Querying with Preferences in a Digital Library*

Nicolas Spyratos[1] and Vassilis Christophides[2]

[1] Laboratoire de Recherche en Informatique
Université Paris Sud
91405 Orsay Cedex
spyratos@lri.fr
[2] Institute of Computer Science
Foundation for Research and Technology-Hellas
P.O. Box 1385, 71110 Heraklio, Greece
christop@ics.forth.gr

Abstract. We consider a collection of federated sources of the web, and a community of users that are interested in documents residing in one or more among the federated sources. The search for documents of interest is supported by a mediator that we call a digital library. The library simply indexes all documents that are made available to users by the federated sources. When a user addresses a query to the library, the library returns the URLs of documents satisfying the query. In such a context, one of the factors influencing user satisfaction is the size of the answer set when it is either too small (few or no documents), or too large (several hundreds or thousands of documents). In this paper, we address the problem of too large answer sets and we call *personalized query* a usual query together with (a) an upper bound on the number of documents returned and (b) a set of preferences as to the order in which the returned documents should be presented to the user; *both* these parameters are defined by the user *online*, during query formulation time. The main contribution of the paper is to propose a framework in which the problem can be stated formally, and a method for the evaluation of personalized queries.

1 Introduction

As information becomes available in increasing volumes, and to growing numbers of users, the shift towards a more user-centered access to information is becoming an important issue. As a consequence, support of personalized user interaction is an important concern in the design of advanced information systems, in general, and of e-services, in particular [3, 4, 9].

Personalization in an interactive information system is about building a meaningful one-to-one relationship by understanding the needs of each individual user. Personalization can involve either adapting the user interface or adapting the content to the needs or preferences of a specific user. Both aspects have received increasing attention during the past few years. In this paper we propose

* This work is partially supported by the EU Network of Excellence in Digital Libraries (Delos NoE-6038-507618)

to study a specific aspect of content adaptation, namely query personalization, in the context of a digital library.

We view a digital library as a mediator over a collection of federated sources that store electronic documents of various kinds. The sources make available (i.e. accessible) some or all of their documents to a community of users (the library subscribers). Users address their queries to the library, in search of documents of interest residing in one or more federated sources. In such a context, one of the factors that influences user satisfaction is the size of the answer set when it is either too small (few or no documents), or too large (several hundreds or thousands of documents).

In this paper, we address the problem of too large answer sets and we call *personalized query* a usual query q together with (a) an upper bound k on the number of documents returned and (b) a set of preferences as to the order in which the returned documents should be presented; *both* these parameters are defined by the user *online*, during query formulation time. In other words, the query q specifies (intentionally, as usual) a set of documents which is unordered, whereas the preferences and the bound k select an ordered subset of desired size. The main contribution of the paper is to propose a framework where the problem can be stated formally, and a method for the evaluation of personalized queries.

Regarding related work, first we note that personalization in electronic publishing addresses mainly three items:

- *Notification*: The user need for receiving timely and accurate information relevant to his interests.
- *Query personalization*: The user need to be adequately supported during search of archive information.
- *Recommendation*: The publishers need to proactively disseminate information only to interested users.

An essential feature of personalization techniques for information access is the capability to autonomously and automatically learn user interests and preferences from the observation of user's behavior, i.e., adaptability. This adaptability is based on various machine learning techniques and provides the means to unobtrusively build user profiles [5, 12]. However, profiles contain information describing *past* interests or preferences of the user. In this paper, we are interested in query personalization based on *current* interests and preferences of the user, expressed *online*, during query formulation time.

The possibility for the user to express preferences online, during query formulation, is an aspect of personalization that has received increasing attention during the past few years. The main reason for this increasing attention is that traditional systems require the user to specify the information to be retrieved *exactly*, allowing no provision for "customizing" the answer set. In the area of databases, two different approaches of query personalization are being pursued currently: qualitative [6, 7, 11, 13, 15, 17] and quantitative [1, 14, 16]. A fundamental study concerning general preference relations is presented in [2].

In the qualitative approach, the preferences between tuples in the answer to a query are specified directly, typically using so-called preference relations, i.e., binary relations over tuples, satisfying certain natural conditions such as transitivity and reflexivity. In the quantitative approach, preferences are specified indirectly using scoring functions that associate a numeric score with every tuple of the query answer; then a tuple t is preferred to a tuple t' if the score of t is higher than the score of t'.

The qualitative approach seems to be more general than the quantitative one, since one can define preference relations in terms of scoring functions, whereas not every (intuitively plausible) preference relation can be captured by scoring functions. The approach that we propose in this paper is related with the qualitative approach, and is in fact motivated by current research in that area. It also draws on ideas and results from [19] and [20].

A notable feature of the qualitative approach is the clear separation between the formal specification of preferences from their embedding in the query language, and this allows studying abstract properties of preferences separately from query evaluation (similarly to the separation between relational algebra and its embedding into the algebraic query language, in the relational database model).

The remaining of the paper is organized as follows. In section 2, we define the model of digital library that we use, and recall some basic definitions from [18]. In sections 3 and 4, we define formally a personalized query and its answer, respectively, and in section 5 we outline an algorithm for the evaluation of personalized queries. Finally, in section 6, we discuss ongoing research and offer some concluding remarks.

2 Digital Libraries

A *digital library* serves a network of federated sources that store electronic documents of various kinds. The sources make available (i.e. accessible) some or all of their documents to a community of users (the library subscribers). Users address their queries to the library, in search of documents of interest residing at the local repositories of one or more federated sources. The digital library acts as a mediator, supporting transparent access to all shareable documents by library users.

We emphasize the fact that the library stores no document at its local repository: the documents stay at the sources and the library merely acts as a mediator to access them. We note that, as the documents accessible through the library reside at the local repositories of the participating sources, the local repositories can be seen collectively as a single distributed repository of documents spread over the network [18].

In order to carry out its basic function (i.e. that of a mediator), the library is required to support two basic services, *registration* of documents that the sources are willing to make available to library users, and *querying* for documents of interest by the library users.

2.1 Registration

When a source wishes to make a document shareable (i.e. accessible) by library users it must register the document at the library. To do so the source must provide two items to the library:

- the document identifier (indispensable for document registration)
- the document description (optional)

We assume that the document identifier is just the URL where the document can be accessed by library users. Clearly, as the document identifier is the only means for accessing the document, no document can be registered unless its identifier is provided by the source to the library. This is why providing the document identifier is indispensable for document registration. Hereafter, in our examples, we use integers as document identifiers instead of URLs, for convenience of notation.

As for document description, we consider only content description and we assume that such a description is available in the form of a set D of keywords, or *terms*. For example, if D={QuickSort, Java} this means that the document is about the quicksort algorithm written in Java. We make no assumption as to the provenance of the terms in the description of the document. These terms may or may not be coming from a controlled vocabulary, and they may be given manually or extracted from the document content using some document mining algorithm. Moreover, some sources may simply provide the document identifier without providing any content description. It will then be the job of the library to provide such a description (manually or using some mining algorithm). Finally, it may be the case that the source does provide a content description but the library "augments" this description to provide a finer one for its users (see [18] for more details).

Therefore registration of a document at the library requires the document identifier, say i, and a set of terms D (defined as explained above). The actual registration of the document by the library is done by storing a pair <i, t> in the library repository, for each term t in D. Thus if i is the document identifier, and D={QuickSort, Java} is the document description, then the library will store two pairs: <i, QuickSort> and <i, Java>. The set of all such pairs <i, t>, for all registered documents, is what we call the library directory, or simply *directory* (the well known Open Directory of the Web is an example of such a directory). Clearly the directory is a binary relation between document identifiers and terms. As such, it can be seen also as a formal context [8].

Each term in the directory has an extension, and each document identifier has a description (also called its intension, or its index). The *extension* of a term t, denoted as *ext(t)*, is the set of all document identifiers i such that <i, t> is in the directory; and the *description* of a document i, denoted *descr(i),* is the set of all terms t such that <i, t> is in the directory. More formally, we have the following definitions:

ext(t) ::= {i : <i, t> is in the directory}

descr(i) ::= {t : <i, t> is in the directory}

We assume that, if a term does not appear in the directory then its extension is empty. Similarly, we assume that, if a document identifier does not appear in the directory then its description is empty.

We note that from a mathematical point of view, *ext* and *descr* are set-valued functions, and that they may change in time, as the directory is updated. Moreover, either of these functions is sufficient to define the directory, and vice versa, the directory is sufficient to define each of these functions.

2.2 Querying

Library users access the library in search of documents of interest, either to use them directly (e.g. as learning objects) or to reuse them as components in new documents that they intend to compose, see [18]. Search for documents of interest is done by issuing queries to the library, and the library uses its directory to return the identifiers (i.e. the URLs) of all documents satisfying the query.

The query language that we use is a simple language in which a query q is just a Boolean combination of terms:

$$q ::= t \mid q1 \wedge q2 \mid q1 \vee q2 \mid q1 \wedge \neg q2$$

The answer to a query q is defined recursively as follows:

```
if q is a term then ans(q):=ext(q)
else begin if q=q1 ∧ q2 then ans(q):=ans(q1)∩ans(q2);
           if q=q1∨q2 then ans(q):=ans(q1)∪ans(q2);
           if q=q1∧¬q2 then ans(q):=ans(q1)\ans(q2)
     end
```

In other words, to answer a query q, the library simply replaces each term appearing in q by its extension from the directory, and performs the set theoretic operations corresponding to the Boolean connectives.

3 Personalized Queries

In our approach, a personalized query Q consists of three parts:

- a usual query q, e.g. QuickSort∨Java,
- an integer k, expressing the maximum number of documents to be returned,
- a set of preferences, e.g. "I prefer documents about QuickSort to those about Java"

In this paper, we model the set of preferences as a *pre-order* over terms, i.e., as a binary relation over terms which is reflexive and transitive. These two properties of preferences seem to be reasonable in our context, although no strong argument can be presented in their favor, e.g. Fishburn considers non-transitive preferences in decision theory [10].

On the contrary, a strong argument can be made *against* including anti-symmetry as a property of preferences. Indeed, preferring QuickSort over Java *and* Java over QuickSort doesn't make QuickSort equal to Java but simply equivalent to Java (with respect to user preferences).

In fact, in our context, equivalence of terms is a highly desirable property, as it implies the existence of one or more *alternatives*. For example, assume that QuickSort is equivalent to Java (according to the preferences declared by the user), and that documents described by one or the other term are the most preferred by the user. If no document described by QuickSort is currently available in the directory, then the system can return any document described by Java as an alternative.

Therefore, we allow the user to declare two kinds of statements over terms, alternatives and preferences. Alternatives are statements of the form t≡t', meaning that the user is equally interested in documents described by either t or t', while preferences are statements of the form s≤s' meaning that the user prefers documents described by s (or any term equivalent to s) to documents described by s' (or any term equivalent to s'). In what follows we assume that alternatives form an equivalence relation over terms, i.e. a binary relation that is reflexive, symmetric and transitive. We note that synonyms are alternatives as well, and they are captured by the equivalence relation just defined, as long as they are declared, e.g. Databases≡DBs.

Now, if we consider the equivalence classes of terms formed by the alternatives, then the preference relation becomes a partial order over those equivalence classes. In what follows, in order to simplify the presentation, we shall consider the terms of the directory up to equivalence, i.e. when we talk of a term we shall actually mean the equivalence class of that term (with respect to the equivalences declared by the user).

Following our discussion so far, we can define a personalized query, formally, as follows:

Definition 1. *Personalized Query*
A personalized query over the library directory is a triple Q= (q, k, ≤), where q is a Boolean query, k is a positive integer, and ≤ is a partial order relation over terms, called the preference relation.

In the above definition, we note that the Boolean query q can be evaluated as described in the previous section, and that the evaluation returns a set of documents, call it Ans(q). Normally, i.e. in a traditional system, what is returned to the user is the set Ans(q). In our approach, however, as explained in the introduction, what is returned to the user is *not* necessarily the whole set Ans(q) *but* a subset thereof, of size at most k; and moreover, the subset returned to the user is ordered according to the preference relation. So, in the next section, we explain how one orders the set Ans(q), based on the preference relation, and how one selects the subset of Ans(q), of size k, to be returned to the user.

We also note that the terms appearing in the preference relation may not appear in the query q. This means that it is perfectly possible to ask for all

documents concerning quicksort and java, but preferring those concerning theory to those concerning applications.

4 Answering a Personalized Query

The first thing to note is that the preference relation is defined over terms, and not over documents. Therefore, we use the preference relation to define a pre-order $\preceq$ over documents as follows.

Definition 2. *Ordering Documents*
Let d, d' be two documents. We define $d \preceq d'$ *iff* $\forall t \in descr(d)\ \exists t' \in descr(d')\ t \leq t'$

It is easy to see that is reflexive and transitive but not anti-symmetric (as two documents can have the same description, while being different documents). Therefore we define two documents d, d' to be *equivalent*, denoted as $d \equiv d'$, iff $d \preceq d'$ and $d' \preceq d$. With this definition at hand, the preorder $\preceq$ becomes a partial order over equivalence classes of documents.

Proposition 1. *The relation* $\preceq$ *is a partial order over (equivalence classes of) documents.*

In the following we shall talk about documents up to equivalence, i.e. we shall not distinguish documents that are equivalent. With this partial order at hand, we now proceed to define the subset ans(Q) of the ordered set Ans(q), to be returned to the user. First, we introduce some auxiliary definitions and notation.

We call a document of Ans(q) *active* if it appears in some comparison under $\preceq$, and *inactive* otherwise. Roughly speaking, active documents are those that can be ordered with respect to other documents (as they participate in at least one comparison), while inactive documents are those that cannot be ordered. As a consequence, the set Ans(q) is partitioned into two sets, denoted Act(q) and Inact(q), the set of active documents and the set of inactive documents, respectively. Our approach to answering the personalized query Q uses only the set of active documents; so hereafter whenever we say "document" we mean "active document". In a nutshell, our approach can be described as follows:

1. Define a partition of the set Act(q) into blocks, and a linear ordering of the blocks, satisfying the following properties:
   (a) Each block consists of non comparable documents
   (b) The first block contains the most preferred documents
   (c) For each block B other than the first, and for each document in B, there is a more preferred document in the previous block
2. Start presenting each block of documents to the user, one by one, in their linear order
3. If the number of all documents in Ans(q) is less than, or equal to k then show all documents else terminate the presentation of documents after showing the k-th document (this can happen before all documents of a block have been shown to the user)

To define our approach formally we need some auxiliary definitions and notation that we adapt from. First, let us call *path* from a document d to a document d′, any sequence of pairs of the form

$<d, d_1>, <d_1, d_2>, \ldots, <d_{n-1}, d_n>, <d_n, d'>$

such that $d \preceq d_1$, $d_n \preceq d'$, and $d_{i-1} \preceq d_i$, for i = 2, ..., n. The integer n+1 is called the length of the path, and it should be clear that there may be zero, one or more paths from d to d′.

Roughly speaking, the partition of the set Act(q) that we define next uses the set B_0 of all documents that are minimal with respect to $\preceq$ as the basic block. The definition of each other block B_i of the partition relies on the notion of distance of a document from B_0. The *distance* of a document d′ from B_0 is defined to be the length of a longest path from a document d to the document d′, when d ranges over all documents of B_0; and if d′ belongs to B_0 then this distance is defined to be equal to 0. The block B_i is defined to be the set of all documents that are at distance i from B_0. The following theorem (which is our main theorem) states formally the definition of the partition, as well as its basic properties.

Theorem 1. *Partitioning the Set Act(q)*
Let B_0 be the set of all documents that are minimal with respect to $\preceq$, and let m be the length of a longest path from B_0 to a maximal document. For i=1, ..., m, define the set B_i as follows: d′ is in B_i iff d′ is at distance i from B_0. Then the following statements hold:
(a) The collection $\{B_0, B_1, \ldots, B_m\}$ is a partition of Act(q).
(b) In every block B_i no two documents are comparable, i=0, 1, . . . , m.
(c) For every document d′ in B_i there is a document d in B_{i-1} such that $d \preceq d'$, i=1, 2, . . . , m.

Proof.
(a) It suffices to observe that every document will belong to some B_i (because it appears in at least one comparison) and that no document can belong to two different B_i's (as its distance from B_0 is unique). Therefore, the B_i's are mutually disjoint and their union is Act(q), i.e. the collection of all B_i's is a partition of Act(q).
(b) Suppose now that there are documents d, d′ in B_i such that $d \preceq d'$. As the distance of d from B_0 is i, it follows that the distance of d′ from B_0 is i+1, a contradiction.
(c) For every document d′ in B_i there is a longest path p from some document of B_0 to d′. Let d be the predecessor of d′ in p. Clearly, the sub-path of p ending in d is a longest path from B_0 to d (otherwise, p is not a longest path to d′, a contradiction). It follows that d is in B_{i-1} and that $d \preceq d'$.

We now define a partial order over sets of documents and show that the sequence B_0, B_1, ..., B_m is actually a chain in that partial order. This is one of the possible orders one can define over sets of documents, which are induced by the partial order $\preceq$ over documents (see [18, 20] for more details).

Definition 3. *Ordering Sets of Documents*
Let D, D' be two sets of documents. We define $D \sqsubseteq D'$ *iff* $\forall d' \in D'\ \exists d \in D$ *such that* $d \preceq d'$

Clearly, the relation $\sqsubseteq$ is reflexive and transitive but not anti-symmetric; hence it is a pre-order over sets of documents. However, it is not difficult to see that if each set of documents consists of mutually incomparable documents, then $\sqsubseteq$ becomes anti-symmetric, and thus a partial order, see also [20]. Now, each block B_i of the partition defined earlier, does have this property (see Theorem 1(b)), and thus $\sqsubseteq$ is a partial order over the partition $\{B_0, B_1, ..., B_m\}$. Moreover, in view of Theorem 1(c), we have $B_0 \sqsubseteq B_1 \sqsubseteq ... \sqsubseteq B_m$, i.e. the sequence B_0, B_1, ..., B_m is a chain in $\sqsubseteq$; hence the following proposition.

Proposition 2. *Ordering the Partition of Act(q)*
The relation $\sqsubseteq$ *is a partial order over sets of non comparable documents, and the sequence* B_0, B_1, ..., B_m *is a chain in* $\sqsubseteq$, *i.e.* $B_0 \sqsubseteq B_1 \sqsubseteq ... \sqsubseteq B_m$

Thus the sequence $B_0, B_1, ..., B_m$ is actually the right sequence for presenting documents to the user, in the sense that each document in B_i is less preferred than some document of B_{i-1}. Of course, there may be documents in B_{i-1} that have no less preferred documents in B_i. However, this doesn't go against the intuition "most preferred documents first"!

In the following section we present an algorithm for evaluating the answer to a personalized query Q.

5 The Evaluation Algorithm

Our algorithm constructs the blocks of the sequence $B_0, B_1, ..., B_m$, one by one, while keeping track of the number of documents produced. If the size of Act(q) is less than, or equal to k then all blocks are constructed. Otherwise computation should normally terminate when the k-th document is produced -something that could happen before all documents of the k-th document's block have been produced. However, in the following definition of answer to Q, in order to simplify matters, we include in the answer *all* documents of the block containing the k-th document. We use the notation card(X) to denote the cardinality of a set X of documents.

Definition 4. *The Answer to a Personalized Query*
Let $Q = (q, k, \leq)$ *be a personalized query. The answer to Q is the sequence of blocks* B_0, B_1, ..., B_j, $0 \leq j \leq k$, *defined as follows:*
if $card(Act(q)) \leq k$ *then the answer is* B_0, B_1, ..., B_m
else the answer is the sequence B_0, B_1, ..., B_j *such that*
$card(B_0) + ... + card(B_{j-1}) \leq k \leq card(B_0) + ... + card(B_{j-1}) + card(B_j)$

Our algorithm is best understood if we view the pair G= (Act(q), $\preceq$) as a graph, defined as follows:

– the documents in Act(q) are the nodes of G
– there is an arrow from document d to document d′ iff d$\preceq$d′

We note that, as $\preceq$ is a partial order, this graph is acyclic.

To keep the presentation of the algorithm as simple as possible, we assume the following:

1. There is a function that takes as input the personalized query Q= (q, k, $\leq$), computes successively ans(q), Act(q), G=(Act(q), $\preceq$) and the length m of a longest path from the set of minimal documents to a maximal document. In the presentation of the algorithm we assume that the set Act(q) is not empty, i.e. the graph G has at least one edge.
2. There is a function card that takes as input a set of documents B= $\{d_1, d_2, \ldots, d_j\}$ and computes the number of documents in B, as follows:

$$\text{card(B)= card}(d_1) + \text{card}(d_2) + \ldots + \text{card}(d_j)$$

We recall that documents are considered up to equivalence, hence card(d) returns the number of documents in the equivalence class of d.

```
Algorithm PERSO
Input : A personalized query Q= (q, k, ≤)
Output: The answer to Q
Method
count:=0; i:=0
repeat Bi:={all roots of G}; output Bi;
       count:= count + card(Bi);
       remove from G all nodes of Bi and all edges
       emanating from nodes of Bi;
       i:=i+1
until G = ∅ or k≤count;
```

The correctness of this algorithm follows rather easily from Theorem 1, and our preceding discussions. Complexity and performance issues, however, lie outside the scope of the present paper, and will be discussed in the full paper.

6 Concluding Remarks

We have seen an approach to defining and evaluating personalized queries in the context of digital libraries. A personalized query is defined as a usual Boolean query together with an upper bound on the number of documents returned and a set of preferences and alternatives as to the order in which the returned documents should be presented; both these parameters can be defined by the user *online*, during query formulation time. We have also seen an evaluation algorithm for answering personalized queries.

However, the research reported in this paper is preliminary, and much work remains to be done in support of the proposed approach, in particular with respect to the implementation of the evaluation algorithm and its performance. Once this is done, our plan is to use the query evaluation algorithm for implementing the following two-step interactive scheme for query personalization:

Step 1
The user submits a query q and the system returns an integer, which is the size of the answer set (or an estimate thereof)

Step 2
if the user is satisfied then the system returns the actual answer set
else if the user judges the size of the answer set too large then he/she can "personalize" the query (by declaring an upper bound k plus preferences and alternatives); the system then returns the desired ordered subset of the answer set.

Concerning the implementation of the algorithm, the choice of an appropriate representation for the library directory seems to be essential. The obvious candidates are the binary relation <*doc-id, term*> and the set-valued functions *ext* and *int* defined earlier (see section 2).

The influence of structure over the set of terms may also prove significant. Indeed, although no assumption of structure was made here over the set of terms, in many applications the terms come from a controlled vocabulary, possibly structured as taxonomy according to a subsumption relation (see [18]). Structuring the set of terms will have an influence on the evaluation of a personalized query Q= (q, k, $\leq$) at two levels. First, on the evaluation of the Boolean query q, as the extension of a term will now include the extensions of all its sub-terms; and second, on the preferences, as we may or may not have the ordering of two terms to imply a "similar" ordering on the sets of their sub-terms.

Concerning performance, a detailed complexity analysis of the evaluation algorithm is necessary. This analysis should take into consideration several factors, including the size of the set of preferences (i.e. the number of terms appearing in preferences), the size of the query q (i.e. the number of terms appearing in q), the bound k, the size of the set ans(q), and the number of active documents.

A final remark concerns the inactive documents. Indeed, the answer to a personalized query as defined here is based solely on the active documents. However, when the set of active documents turns out to be relatively small (with respect to the bound k) then one may wish to include some inactive documents in the answer as well. In this respect, inactive documents whose description contains at least one term appearing in the preference relation should naturally be preferred to the remaining inactive documents (i.e. those whose description contains no term appearing in the preference relation).

Summarizing, further research and extensive experimentation is needed to evaluate the proposed approach and query-answering algorithm, thus validate the choices made here.

References

1. R. Agrawal and E.L. Wimmers. A Framework for Expressing and Combining Preferences. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data, Dallas, USA*, pages 297–306, 2000.
2. H. Andreka, M. Ryan, and P-Y. Schlobbens. Operators and Laws for Combining Preferential Relations. *Journal of Logic and Computation*, 12(1):13–53, 2002.
3. R. Brafman, J. Doyle, U. Junker, and P. Pu. Tutorial on Preference Models and Application, 2005. 19th International Joint Conference on Artificial Intelligence, IJCAI.
4. S. Braynov. Personalization and Customization Technologies, 2003. Seminar on Personalization and Customization in E-Commerce, Available at: http://www.cs.buffalo.edu/˜ sbraynov/seninar2003/papers/Personalization.pdf.
5. L. Chen and P. Pu. Survey on Preference Elicitation Methods, 2004. Ecole Politechnique, Federale de Lausanne (EPFL), Switzerland.
6. J. Chomicki. Querying with Intrinsic Preferences. In *Proceedings of the 8th International Conference on EDBT, Prague, Czech Rep.*, pages 34–51, 2002.
7. J. Chomicki. Preference formulas in relational queries. *ACM Transactions on Database Systems*, 28(4):427–466, 2003.
8. B.A. Davey and H.A. Priestly. *Introduction to Lattices and Order (2nd edition)*. Cambridge Mathematical Textbooks. Cambridge University Press, 2002.
9. J. Doyle and U. Junker. Tutorial on Preferences, 2004. 19th AAAI National Conference on Artificial Intelligence.
10. P.C. Fishburn. Non-transitive Preferences on Decision Theory. *Journal of Risk and Uncertainty*, 4:113–134, 1991.
11. K. Govindarajan, B. Jayaraman, and S. Mantha. Preference Queries in Deductive Databases. *New Generation Computing*, 19(1):57–86, 2000.
12. S. Holland, M. Ester, and W. Kießling. Preference Mining: A Novel Approach on Mining User Preferences for Personalized Applications. Technical report, Institute of Computer Science, University of Augsburg, Germany, 2003.
13. S. Holland and W. Kießling. Situated Preferences and Preference Repositories for Personalized Database Applications. In *Proceedings of the 23rd International Conference on Conceptual Modeling, ER, Shanghai, China*, pages 511–523, 2004.
14. V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A system for the efficient execution of multiparametric ranked queries. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data, Santa Barbara, California*, page 259269, 2001.
15. W. Kießling. Foundations of Preferences in Database Systems. In *Proceedings of the 28th International Conference on Very Large Databases, Hong Kong, China*, pages 311–322, 2002.
16. G. Koutrika and Y. Ioannidis. Personalization of Queries in Database Systems. In *Proceedings of the 20th International Conference on Data Engineering, Boston, USA*, pages 597–608, 2004.
17. M. Lacroix and P. Lavency. Preferences: Putting More Knowledge Into Queries. In *Proceedings of the International Conference on Very Large Databases*, pages 217–225, 1987.
18. Ph. Rigaux and N. Spyratos. Metadata Inference for Document Retrieval in a Distributed Repository (Invited Paper). In *Proceedings of the 9th Asian Computing Science Conference (ASIAN'04) (LNCS 3321), Chiang-Mai, Thailand*, 2004.

19. N. Spyratos. A Functional Model for Dimensional Data Analysis. Course notes 2001-2004, LRI Research Report, To appear.
20. N. Spyratos. Decision Support Problems. Course notes 2001-2004, LRI Research Report, To appear.