2.4).

Example. Figure 17 illustrates these steps for the root p_a. The algorithm retrieves Q's tuples that satisfy p_a, i.e., three tuples depicted as different shapes. Next, it examines whether any of these satisfy p_f. Assume that the triangle and square tuples do. To illustrate that, they are shown replicated on this node. Then, all tuples move freely down the network until they get stopped by some valve. For instance, we find that the square tuple satisfies the independent preferences p_d and p_f. $\Box$

The algorithm uses a query Q, a constraint l, and a network $G_H(V_H,\ E_H)$ of related preferences to generate personalized results for Q. For each root p_{rt} in the network, it executes a query $Q \wedge p_{rt}$. Each tuple (with id tid) returned by the query $Q \wedge p_{rt}$ is processed only once.

For each tuple processed, the algorithm performs the following steps. It keeps a set P of preferences satisfied by tid. Initially, this set contains p_{rt}. In order to find which other preferences are satisfied by tid, the algorithm executes a parameterized query $Q_{rt}(tid)$, which checks whether tid satisfies any other root preference following p_{rt} in order of selectivity (ln: 3.2.2). This query returns zero or more occurrences of tid, depending on the number of root preferences that are satisfied by tid. All root preferences satisfied by tid are placed in P (ln: 3.2.3).

Then, for each preference p in P, the algorithm checks whether there are more specific preferences that override p. For each edge $e(p, p_i)$, the algorithm executes a parameterized query $Q_i(tid)$ that checks whether tid satisfies p_i. If that happens, then p_i is inserted in P. A preference p stays in P if tid does not satisfy any more specific preference p_i. Note that when considering an edge $e(p, p_i)$, if p_i is already in P, this means that $Q_i(tid)$ has been executed in a previous step (that happens when p_i overrides more than one preference.) In that case, the algorithm does not execute $Q_i(tid)$ again. At the end of this process, P contains the most specific, independent preferences satisfied by tid. If there are at least l

such preferences, then the degree of interest in *tid* is computed from *P* using a function *h* and *tid* is inserted in the final results (ln:3.2.5).

When l = 0, all the results of the initial query are returned and are ranked based on the preferences in the network. In that case, the algorithm executes query Q (ln: 2). The tuples returned, $Results(Q)$, have default degree equal to zero as explained for algorithm EXCLUDE_COMBINE. Any tuple id *tid* that is found to satisfy some preference in the network gets removed from $Results(Q)$ (ln: 3.2.6). At the end, any remaining results in $Results(Q)$ are added to the results with degree of interest equal to 0 (ln: 4).

6.1 Personalized Query Answering Analysis

We can decompose algorithm EXCLUDE_COMBINE into a number of operations, some of which involve processing of arbitrary queries (ln:2 and 3), whose cost we cannot estimate analytically.

(a) To generate the partial results sets (ln:2), the algorithm executes a query $Q \wedge p_i$, for each preference p_i, $i = 1...$k. In the case of non-personalizing the output tuples (ln:3), it also executes the original query Q on its own (for ease of reference, we may denote it as $Q \wedge p_0$, where $p_0 =$ '*true*'). The total cost of these operations is:

$$\sum_{i=0}^{k} cost(Q \wedge p_i)$$

(b) After the results are gathered in main-memory hash tables, the algorithm executes r difference operations (ln:4), where r is the number of preference relationships, i.e., the edges in the network. Assuming that each result set contains at most n tuples, then each such difference can be executed in O(n) time, for a total of O(r*n) time.

(c) Finally, the algorithm reads the in-memory results to output those satisfying at least l preferences. Each iteration of the loop (ln:5) requires O(k) time: (ln:5.1) examines each one of the k result sets to remove the highest-tid common head, while (ln:5.2) uses at most k doi's to compute the overall doi of the tid examined. In the worst case, the loop of (ln:5) will consume all tuples in the results, of which there are at most k*n, and will do that by removing the smallest possible number of them each time, which is equal to l, this way maximizing the number of iterations required. Hence, the worst-case total time required by this operation is $O(\mathrm{k}^2/\mathrm{l}*n)$. Taking all the above into account, the total cost of algorithm EXCLUDE_COMBINE, is

$$\sum_{i=0}^{k} cost(Q \wedge p_i) + O(r*n + k^2/l*n).$$

Similarly to the above analysis, for algorithm REPLICATE_DIFFUSE, we consider the following:

Let v and r be the number of nodes and edges in the network, respectively.

(a) For each root p_{rt} (ln:3.1), the algorithm executes a query $Q \wedge p_{rt}$. In the case of non-personalizing the output tuples (ln:2), the algorithm executes the original query Q on its own as well (again, for ease of reference in this case, let us assume that a preference

$p =$ '$true$' is a root as well). The total cost of all these query executions is:

$$\sum_{p_{rt}\ is\ root} cost(Q \wedge p_{rt})$$

(b) For each tuple returned from the queries above, the algorithm first executes a parameterized query $Q_{rt}(tid)$ (ln: 3.2.2) and then parameterized queries $Q_i(tid)$ (within ln: 3.2.4) to check which other (less selective) root preferences or other preferences down the network, respectively, the tuple satisfies. Note that $Q_{rt}(tid)$ is equivalent to running a $Q_i(tid)$ for each of the roots that follow the one being considered in the main loop of the algorithm. Each tid undergoes this process only once and is used as a parameter for each query only once as well. Hence, assuming there are m tuples in all root query results together and that the cost of each parameterized query remains the same independent of the tid used as a parameter, in the worst case (when all tuples satisfy all preference queries), the total cost of these operations is:

$$m * \sum_{i=1}^{k} cost(Q_i(.))$$

(c) Besides query executions, the main-memory operations of the algorithm are inside the loop of (ln: 3.2). For each one of its m iterations, they are performed either once for every one of the r edges of the network (ln: 3.2.4 - top) or once for every one of the v preference-nodes of the network (ln: 3.2.4 - inner foreach-loop). In addition, each iteration uses again at most k degrees of interest to compute the overall degree of interest of the tid examined (ln:3.2.5). Hence, the worst-case total time required by these operations is $O(m*(r+v+k))$.

As with algorithm EXCLUDE_COMBINE, taking all the above into account, the total cost of algorithm REPLICATE_DIFFUSE is

$$\sum_{p_{rt}\ is\ root} cost(Q \wedge p_{rt}) + m * \sum_{i=1}^{k} cost(Q_i(.)) + O(m*(r+v+k)).$$

7. EXPERIMENTS

The novelty of our framework stems from allowing both generic and specific preferences to be explicitly stated in a profile with user-specific, "freely" selected, degrees of interest assigned to them. Earlier approaches (e.g., [Koutrika and Ioannidis 2004; Stefanidis et al. 2007]) capture only simple preferences, i.e., preferences on single attributes, and rely on special mechanisms to derive more complex preferences from simpler ones. Two questions naturally arise:

- "*Simplicity or expressiveness*?" One question is whether the proposed model can have a higher impact than an approach to preference modeling that enables capturing simple preferences (i.e., in the form of general rules that hold for a user) and derive more complex user preferences using appropriate preference composition and inference mechanisms. One can argue that "simple is beautiful". We will test this argument through a user study (Section 7.1).
- "*Expressiveness or performance*?" When a profile contains a mix of generic and more specific preferences, preferences cannot be freely combined. We need sophisticated

algorithms in order to find which preferences can be combined and how to generate and rank the results of a query taking into account relationships among a set of preferences. Is expressiveness achieved at the expense of performance and to what extent? If there is a tradeoff, does it justify the use of this framework instead of a simpler one that assumes independent preferences? We will investigate these questions through an extensive experimental evaluation of our algorithms (Section 7.2).

7.1 User Study

Existing approaches (e.g., [**?**]), formulate structurally (and semantically) simple preferences, i.e., over a single attribute, and they provide mechanisms to derive complex preferences based on their simpler, constituent ones. We believe that mechanisms for composition of complex preferences cannot always capture the real complex user preferences. We need to enable complex preferences to be explicitly stated (not be solely derivatives of other preferences). The objective of the user study is to highlight this problem of accuracy in existing approaches and compare them to our approach.

We conducted an empirical evaluation of our approach with 11 human subjects with or towards a diploma in computer science. We built a database containing information about over 480,000 movie titles that we obtained from IMDB (http://www.imdb.com/interfaces). Our schema contains 22 relations storing information about movies, actors, directors, ratings, writers, and so forth.

We built a web interface that allowed users to manually create their preference profiles and perform searches. In order to gain insight as to the appropriateness of the proposed preference model and its benefits for query personalization compared to flat preference representations, each user manually provided two profiles in the system, one containing only independent preferences (FLAT_PROFILE) following the model presented in [Koutrika and Ioannidis 2004] and a more elaborate one following the model presented in this paper (NET_PROFILE). Figure 19 shows part of one of the profiles that we have in the system.

An initial test for the appropriateness of our model took place during the creation of these profiles. We explained to the users that they had two options: either describing complex preferences (such as a preference for comedies with Woody Allen) or simple (independent) preferences (such as a preference for comedies and a preference for Woody Allen). In the latter case, we will infer their more complex preferences based on what we know about the people's generic preferences. We will see how well we can do and whether the model we have presented in this paper, which allows expressing explicitly complex preferences, is more beneficial.

8 out of 11 people started with the first option, 2 started with simple preferences and decided that they had more complex preferences to express and only 1 formulated simple preferences. Once done with their profile (in either form), all users were asked to create a second "view" of their preferences either more elaborate or more generic depending on what was their first profile. Hence, 10 people had to create a FLAT_PROFILE and 1 had to elaborate his preferences in a NET_PROFILE. The largest group of users complained for "re-formulating" their preferences as simpler ones expressing their concerns regarding the system's ability to provide accurate personalization when it relies on simple, partially correct, profiles. These observations seem to indicate that people tend to trust more a system that captures their preferences more accurately.

There is of course a psychological dimension in how people perceive preferences and personalization. For example, in our case, the user that had initially simple preferences explained that he would be probably content with a simple personalization approach separating the chaff from the seed. In addition, the type of networks expected in practice depends on the way user information is (explicitly or implicitly) collected. Different people have different types of preferences, depending on their background, their expectations etc. In our experiments, "full-fledged" networks that could be derived from the complex profiles had an average depth of 3, and the average number of preferences stored per profile was 21.

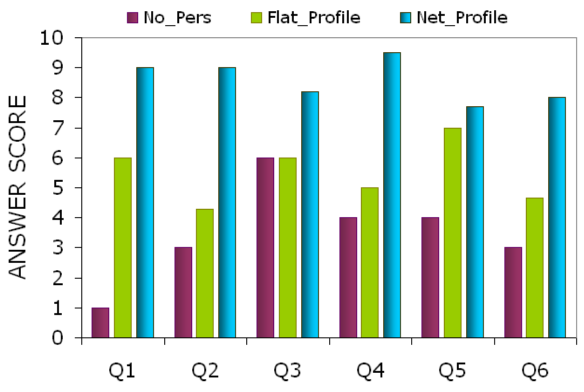
We have built a search interface over the database that allows a user to search for movies, actors or directors based on different criteria, such as the genre and the year of a movie. We discussed with the users about typical searches they would perform over a movie database and we made a pool of possible searches, from which we picked the 4 most popular queries among users. These included a search for short movies (Q_1), a search for movies based on the movie genre (Q_2), and a search for favorite directors after 2000 (Q_3). All subjects were asked to perform these 4 preselected searches plus 2 of their own choice.

In order to evaluate the effectiveness of our approach, we had to take into account several issues. We could not use traditional metrics, such as precision and recall. One reason is that we do not have complete information regarding which of the (possibly too many) results of a search are liked by a user. Even if we showed all results returned for a particular, non-personalized, search, users are not willing to browse more than one page of results. In addition, we wanted to get insights into how personalized search results compare to non-personalized results. However, there is no single uniform way to compare them all together. For example, in the case of the unranked results, all results are relevant to the query (by definition of the database query model) and there is no way to distinguish between them. On the other hand, when comparing ranked lists we are actually interested in how the results are ranked.

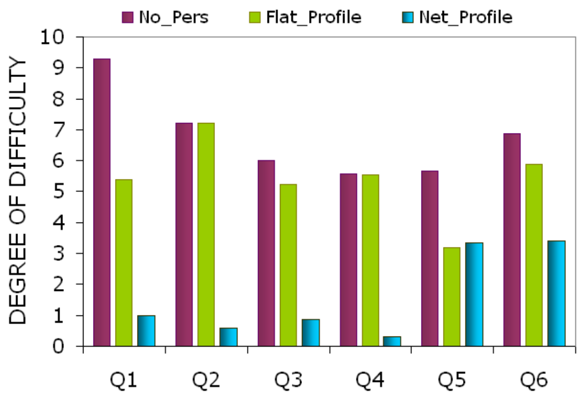
Taking all these issues into account, we proceed in two steps. First, we asked users to give scores to query answers as a whole, where an answer would contain at most 10 results, in order to have an overall picture of how personalized and non-personalized answers compare overall. Then, we focused on the most interesting part of the study which is how effective the rankings are based on different user representations.

For the first part of the evaluation, each user submitted these queries three times in arbitrary order. Queries were executed once without personalization (NO_PERS), once using the user's NET_PROFILE and once using the user's FLAT_PROFILE. The system randomly rotated these options so that the user would not be aware of the query processing performed and hence make an unbiased evaluation of query answers. As default parameters for personalization, we chose k to be half of the preferences in a user's profile and l=1. We ranked results based on the average degree of interest of the preferences satisfied and returned (up to) top-10 results. In the case of NO_PERS, the first 10 results for the query were returned. Users evaluated query answer using two scores measuring [Koutrika and Ioannidis 2004]: (a) the difficulty to find something interesting, if anything was found at all (DEGREE OF DIFFICULTY) and (b) how satisfactory was the answer (ANSWER SCORE) (both scores in the range [0, 10].)

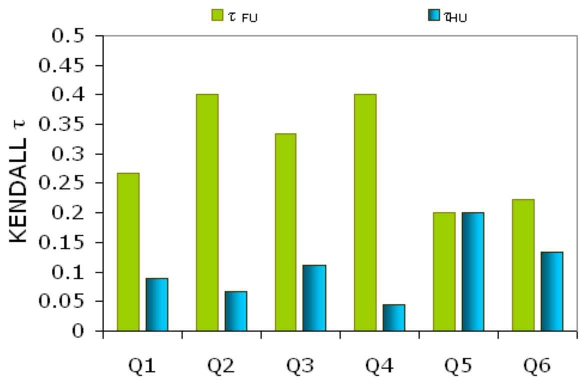
Figures 18(a) and 18(b) present the average answer score and degree of difficulty, respectively, per query for each of the three different runs, i.e., NO_PERS, NET_PROFILE and FLAT_PROFILE. The use of the complex profiles substantially reduces the difficulty to find interesting tuples within an answer and attracts distinctively higher answer scores.



(a) Answer score



(b) Degree of difficulty



(c) Result ranking

Fig. 18. Impact of composite preferences and benefits of query personalization.

Using the flat profiles improves answers (to a lesser degree than NET_PROFILE). However, we observe that often the improvement is marginal compared to NO_PERS and the degree of difficulty in many cases remains high.

During the study, users were also asked to re-rank the tuples returned for each query and put them in an order that they thought closer to their preferences. For the second part of the study, we compare the ordering of results using the complex profile or the flat profile to the user's ordering of the same results for each query. There are several standard methods for comparing two lists that are permutations, such as Kendall's tau distance (τ) [Fagin et al. 2003; Kendall and Gibbons 1990]. We used the normalized Kendall tau distance (τ) , which lies in the interval [0,1] with 1 indicating maximum disagreement. τ_{FU} compares the list of results based on the FLAT_PROFILE and the same list when re-ordered by the user. τ_{HU} compares the list of results returned using NET_PROFILE and the same list re-ordered by the user. Figure 18(c) plots the average distances for each query. We observe that when the FLAT_PROFILE was used, users often disagreed with the system-based ordering of results because it did not quite respect their real, more elaborate, preferences.

A closer look at the user-ordered results revealed that often 1 to 3 tuples appeared out of order in the system answers due to the less accurate profiles applied. On the other hand, finer-grained result rankings could be achieved with the complex profiles that captured more accurately user expectations. The accuracy achieved also depends on the query. For example, for the query *Q*5, none of the profiles had adequate (or different) information for differentiating the output of this query. The figure brings up another issue: one would expect that since the complex profiles captured more accurately the user preferences, τ_{HU} would be 0 in all cases. Users still moved tuples in the results for different reasons (e.g., they knew some additional information, such as reviews for certain movies, or while inspecting the results they were able to evaluate them based on properties they had not captured in their profiles.)

Below, we describe some interesting cases of personalized searches that we observed during the study in order to illustrate the impact of our approach. Figure 19 shows part of the profile for one of the participants. This profile was one of the most detailed ones. We observe that there are some general preferences for movie genres, actors and so forth, but there also finer-grained preferences. In one of the searches, the user asked for comedies after 2008 that satisfy at least one of her top 5 preferences. Her top 5 preferences are the following:

MOVIE.mid = CAST.mid and	
CAST.aid = ACTOR.aid and ACTOR.name = "G.Clooney"	
CAST.aid = ACTOR1.aid and ACTOR1.name = "B.Pitt"	1
MOVIE.mid = CAST.mid and	
CAST.aid = ACTOR.aid and ACTOR.name = "G.Clooney	
CAST.aid = ACTOR1.aid and ACTOR1.name = "E.McGregor"	
CAST.aid = ACTOR2.aid and ACTOR2.name = "K.Spacey"	1
MOVIE.year > 2000 and MOVIE.mid = GENRE.mid	
GENRE.genre = "animation"	0.95
MOVIE.mid = CAST.mid and	
CAST.aid = ACTOR.aid and ACTOR.name = "H.Ford"	0.95
MOVIE.year > 2000 and MOVIE.mid = GENRE.mid and	
MOVIE.mid = GENRE.mid and GENRE.genre = "animation"	0.95

MOVIE.year > 1990,	0.8
GENRE.genre = "comedy",	0.7
GENRE.genre = "adventure",	0.85
GENRE.genre = "drama",	0.2
GENRE.genre = "thriller",	0.9
GENRE.genre = "mystery",	0.9
GENRE.genre = "animation",	0.6
DIRECTOR.name = "W. Allen",	0.6
DIRECTOR.name = "S. Spielberg",	0.8
PRODUCER.name = "J. Bruckheimer",	0.6
PRODUCER.name = "S. Spielberg",	0.75
ACTOR.name = "K. Spacey",	0.8
ACTOR.name = "G. Clooney",	0.75
ACTOR.name = "M. Freeman",	0.8
ACTOR.name = "H. Ford",	0.9
ACTOR.name = "A. Sandler",	0.8
ACTOR.name = "B. Stiller",	0.7
MOVIE.mid=GENRE.mid,	0.8
MOVIE.mid = DIRECTED.mid and DIRECTED.did = DIRECTOR.did,	0.9
MOVIE.mid = CAST.mid,	1
MOVIE.mid = CAST.mid and CAST.aid=ACTOR.aid,	1
MOVIE.mid = GENRE.mid and GENRE.genre = "TV series" and MOVIE.mid = CAST.mid and CAST.aid = ACTOR.aid and ACTOR.name = "B. Stiller",	0.1
MOVIE.year > 2000 and MOVIE.mid = GENRE.mid and GENRE.genre = "animation",	0.95
MOVIE.mid = GENRE1.mid and GENRE1.genre = "animation" and MOVIE.mid = GENRE.mid and GENRE.genre = "thriller",	0.1
MOVIE.year > 1995 and MOVIE.mid = DIRECTED.mid and DIRECTED.did = DIRECTOR.did and DIRECTOR.name = "W. Allen",	0.8
CAST.aid=ACTOR.aid and ACTOR.name = "O. Wilson" and CAST.aid=ACTOR1.aid and ACTOR1.name = "B. Stiller" ,	0.4
CAST.aid=ACTOR.aid and ACTOR.name = "G. Clooney" and CAST.aid=ACTOR1.aid and ACTOR1.name = "B. Pitt",	1
CAST.aid=ACTOR.aid and ACTOR.name = "G. Clooney" and CAST.aid=ACTOR1.aid and ACTOR1.name = "E. McGregor" and CAST.aid=ACTOR2.aid and ACTOR2.name = "K. Spacey",	1
MOVIE.mid=GENRE.mid and GENRE.genre = "comedy" and MOVIE.mid = DIRECTED.mid and DIRECTED.did = DIRECTOR.did and DIRECTOR.name = "G. Clooney",	0.6
MOVIE.mid=GENRE.mid and GENRE.genre = "drama" and MOVIE.mid = DIRECTED.mid and DIRECTED.did = DIRECTOR.did and DIRECTOR.name = "G. Clooney",	0.85

Fig. 19. Part of a real user profile with movie preferences.

It is interesting to observe that the user had preferences for specific actors but she had given very high preferences for groups of actors participating in the same movie as a wish-list. While combining her individual preferences would not lead to these strong preferences, being able to code such fine-grained taste made possible to discover a new movie with title "The Men Who Stare at Goats" (a comedy to be released in 2010) that satisfies her wish that if her some of her favorite actors ever acted together this would make a not-to-miss movie.

Another search performed by the user was for short movies (duration < 1h45min) that would satisfy at least 2 preferences. The network built for this query is quite large and Figure 20 shows part of it. Short movies satisfying this user's preferences were some animation movies. Based on the network shown in the figure, we were able to rank high animation movies, such as "Ice Age" but give very low scores to animations, such as "Corpse Bride". We were not able to compute such accurate rankings using the simpler profile for this user because the complex preferences for animations are not derivable from the general preferences for animations and thrillers. If we combine these preferences then

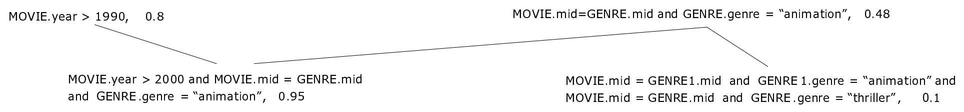

MOVIE.year > 1990, 0.8
MOVIE.mid=GENRE.mid and GENRE.genre = "animation", 0.48
MOVIE.year > 2000 and MOVIE.mid = GENRE.mid and GENRE.genre = "animation", 0.95
MOVIE.mid = GENRE1.mid and GENRE1.genre = "animation" and MOVIE.mid = GENRE.mid and GENRE.genre = "thriller", 0.1

Fig. 20. Part of a network of preferences related to a particular search.

"Corpse Bride" would rank high.

7.2 Performance Study

A user may explicitly give only a handful of preferences and implicit sources of preference elicitation, such as log mining and relevance feedback, may indicate many (additional) preferences. In both cases, incomplete information may prevent deriving a small set of generic preferences and a large number of preferences may be recorded that possibly contain many relationships. Handling multi-granular preferences and preference relationships add up to increased complexity and, thus, higher execution times. Therefore, in this section, we evaluate the performance of query personalization using our preference model (specific details are given per experiment.) We considered query loads of 50 random queries representing hypothetic user queries, each one containing one relation and one selection on this relation, that we want to personalize and we divided the query personalization process in two main phases: a preference construction phase, which builds a network of related preferences for a query and a preference query answering phase, which generates personalized results using this network. We study each phase separately and identify the critical factors that affect their performance before discussing the overall performance of personalization. Two parameters are important:

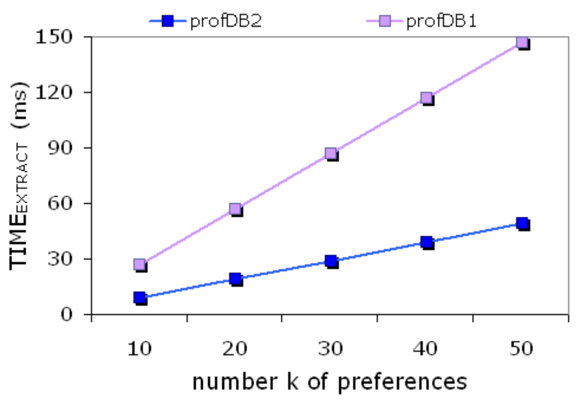
- the number k of selection preferences manipulated
- the number r of $\sqsubseteq$-relationships existing among them

We test our algorithms and we compare them with simpler algorithms that operate on the assumption of independent preferences. We generate synthetic profiles and networks depending on the requirements of each experiment, as we explain in the following subsections, and we build the indexes required for the processing. Times are shown in ms.

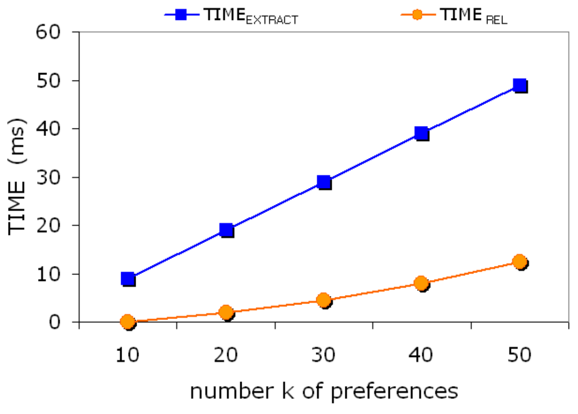
7.2.1 *Preference Construction.* This phase involves two tasks: extraction of the related preferences from a profile and construction of their network. These tasks are interleaved during preference construction (Section 5). We measure:

- the total time required to extract k selection preferences from a user profile - $\text{TIME}_{\text{EXTRACT}}$
- the total time required to find the relationships among k preferences and build their network, i.e., the total time required by the *Preference-Network Integration* and *Relationship Finder* modules - TIME_{REL}

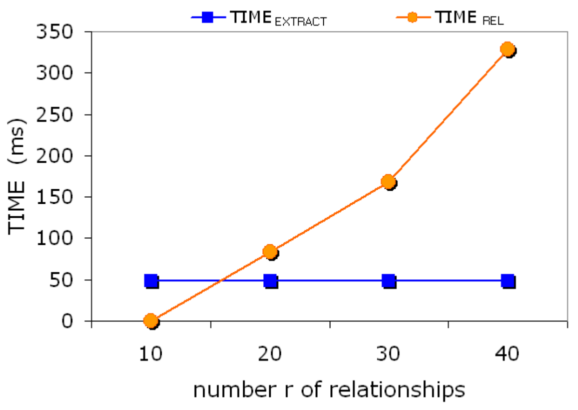
The time TIME_{REL} for processing independent preferences, as well as the extra time in $\text{TIME}_{\text{EXTRACT}}$ for preparing the structures required in preference comparisons, such as representing preferences as sets of path strings, are negligible. This observation allows us to consider that when having only independent preferences, the required execution time for this phase is equal to the time $\text{TIME}_{\text{EXTRACT}}$, and hence, the overhead from the processing of composite preferences is reflected in TIME_{REL}.



(a) Extraction for different databases



(b) Extraction & network integration vs. k



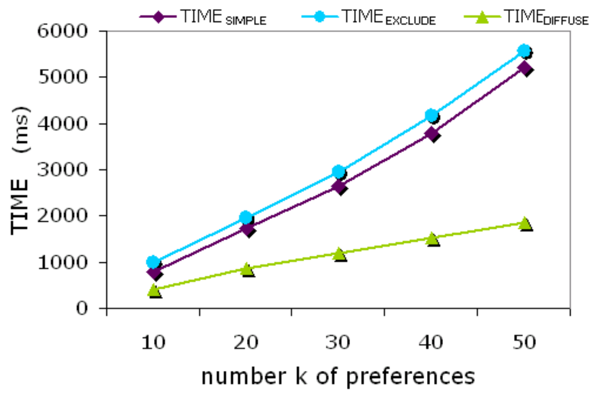
(c) Extraction & network integration vs. r

Fig. 21. Times for extracting and integrating k preferences with r preference relationships.

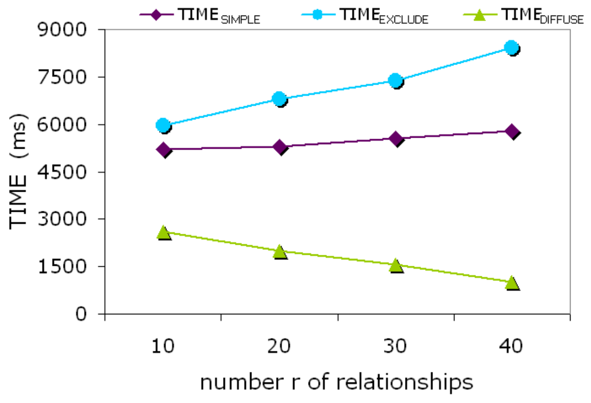
$\text{TIME}_{\text{EXTRACT}}$ depends on the number k of preferences handled, as Figure 21(a) confirms. The task of extracting related preferences does not simply read preferences stored in a profile but it also builds new preferences from the stored ones. Depending on the database schema and the number of selection preferences defined per relation, it may be able to retrieve k stored selection preferences with a few database accesses or it may need to search in relations further away from the initial query relations and compose preferences out of many stored ones. In order to gain insights into the impact of this phenomenon over $\text{TIME}_{\text{EXTRACT}}$, we generated the schema of a hypothetic database comprised of 100 relations, each one having 3 attributes, one of which possibly joining this relation to another. Then, we generated two synthetic profile databases of 100 profiles each. Profiles in both databases contained 100 selection preferences each but were generated in a different way: a profile in PROFDB1 was generated with the constraint that each attribute of the database could be used in at most one preference, while a profile in PROFDB2 was generated with the constraint that each attribute of the database should be used in at most three selection preferences. Consequently, profiles in PROFDB1 were sparser than profiles in PROFDB2. We considered a query load of 50 queries over this synthetic database and we measured the times for extracting the top k preferences for each of these queries. Figure 21(a) shows $\text{TIME}_{\text{EXTRACT}}$ as a function of k over all (100) profiles of each database for the 50 queries. The difference in execution times can be interpreted as the algorithm's effort to collect k preferences depending on how preferences are distributed over the database. When preferences are sparsely placed (e.g., in PROFDB1), it takes substantially more effort as k increases because it needs to search more in a profile.

Figures 21(b) and 21(c) show the execution times for finding k preferences ($\text{TIME}_{\text{EXTRACT}}$) and constructing a network of k preferences having r relationships (TIME_{REL}). For the former, we considered the profiles in PROFDB1 and the query load of 50 queries used in the previous experiment. Measuring the latter, especially with respect to r, is tricky. We wanted to explicitly control the number of relationships r found between k preferences in order to measure their impact. For this reason, we ignored the actual relationships among the k preferences found from the extraction step and we built a synthetic network generator, which takes as inputs the number k of preferences and the number r of preference relationships, and generates a network with these characteristics, i.e., containing r $\sqsubseteq$-relationships between k preferences, generated in a random way but w.r.t. certain constraints, e.g., defining at most one relationship per pair of preferences. The set of preferences and relationships of a synthetic network are fed into the *Relationship Finder*, which can guide *Preference-Network Integration* to build the same network from scratch. The latter is presented with the set of preferences of the synthetic network in random order and asks the *Relationship Finder* questions regarding $\sqsubset$-relationships between pairs of preferences in order to build the network.

Figure 21(b) shows the execution times as a function of k assuming r=5. Each point in the figure is the average of execution times for 100 networks with the same characteristics. We observe that $\text{TIME}_{\text{EXTRACT}}$ dominates and that is due to the number of database accesses required for the extraction of the top-k preferences (which increases with k), whereas the construction of the network is an in-memory process. In addition, we observe that although the latter phase depends strongly on k, the overhead of comparing more preferences as k increases is not as large as one might expect. The reason is that although we increase k, there are always r=5 preference relationships to be detected. That means that each



(a) Query answering vs. k



(b) Query answering vs. r

Fig. 22. Times for query answering using k preferences organized in a network with r relationships.

time a pair of preferences is compared, they are most likely independent and hence the comparison finishes instantly.

Figure 21(c) shows the execution times as a function of r assuming k=50. $\text{TIME}_{\text{EXTRACT}}$ is constant with r. We observe that as the preference network gets more complicated (i.e., with r increasing), TIME_{REL} deteriorates and exceeds $\text{TIME}_{\text{EXTRACT}}$. As more preference relationships exist, more complicated comparisons between preferences take place.

Consequently, each task's contribution to the total execution time for preference construction is different: one task may dominate the other depending on the parameter settings. Ultimately, the overhead from constructing a network for a query may comes not from the number of preferences but from the number of relationships among them.

7.2.2 *Preference Query Answering.* For this phase, we want to evaluate the performance of the two algorithms proposed and the overhead occurred due to the preference networks. Hence, we measure:

- the execution time required by the EXCLUDE_COMBINE algorithm - $\text{TIME}_{\text{EXCLUDE}}$

- the execution time required by the REPLICATE_DIFFUSE algorithm - $\text{TIME}_{\text{DIFFUSE}}$
- the execution time required by a NAÏVE algorithm that works with independent preferences - $\text{TIME}_{\text{SIMPLE}}$

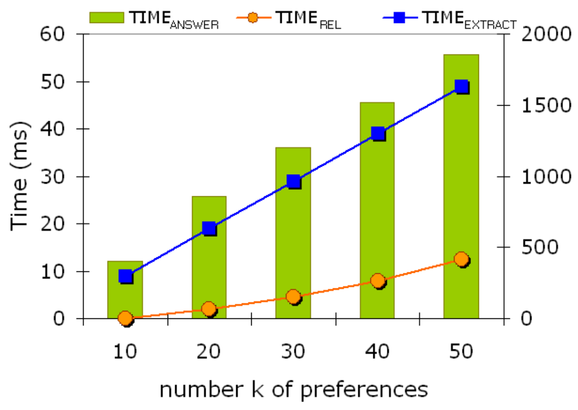
We built NAÏVE as a simpler version of algortihm EXCLUDE_COMBINE by omitting the exclude step of the latter. NAÏVE executes the set of queries that integrate one preference in the initial query, and combines their results, w.r.t. the constraint that at least l preferences are satisfied, treating the preferences as independent. For this series of experiments, we used the movie database. We generated a set of 50 random queries representing hypothetic user queries, each one containing one relation and one selection on this relation. Note that experiments with other sets of queries with different features, e.g., with one join and one selection, show similar trends and are not discussed. We generated different sets of preference networks over the movie database, each set containing 50 networks with k of preferences and r edges. Each network was meant to be combined with one query from the above set, and was generated as follows: we first composed k independent selection preferences for our movie database related to the query, with the constraint that they contained only one selection (but any joins required.) Then, we ran an iterative procedure that randomly picked r pairs of preferences and combined them to complex preferences. In essence, at each round, in a pair (p_i, p_j), p_j was replaced by $p_i \wedge p_j$ to form a more specific preference than p_i.

Figure 22(a) shows execution times as a function of k with r=5. Each point in the figure is the average of execution times for the 50 queries combined with their respective networks for the same k and r values. EXCLUDE_COMBINE requires more time than the NAÏVE approach. Since they are both built on the same philosophy, the overhead observed is due to the additional actions required to sort out results w.r.t. preference relationships. Algorithm REPLICATE_DIFFUSE shows the best performance. It executes as few queries as possible and process a smaller number of tuples than EXCLUDE_COMBINE close to the final number of results returned. These queries depend on the number of root preferences in the network. As k increases, more root preferences emerge but they are fewer than k.

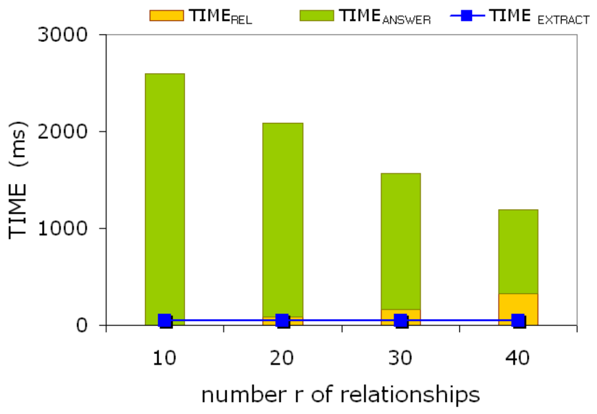
Figure 22(b) shows times as a function of r with k=50. While $\text{TIME}_{\text{EXCLUDE}}$ deteriorates when more preference relationships need to be resolved, $\text{TIME}_{\text{DIFFUSE}}$ is actually benefited because the total number of queries executed is lower. That is fewer queries of the type $Q \wedge p_{rt}$ may be executed (as r increases, fewer root preferences may exist) or fewer parameterized queries Q_i (tuples may be distributed at different nodes in the preference network). Hence, REPLICATE_DIFFUSE for generating results based on preference networks exhibits a better behavior in contrast to a naïve approach that works with independent preferences because it exploits preference relationships.

Overall, our results so far indicate that allowing complex preferences in query personalization may make preference construction more expensive but benefit query answering. How these phenomena shape the total execution time?

7.2.3 *Overall Performance.* To complete the picture regarding the efficiency tradeoffs, we compare the contribution of all parts in the total processing time of a query. We consider a query load of 50 queries over the movie database (each one containing one relation and one selection on this relation) and a set of 50 profiles and we plot the time $\text{TIME}_{\text{EXTRACT}}$ required to extract k selection preferences from a profile related to a query, the time TIME_{REL} required to find the relationships among these preferences and build the network and the time $\text{TIME}_{\text{ANSWER}}$ to generate the personalized results. For the latter, we



(a) Overall performance vs. k



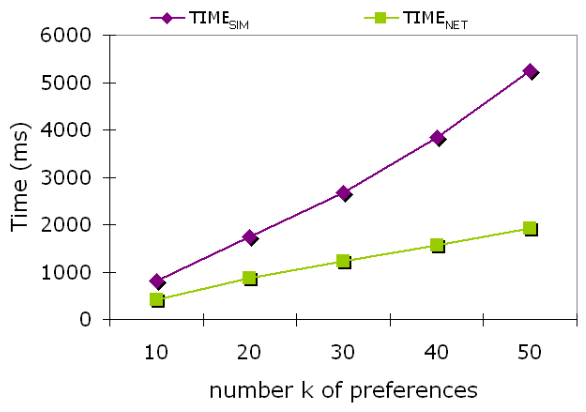
(b) Overall performance vs. r

Fig. 23. Overall times for personalizing queries using k preferences organized in a network with r relationships.

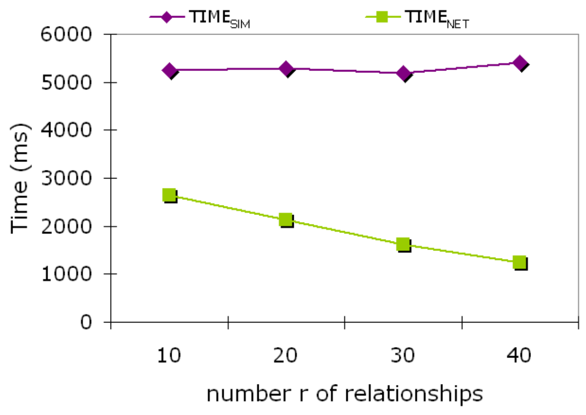
consider the time $\text{TIME}_{\text{DIFFUSE}}$, since REPLICATE_DIFFUSE has been shown to be the most efficient.

Figure 23(a) shows the execution times as a function of k for r=5 and Figure 23(b) shows the execution times as a function of r for k=50. Each point in the graphs is the average of execution times over the sets of queries and profiles for this experiment. When k increases the execution time of the preference query answering phase ($\text{TIME}_{\text{ANSWER}}$) shapes the overall performance. This time is measured on the right y-axis in Figure 23(a). On the other hand, when r increases (Figure 23(b)), we witness the effect of two opposite forces: decreasing time for generating results tends to "compensate" for the increase in the time required for building the network. Thus, the whole personalization process behaves in a more balanced way.

Finally, Figures 24(a) and 24(b) compare the efficiency of personalization using networks (TIME_{NET}) with personalization using only simple, independent preferences (TIME_{SIM}). Overall, performance is not sacrificed for expressiveness because we can use algorithms



(a) Impact of expressiveness on performance vs. k



(b) Impact of expressiveness on performance vs. r

Fig. 24. Impact of expressiveness.

that can benefit from the existence of preference relationships to adapt more smoothly to changes to k or r.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have studied query personalization based on multi-granular preferences. Existing approaches in preference modeling for query personalization support only preferences that can independently hold of each other and hence they can be freely combined for result ranking. Hence, a critical departure from these works is that we have lifted this independence assumption. Our framework allows the formulation of multi-granular preferences. We have introduced the concept of a preference network to capture preferences and their relationships, where each node refers to a subclass of the entities that its parent refers to, and whenever they both apply, more specific preferences override more generic ones. We have described algorithms for identifying preference relationships, constructing a network of related preferences for a query, and using this network to generate personalized answers. We have evaluated our framework and the algorithms by comparing them with simpler models and algorithms, which assume preference independence, both for efficiency and answer accuracy and we show that personalized answers that more accurately capture user preferences are feasible without losing in efficiency.

Our approach is applicable to any graph model representing information at the level of entities and relationships. User preferences may be articulated over a higher level graph model representing the data over 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 database restructuring and normalization. Preferences expressed at a higher level can be transparently mapped to the underlying schema for executing queries. Schema mappings have been studied in the literature, examining them in the context of query personalization seems an interesting direction. Furthermore, a number of interesting issues arise regarding the efficient management of complex preferences. For instance, how to exploit preference commonalities and access patterns in the same or different profiles for developing efficient storage and access methods and structures for complex preferences or for materializing parts of preference networks that are shared among multiple profiles or that are frequently applied for query personalization.

For the long run, a more "holistic" query optimizer that could take both a user query and a structured preference set into account for optimization is an intriguing direction to pursue. There are several possibilities to explore towards this end. For example, one could implement a special operator that ranks an input relation with respect to a preference relation in the spirit of [Koutrika et al. 2008]. Alternatively, one could imagine dynamic optimization techniques building on the concept of Eddies [Avnur and Hellerstein 2000]: the preferences act as filter operators and the Eddy routes tuples to them choosing adaptively the way each tuple is routed. A tuple is sent to the output when it has been handled by at least l operators.

Designing intuitive GUI's that facilitate defining and editing preferences is an open issue. In the same direction, designing interfaces allow the user to customize the extent of query personalization and get explanations regarding the effect of query personalization is also challenging. Finally, having the freedom to capture multi-granular preferences, we can also tune user profiling methods, such as user log mining, to store richer preferences in profiles instead of being forced to construct simpler profiles.

REFERENCES

A. Aho, Y. S. and Ullman, J. D. 1979. Equivalence of relational expressions. *SIAM J. of Computing 8,* 2, 218–246.

Agrawal, R., Rantzau, R., and Terzi, E. 2006. Context-sensitive ranking. In *SIGMOD*. 383–394.

Agrawal, R. and Wimmers, E. 2000. A framework for expressing and combining preferences. In *SIGMOD*.

Avnur, R. and Hellerstein, J. M. 2000. Eddies: Continuously adaptive query processing. In *SIGMOD*.

Balabanovic, M. and Shoham, Y. 1997. Fab: Content-based, collaborative recommendation. *CACM 40,* 3, 66–72.

Balke, W.-T., Guntzer, U., and Lofi, C. 2007. User interaction support for incremental refinement of preference-based queries. In *IEEE RCIS*. 511–523.

Borzsonyi, S., Kossmann, D., and Stocker, K. 2001. The skyline operator. In *ICDE*. 421–430.

Bruno, N., Chaudhuri, S., and Gravano, L. 2002. Top- k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM TODS 27,* 2, 153–187.

Chang, K. and Hwang, S. 2002. Minimal probing: Supporting expensive predicates for top-k queries. In *SIGMOD*.

CHEKURI, C. AND RAJARAMAN, A. 1997. Conjunctive quer containment revisited. In *ICDT*.

CHOMICKI, J. 2003. Preference formulas in relational queries. *ACM TODS 28,* 4, 427–466.

CHOMICKI, J. 2004. Semantic optimization of preference queries. In *Int'l Sym. on Applications of Constraint Databases*. 133–148.

COHEN, W. W., SCHAPIRE, R. E., AND SINGER, Y. 1998. Efficiently mining frequent trees in a forest. *Advances in Neural Information Processing Systems 10*.

CUPPENS, D. AND DEMOLOMBE, R. 1989. How to recognize interesting topics to provide cooperative answers. *Information Systems 14,* 2, 163–173.

DAS, A., DATAR, M., GARG, A., AND RAJARAM, S. 2007. Google news personalization: scalable online collaborative filtering. In *WWW*. 271–280.

FAGIN, R., KUMAR, R., AND SIVAKUMAR, D. 2003. Comparing top k lists. In *SIAM*. 28 – 36.

FISHBURN, P. 1999. Preference structures and their numerical representations. *Theor. Comput. Sci.* 217, 359–383.

GAASTERLAND, T., GODFREY, P., AND MINKER, J. 1992. An overview of cooperative query answering. *Journal of Intelligent Information systems 1,* 2, 123–157.

HANSSON, S. O. 2001. Preference logic.preference logic. *Handbook of Philosophical Logic (D. Gabbay, Ed.) 8*.

HOLLAND, S., ESTER, M., AND KIESSLING, W. 2003. Preference mining: A novel approach on mining user preferences for personalized applications. In *PKDD*. 204–216.

HOLLAND, S. AND KIESSLING, W. 2004. Situated preferences and preference repositories for personalized database applications. In *ER*. 511–523.

HRISTIDIS, V., KOUDAS, N., AND PAPAKONSTANTINOU, Y. 2001. PREFER: A system for the efficient execution of multiparametric ranked queries. In *SIGMOD*.

ILYAS, I., SHAH, R., AREF, W., VITTER, J., AND ELMAGARMID, A. 2004. Rank-aware query optimization. In *SIGMOD*.

JIANG, B., PEI, J., LIN, X., CHEUNG, D. W., AND HAN, J. 2008. Mining preferences from superior and inferior examples. In *KDD*. 390–398.

JOACHIMS, T. 2002. Optimizing search engines using clickthrough data. In *KDD*.

JOACHIMS, T., FREITAG, D., AND MITCHELL, T. 1997. Webwatcher: a tour guide for the world wide web. In *IJCAI*.

KENDALL, M. AND GIBBONS, J. D. 1990. *Rank Correlation Methods*. Edward Arnold, London.

KIESSLING, W. 2002. Foundations of preferences in database systems. In *VLDB*. 311–322.

KIESSLING, W. AND KOSTLER, W. 2002. Preference SQL-design, implementation, experiences. In *VLDB*.

KOSSMANN, D., RAMSAK, F., AND ROST, S. 2002. Shooting stars in the sky: An online algorithm for skyline queries. In *Int'l VLDB Conf.* 275 – 286.

KOUTRIKA, G. 2006. Personalization of structured queries with personal and collaborative preferences. In *ECAI Workshop about Advances on Preference Handling*.

KOUTRIKA, G., BERCOVITZ, B., AND GARCIA-MOLINA, H. 2008. FlexRecs: Expressing and combining flexible recommendations. In *SIGMOD*.

KOUTRIKA, G. AND IOANNIDIS, Y. 2004. Personalization of queries in database systems. In *ICDE*. 597–608.

KOUTRIKA, G. AND IOANNIDIS, Y. 2005a. Constrained optimalities in query personalization. In *SIGMOD*. 73–84.

KOUTRIKA, G. AND IOANNIDIS, Y. 2005b. Personalized queries under a generalized preference model. In *ICDE*.

LACROIX, M. AND LAVENCY, P. 1987. Preferences: Putting more knowledge into queries. In *VLDB*. 217–225.

LEE, J., WON YOU, G., WON HWANG, S., SELK, J., AND BALKE, W.-T. 2008. Optimal preference elicitation for skyline queries over categorical domains. In *DEXA*. 610–624.

LINDEN, G., SMITH, B., AND YORK, J. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*.

LIU, F., YU, C., AND MENG, W. 2004. Personalized web search for improving retrieval effectiveness. *IEEE TKDE 16,* 1.

MIAH, M., DAS, G., HRISTIDIS, V., AND MANNILA, H. 2008. Standing out in a crowd: Selecting attributes for maximum visibility. In *ICDE*. 356–365.

PAPADIAS, D., TAO, Y., FU, G., AND SEEGER, B. 2003. An optimal and progressive algorithm for skyline queries. In *ACM Int'l Conf. on Management of Data*. 467–478.

PEI, J., JIANG, B., LIN, X., AND YUAN, Y. 2007. Probabilistic skylines on uncertain data. In *VLDB*. 15–26.

PEI, J., JIN, W., ESTER, M., AND TAO, Y. 2005. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *VLDB*. 253–264.

PITKOW, J. E., SCHÜTZE, H., CASS, T. A., COOLEY, R., TURNBULL, D., EDMONDS, A., ADAR, E., AND BREUEL, T. M. 2002. Personalized search. *Comm. of the ACM 45,* 9, 50–55.

SARKAS, N., DAS, G., KOUDAS, N., AND TUNG, A. K. H. 2008. Categorical skylines for streaming data. In *SIGMOD*. 239–250.

SATZGER, B., ENDRES, M., AND KIESSLING, W. 2006. A preference-based recommendation system. In *ECWeb*.

STEFANIDIS, K. AND PITOURA, E. 2008. Fast contextual preference scoring of database tuples. In *EDBT*. 344–355.

STEFANIDIS, K., PITOURA, E., AND VASSILIADIS, P. 2007. Adding context to preferences. In *ICDE*.

TAO, Y., HRISTIDIS, V., PAPADIAS, D., AND PAPAKONSTANTINOU, Y. 2007. Branch-and-bound processing of ranked queries. *Inf. Syst. 32,* 3, 424–445.

VAN BUNNINGEN, A., FENG, L., AND APERS, P. M. G. 2006. A context-aware preference model for database querying in an ambient intelligent environment. In *DEXA*. 33–43.

VLACHOU, A., DOULKERIDIS, C., KOTIDIS, Y., AND VAZIRGIANNIS, M. 2007. SKYPEER: Efficient subspace skyline computation over distributed data. In *ICDE*. 416–425.

WELLMAN, M. AND DOYLE, J. 1991. Preferential semantics for goals. In *National Conf. on AI*. 698–703.

WONG, R. C.-W., PEI, J., FU, A. W.-C., AND WANG, K. 2007. Mining favorable facets. In *KDD*. 804–813.

XIN, D. AND HAN, J. 2008. P-cube: Answering preference queries in multi-dimensional space. In *ICDE*. 1092 – 1100.

YIU, M. L., DAI, X., MAMOULIS, N., AND VAITIS, M. 2007. Top-k spatial preference queries. In *ICDE*. 1076 – 1085.

ZAKI, M. J. 2005. Efficiently mining frequent trees in a forest. *Inf. Syst. 17,* 8, 1021 – 1035.