Preference-based Query Tuning through Refinement/Enlargement in a Formal Context

Nicolas Spyratos[1] and Carlo Meghini[2]

[1] Université Paris-Sud, Laboratoire de Recherche en Informatique, Orsay Cedex, France spyratos@lri.fr
[2] Consiglio Nazionale delle Ricerche, Istituto della Scienza e delle Tecnologie della Informazione, Pisa, Italy meghini@isti.cnr.it

Abstract. The user of an information system rarely knows exactly what he is looking for, but once shown a piece of information he can quickly tell whether it is what he needs. Query tuning is the process of searching for the query that best approximates the information need of the user. Typically, navigation and querying are two completely separate processes, and the user usually has to switch often from one to the other–a painstaking process producing a frustrating experience. In this paper, we propose an approach to query tuning that integrates navigation and querying into a single process, thus leading to a more flexible and more user friendly method of query tuning. The proposed approach is based on formal concept analysis, and models the directory of an information source as a formal context in which the underlying concept lattice serves for navigation and the attributes of the formal context serve for query formulation. In order to support the user in coping with a possibly overwhelming number of alternative query tunings, preferences are introduced.

1 Introduction

The work reported in this paper originates in the following basic observation: the user of an information system rarely knows exactly what he is looking for, but once shown a piece of information he can quickly tell whether it is what he needs.

Query tuning is the process of searching for the query that best approximates the information need of the user. We note that, finding the query that best expresses a given information need is important not only for retrieving the information satisfying the current need, but also in order to name and store the query for use at some later time (without having to re-invent it over and over).

In advanced information systems, query tuning proceeds in two steps: (a) the user navigates the information space until he finds a subspace of interest and (b) in that subspace, the user issues a query. If the answer to the query is satisfactory then the session terminates, otherwise a new navigation step begins. However, navigation and querying are two completely separate processes, and the user usually has to switch often from one to the other -a painstaking process usually producing a frustrating experience.

In this paper, we propose an approach to query tuning that interleaves, or integrates navigation and querying into a single process, thus leading to a more flexible and more user friendly method of query tuning.

The proposed approach is based on Formal Concept Analysis (FCA, from now on), and models the directory of an information source (IS, for short) as a formal context in which the objects represent documents and the attributes represent indexing terms [16]. As a result, the concepts of the underlying concept lattice represent meaningful classes of documents, in the sense that all documents in a class share the same set of indexing terms. Therefore, we propose to use the concept lattice as the basic navigation tool.

We assume the user queries to be Boolean combinations of indexing terms, and more specifically conjunctions of indexing terms. Indeed, the objective of this paper is not to propose a new, more powerful query language but, rather, use an existing (simple) language in order to illustrate our approach to query tuning.

Our approach is user-controlled, and proceeds as a sequence of refine/enlarge commands until a user-approved query is obtained. More precisely, query tuning in our approach proceeds roughly as follows:

- *Query mode* The user formulates a query to the IS and receives an answer; if the answer is satisfactory then the session terminates and the issued query is considered tuned, otherwise the user can issue a Refine or an Enlarge command.
- *Refine* A Refine command returns all maximal concepts (from the concept lattice) that refine the user query, in the sense that their extent is strictly included in the answer to the user query; then the user can decide to either return to query mode and query one or more of those concepts, or terminate.
- *Enlarge* An Enlarge command returns all minimal concepts (from the concept lattice) that subsume the user query, in the sense that their extent strictly includes the answer to the user query; then the user can decide to either return to query mode and query one or more of those concepts, or terminate.

In what follows, we firstly relate our work to existing results; then we introduce the IS model (Section 3) and recall the basic notions needed from formal concept analysis (Section 4). Our query tuning approach is presented in Section 5 and illustrated through a running example. The preference-based ordering of query tunings is given in Section 6. Finally, we offer some concluding remarks in Section 7.

2 Related work

The use of FCA in information system is not new. The structuring of information that FCA supports has inspired work on browsing [13, 3], clustering [4], and ranking [6, 15]. A basic drawback of these approaches is that they require the computation of the whole concept lattice, whose size may be exponential in that

of the context, as it will be argued below. An integrated approach to browsing and querying that uses only part of the lattice, and thus can be computed efficiently, is presented in [5].

Preferences, on the other hand, are enjoying a vast popularity, due to their ability of capturing user requirements. In general, preferences can be captured either quantitatively, or qualitatively as formulas inducing orderings [9, 14]. Our approach subscribes to the latter view.

Our approach extends efficient, FCA-based query tuning by considering qualitatively expressed preferences.

3 Information sources

Given the foundational nature of our work, we deliberately adopt a simple model of an information source, close in spirit to that of a *digital library,* (or DL for short). Essentially, a DL serves a network of providers willing to share their documents with other providers and/or consumers (hereafter, collectively called "users"). Each document resides at the local repository of its provider, so all providers' repositories, collectively, can be seen as a distributed repository of documents spread over the network. The DL system acts as a mediator, supporting transparent access to all sharable documents by the library users. Existing DL systems are consistent with this view [7, 8].

Two of the basic services supported by the library are *document registration* and *querying.*

3.1 Document registration

When a provider wishes to make a document sharable over the network of users he must register it at the library. To do so he must provide two items to the library:

- the document identifier
- the document description

We assume that the document identifier is a global identifier, such as a URI, or just the URL where the document can be accessed, (however, for convenience of notation, we use integers as document identifiers in our examples). As for the document description, we consider only content description and we assume that such a description is given by selecting a set of terms from a controlled vocabulary. For example, the document description $\{\mathsf{QuickSort}, \mathsf{Java}\}$ would indicate that the document in question is about the quick sort algorithm and Java.

Therefore, to register a document, its provider submits to the library an identifier i and a set of terms D. We assume that 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. In our previous example, if i is the document identifier, the library will store two pairs: $(\mathsf{i}, \mathsf{QuickSort})$ and $(\mathsf{i}, \mathsf{Java})$. The set of all such pairs (i, t) is what we call the *library directory,* or simply *directory* (the well known Open

Directory [2] is an example of such a directory). Clearly the directory is a binary relation between document identifiers and terms, *i.e.* a formal context in the sense defined in the next section.

3.2 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. Search for documents of interest is done by issuing queries to the library management system, and the library management system uses its directory to return the identifiers (*i.e.*, the URIs) of all documents satisfying the query.

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

$$q ::= t \mid q_1 \wedge q_2 \mid q_1 \vee q_2 \mid q_1 \wedge \neg q_2 \mid (q)$$

where t is any term.

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

if q is a term *then* $ans(q) = \{i \mid (i, q)$ is in the directory$\}$
else begin
  if q is $q_1 \wedge q_2$ *then* $ans(q) = ans(q_1) \cap ans(q_2)$
  if q is $q_1 \vee q_2$ *then* $ans(q) = ans(q_1) \cup ans(q_2)$
  if q is $q_1 \wedge \neg q_2$ *then* $ans(q) = ans(q_1) \setminus ans(q_2)$
  end

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

The reader familiar with logic will have recognized documents as individuals, terms as unary predicate symbols and the library directory as an interpretation of the resulting logic language; in other words, the presence of a pair (i, t) in the library directory means that the individual i is in the interpretation of term t. Query answering can then be seen as based on the notion of satisfaction: an individual i is returned in response to a query q just in case i is in the extension of q (in the current interpretation).

4 Formal Concept Analysis

Formal concept analysis (hereafter FCA for short) is a mathematical tool for the analysis of data based on lattice theory [11, 10, 12, 1].

Let $\mathcal{O}$ be a set of objects and $\mathcal{A}$ a set of attributes. A *formal context*, or simply *context* over $\mathcal{O}$ and $\mathcal{A}$ is a triple $(\mathcal{O}, \mathcal{A}, C)$, where $C \subseteq \mathcal{O} \times \mathcal{A}$, is a binary relation between $\mathcal{O}$ and $\mathcal{A}$, called the *incidence* of the formal context. Figure 1

shows a formal context in tabular form, in which objects correspond to rows and attributes correspond to columns. The pair (o, a) is in C (that is, object o has attribute a) if and only if there is a x in the position defined by the row of object o and the column of attribute a.

	A	B	C	D	E	F
1	x		x	x	x	
2		x	x			
3	x		x	x		x
4		x	x	x	x	x
5	x	x			x	x

Fig. 1. A Formal Context

Let i and e be respectively the functions *intension* and *extension* as they are normally used in information systems, that is:

$$\text{for all } o \in \mathcal{O}, \quad i(o) = \{a \in \mathcal{A} \mid (o, a) \in C\}$$
$$\text{for all } a \in \mathcal{A}, \quad e(a) = \{o \in \mathcal{O} \mid (o, a) \in C\}.$$

In Figure 1, the intension of an object o consists of all attributes marked with a x in the row of o; the extension of an attribute a consists of all objects marked with a x in the column of a. Now define:

$$\text{for all } O \subseteq \mathcal{O}, \quad \varphi(O) = \bigcap\{i(o) \mid o \in O\}$$
$$\text{for all } A \subseteq \mathcal{A}, \quad \psi(A) = \bigcap\{e(a) \mid a \in A\}.$$

A pair (O, A), $O \subseteq \mathcal{O}$ and $A \subseteq \mathcal{A}$, is a *formal concept* of the context $(\mathcal{O}, \mathcal{A}, C)$ if and only if $O = \psi(A)$ and $A = \varphi(O)$. O is called the *extent* and A the *intent* of concept (O, A). In the formal context shown in Figure 1, $(\{1, 3, 4\}, \{C, D\})$ is a concept, while $(\{1, 3\}, \{A, D\})$ is not. The computation of φ for a given set of objects O requires the intersection of $|O|$ sets, thus it requires $O(|\mathcal{O}| \cdot |\mathcal{A}|^2)$ time. Analogously, the computation of ψ for a given set of attributes A requires $O(|\mathcal{A}| \cdot |\mathcal{O}|^2)$ time. These are both upper bounds that can be reduced by adopting simple optimization techniques, such as ordering. In addition, they are worst case measures.

The concepts of a given context are naturally ordered by the *subconcept-superconcept* relation defined by:

$$(O_1, A_1) \leq (O_2, A_2) \text{ iff } O_1 \subseteq O_2 (\text{iff } A_2 \subseteq A_1)$$

This relation induces a lattice on the set of all concepts of a context. For example, the concept lattice induced by the context of Figure 1 is presented in Figure 2.

There is an easy way to "read" the extent and intent of every concept from the lattice. To this end, we define two functions γ and μ, mapping respectively objects and attributes into concepts, as follows:

$$\gamma(o) = (\psi(\varphi(\{o\})), \varphi(\{o\})) \text{ for all } o \in \mathcal{O}$$
$$\mu(a) = (\psi(\{a\}), \varphi(\psi(\{a\}))) \text{ for all } a \in \mathcal{A}.$$

It is easy to see that $\gamma(o)$ and $\mu(a)$ are indeed concepts. In addition, $(o, a) \in C$ is equivalent to $\gamma(o) \leq \mu(a)$. The functions γ and μ are represented in Figure 2 by labeling the node corresponding to the concept $\gamma(o)$ with o as a subscript, and the node corresponding to concept $\mu(a)$ with a as a superscript. Finally, it can be proved that for any concept $c = (O_c, A_c)$, we have:

$$O_c = \{o \in \mathcal{O} \mid \gamma(o) \leq c\}$$
$$A_c = \{a \in \mathcal{A} \mid c \leq \mu(a)\}.$$

Thus, the extent of a concept c is given by the objects that label the concepts lower than c in the concept lattice. For example, the extent of the concept labelled by A, that is $\mu(A)$, is $\{1, 3, 5\}$. Analogously, the intent of c is given by the attributes that label concepts higher than c in the lattice, just $\{A\}$ for $\mu(A)$. It follows that

$$\mu(A) = (\{1, 3, 5\}, \{A\}).$$

By the same token, it can be verified that

$$\gamma(3) = (\{3\}, \{A, C, D, F\}).$$

Notice that this reading is consistent with the reading of class hierarchies in object-oriented languages. In such hierarchies, objects are inherited "upward", *i.e.* by the classes that are more general than the classes where they belong, while attributes are inherited "downward", *i.e.* by the classes that are more special than the classes that define them.

It is easy to see the following:

1. $O_1 \subseteq O_2$ implies $\varphi(O_1) \supseteq \varphi(O_2)$; $A_1 \subseteq A_2$ implies $\psi(A_1) \supseteq \psi(A_2)$; $O \subseteq \psi \circ \varphi(O)$; and $A \subseteq \varphi \circ \psi(A)$; hence, φ and ψ form a Galois connection between the powerset of $\mathcal{O}$, $\mathcal{P}(\mathcal{O})$, and that of $\mathcal{A}$, $\mathcal{P}(\mathcal{A})$.
2. $O \subseteq \psi \circ \varphi(O)$; $O_1 \subseteq O_2$ implies $\psi \circ \varphi(O_1) \subseteq \psi \circ \varphi(O_2)$; and $\psi \circ \varphi(\psi \circ \varphi(O)) = O$.
   The above 3 properties of $\psi \circ \varphi$ tell us that $\psi \circ \varphi$ is a closure operator on $\mathcal{P}(\mathcal{O})$, and similarly $\varphi \circ \psi$ is a closure operator on $\mathcal{P}(\mathcal{A})$.

Since closure systems may be exponentially large in the size of their domain, a concept lattice may have an exponential number of concepts in the size of the underlying context. For this reason, any approach that requires the computation of the whole concept lattice is bound to be intractable, in general. On the other hand, our approach only requires the computation of small portions of the lattice, namely that consisting of single concepts and their lower or upper neighbors. This guarantees the tractability of the involved operations, as it will be argued in due course.

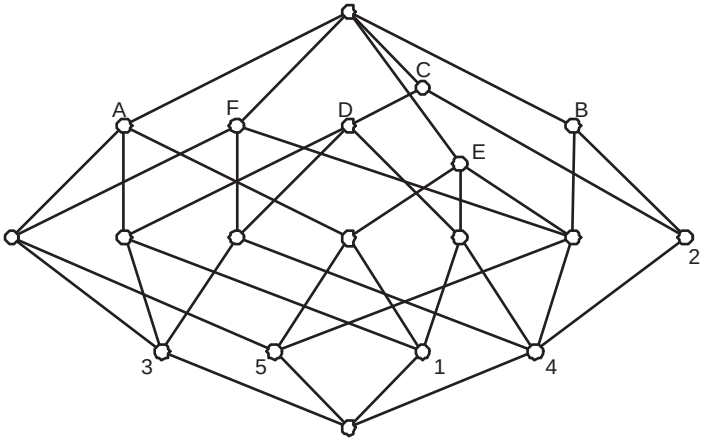

Fig. 2. A concept lattice

4.1 Digital libraries and FCA

We are now in a position to make precise the relationship between DLs and FCA. The directory $\mathcal{D}$ of a DL can be seen as a formal context, defined as follows:

- The objects of the formal context are the identifiers of the documents registered in $\mathcal{D}$; we call the set of all such identifiers the *domain* of $\mathcal{D}$, and denote it by $dom(\mathcal{D})$.
- The attributes of the formal context are the terms which appear in at least one document description; we call the set of all such identifiers the *range* of $\mathcal{D}$, and denote it by $ran(\mathcal{D})$.
- The incidence of the formal context is just $\mathcal{D}$, the DL directory.

So, corresponding to the DL directory $\mathcal{D}$ we have the formal context ($dom(\mathcal{D})$, $ran(\mathcal{D})$, $\mathcal{D}$). Each concept of this context may be understood as a *class* of documents. The extent of the concept gives the instances of the class, while its intent gives the attributes (terms) defining the class. By definition of concept, the extent is the largest set of documents having the attributes in the intent and, dually, the intent is the largest set of attributes shared by the documents in the extent. This is compliant with the notion of class as it stems from object-oriented information modelling, thus it is a merit of FCA to offer a mathematically well-founded and intuitively appealing criterion to model information.

In addition, unlike the traditional approach in which classes are manually derived in the design of the information system, here classes are automatically derived solely on the base of the available information. In this way, they do not run the risk of becoming a straightjacket: they are a perfect fit for the underlying information at any point of the system lifetime, since their definition evolves with the evolving of the information. This is very valuable in the context of DLs,

for which the identification of classes is more difficult than in a traditional information system, due to the complexity and the heterogeneity of the documents, and to the multiplicity of their usage.

Our work is a first exploitation of the notion of formal concept in DLs, hinging on a basic link between concepts and conjunctive queries. Indeed, the intent of each concept in the context ($dom(\mathcal{D})$, $ran(\mathcal{D})$, $\mathcal{D}$) is actually the most specific conjunctive query whose answer is the extent of that concept. In our example, on Figure 1 it can be verified that the queries: $(C \wedge F)$, $(D \wedge F)$ and $(C \wedge D \wedge F)$ are the ones (and the only ones) having the set of documents $\{3, 4\}$ as an answer. The most specific of these queries is $(C \wedge D \wedge F)$ and in fact $(\{3, 4\}, \{C, D, F\})$ is a formal concept.

Conversely, any answer to a conjunctive query is the extent of some concept. For example, the set of documents $\{1, 2, 3\}$ is not the extent of a concept and in fact no conjunctive query can produce it as an answer.

It follows that, by navigating the concept lattice, the user can be guided to the best result he can obtain with a conjunctive query. These facts and observations lie at the heart of our query tuning approach that we present in the next section.

5 Query Tuning

During user interaction with the digital library, query tuning is obtained by using four commands: *Query, Terminate, Refine* and *Enlarge.* In this section we define each of these commands separately and illustrate them in our running example. We recall that we restrict our attention to conjunctive queries only.

5.1 Query

The user issues a query to the system. The system returns the answer, obtained by evaluating $\psi(A)$ where A is the set of terms in the query. In fact:

$$\begin{aligned} ans(\bigwedge A) &= \bigcap \{ans(a) \mid a \in A\} \quad \text{(by definition of } ans \text{ on conjunctions)} \\ &= \bigcap \{\{o \in \mathcal{O} \mid (o, a) \in C\} \mid a \in A\} \quad \text{(by definition of } ans \text{ on terms)} \\ &= \bigcap \{e(a) \mid a \in A\} \quad \text{(by definition of } e) \\ &= \psi(A) \quad \text{(by definition of } \psi). \end{aligned}$$

The system also shows to the user the most precise (*i.e.*the most specific) query that can be used to obtain the same result; this query is the conjunction of all terms in $\varphi(\psi(A))$. In other words, upon evaluating a query $\bigwedge A$ the system "places" itself on the concept $(\psi(A), \varphi(\psi(A)))$, which becomes the current concept. This placement can be obtained using a polynomial amount of time, since it requires the computation of ψ on the set of terms making up the query, followed by a computation of φ on the result.

In our example, let us assume the user poses a query consisting of a single term, say D. In response, the system returns the answer $\psi(D) = \{1, 3, 4\}$ and

shows the query $C \wedge D$, since $\varphi(\psi(D)) = \{C, D\}$. The current concept is therefore $\mu(D)$ in Figure 2.

5.2 Terminate

The user is satisfied by the answer and issues a *Terminate* command. The system switches to next-query mode; otherwise, the user performs one of the two actions described next.

5.3 Refine

The user judges the answer to be too rich, *e.g.* the cardinality of the answer set is too big or, upon inspection, there are too many irrelevant answers in the answer set; and issues a *Refine* command. The system then computes and returns to the user the following pair of information items, for each concept *max* whose extent O_{max} is a maximal subset of the current answer:

1. The intent A_{max}
2. The objects in the current answer lying outside O_{max}.

Let us explain further these two items that are returned to the user. Any concept like *max* above is a maximal sub-concept of the current concept, or, in other words, a lower neighbor of the current concept in the concept lattice. By definition, the intent of such a concept is a superset of the current concept intent. The system computes each such concept *max* and shows its intent to the user, by simply presenting him the additional terms of the intent with respect to the current concept intent. This computation can be done in polynomial time with respect to the size of the context. Let us see how in our example.

Let us assume that the current answer is $\{1, 3, 4\}$, and that the user considers this answer to be too large, so he executes a *Refine.* To compute the terms to be shown to the user, the system looks at a smaller context, consisting of the objects in the extent of the current concept, and of the attributes outside the intent of the current concept. The context we are looking at is:

	A	B	E	F
1	×		×	
3	×			×
4		×	×	×

In order to achieve maximality, we select the terms which have maximal extension in this context; these are A, E and F. Each of these terms t must be shown to the user, since it leads to a maximal sub-concept of the current concept, given by:

$$(\psi(A_c \cup \{t\}), \varphi(\psi(A_c \cup \{t\})))$$

where A_c is the intent of the current concept. In our example, we recall that the current concept is $\mu(D) = (\{1, 3, 4\}, \{C, D\})$. Then, term A leads to concept

$(\{1,3\},\{A,C,D\})$, term E leads to concept $(\{1,4\},\{C,D,E\})$, term F leads to concept $(\{3,4\},\{C,D,F\})$. All these concepts are maximal sub-concepts of the current concept; as a consequence, each has a larger intent, which means that its associated query is a refinement of the query originally posed by the user. Figure 3 shows the 3 maximal sub-concepts (green) of the current concept (light blue) in the concept lattice; next to each one, the additional term is shown.

The second item shown to the user is the set of objects in the extent of the current concept lying outside O_{max}. Intuitively, these are the objects that will be "lost" by selecting the corresponding refinement. For instance, along with the term F, the user is shown the object set $\{1\}$ containing the answers which will no longer be answers if the query is refined by adding the term F to it.

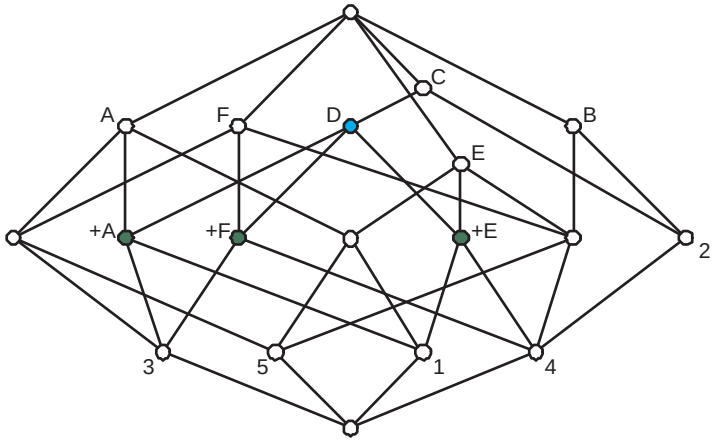

Fig. 3. The maximal sub-concepts of the current concept

The complete answer that the user gets in response to a *Refine* in our example is reported in Table 1. For convenience, the Table also shows the refined query and the concept corresponding to each solution.

Option	Added Terms	Lost Objects	Refined Query	Concept
1	$\{A\}$	$\{4\}$	$A \wedge C \wedge D$	$(\{1,3\},\{A,C,D\})$
2	$\{E\}$	$\{3\}$	$C \wedge D \wedge E$	$(\{1,4\},\{C,D,E\})$
3	$\{F\}$	$\{1\}$	$C \wedge D \wedge F$	$(\{3,4\},\{C,D,F\})$

Table 1. Result of a *Refine*

Upon deciding whether to accept a proposed refinement, the user can figure out the attributes he gains by inspecting the added terms, or the answers he looses by inspecting the lost objects. If the user does decide to move, then the concept corresponding to the selected solution becomes the current concept, and a new interaction cycle begins (by querying, refining and enlarging).

Notice that if no maximal sub-concept exists (*i.e.* the current concept is the least concept of the lattice), then the system returns empty and, subsequently, the user may issue an *Enlarge* command (see below) or try a new query.

5.4 Enlarge

The user judges the answer to be too poor (*e.g.*, the cardinality of the answer set is too small, possibly zero), and issues an *Enlarge* command. The system then computes and returns to the user the following pair of information items, for each concept *min* whose extent O_{min} is a minimal superset of the current answer (in a symmetric manner as in the case of *Refine*):

1. The intent A_{min}.
2. The objects in O_{min} which lay outside the extent of the current context.

Each such concept *min* is a minimal super-concept of the current concept, or an upper neighbor. The set of all such concepts *min* can be computed in polynomial time in an analogous way to the maximal sub-concepts. Let us again see how in our example.

Let us assume that the user refines the initial query by selecting option 1 in Table 1, thus making $(\{1,3\},\{A,C,D\})$ the current concept, and that he then asks to enlarge this set. To compute the objects leading to a minimal super-concept of the current concept, we look at a smaller context, consisting of just the attributes in the intent of the current context and of the objects outside the extent of the current context. That is:

	A	C	D
2		×	
4		×	×
5	×		

From this context we select the objects with maximal intention, that is 4 and 5. Each of these objects o leads to a minimal super-concept of the current concept, given by:

$$(\psi(\varphi(O_c \cup \{o\})), \varphi(O_c \cup \{o\}))$$

where O_c is the extent of the current concept. In our example, object 4 leads back to concept $(\{1,3,4\},\{C,D\})$ while object 5 leads to concept $(\{1,3,5\},\{A\})$, which are all minimal super-concepts of the current concept. Figure 4 shows the 2 minimal super-concepts (red) of the current concept (light blue) in the concept lattice; next to each one, the additional term is shown. Notice that the query

associated to each such concepts is a relaxation of the query associated to the current concept.

The complete answer that the user gets in response to an *Enlarge* in our example is reported in Table 2. For each option, the Table shows the terms that are lost in the query enlargement, the added objects, the enlarged query and the corresponding concept. Upon deciding whether to accept a proposed enlargement, the user can figure out the attributes he looses or the answers he gains. If the user does decide to move, then the concept corresponding to the selected option becomes the current concept, and a new interaction cycle begins (by querying, refining and enlarging).

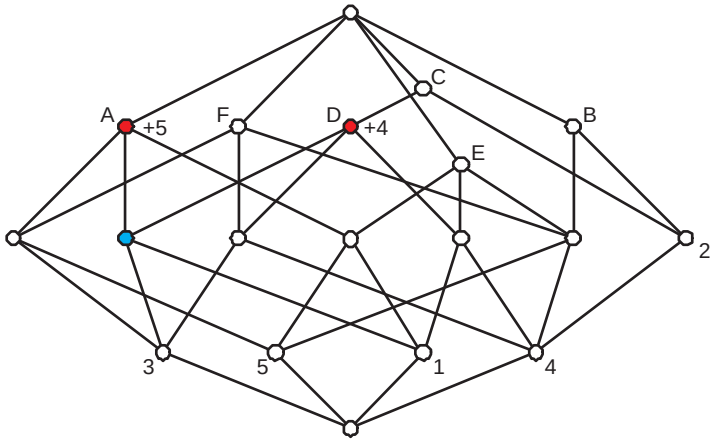

Fig. 4. The minimal super-concepts of the current concept

Option	Lost Terms	Added Objects	Enlarged Query	Concept
1	A	4	$C \wedge D$	$(\{1,3,4\},\{C,D\})$
2	C, D	5	A	$(\{1,3,5\},\{A\})$

Table 2. Result of an *Enlarge*

Notice that if no minimal super-concept exists (*i.e.* the current concept is the greatest concept of the lattice), then the system returns empty and, subsequently, the user may issue a *Refine* command or try a new query.

6 Introducing preferences

Preferences are a way of capturing user specific requirements in the usage of a DL. As the amount of information in a DL and the number of its users may grow very rapidly, capturing user preferences may be a very valuable tool to make the DL usable.

We will model the preferences of a given user u, as a partial order P_u (simply P when there is no ambiguity) over the terms employed for content description. If terms a and b are in the P relation, *i.e.* $(a, b) \in P$, we say that a is preferred over b (by user u).

P may be derived in several ways, for instance by having the user declare preferences, or by mining them from user actions, or both. In addition, while reflexivity and transitivity seem reasonable properties for a preference relation, there is strong evidence that antisymmetry is not so. However, we have assumed it because it is a well-known mathematical fact that a partial order can be uniquely derived from a pre-order by considering term equivalence. We will do this below for term set, thus for the time being suffice it to say that there is no loss in generality by considering P to be a partial order.

6.1 Using preferences in query refining

Upon performing a query refinement, the user is confronted with the set of the maximal sub-concepts of the current concept, the intents of which give the alternative query refinements, as illustrated in Table 1. Let us call this latter set A_{ref}. Now, in a realistic setting A_{ref} may contain dozens of concept intents, making it hard for the user to merely inspect the result of a *Refine,* let alone to select an alternative. Preferences may help solving this problem. They can be used to partition the set A_{ref} into blocks of equally preferred concept intents, which are shown to the user in decreasing order of preference. In this way, the output is divided into more consumable portions, and is offered to the user in a sensible way. In order to achieve this goal, we need to determine: (a) a partial order between concepts based on preferences; and (b) a way to use this order for defining a suitable partition of A_{ref}.

As for the former goal, it is natural to consider a concept (A, B) preferred over a concept (A', B') based on a criterion involving the intents B and B', which are sets of terms and can therefore directly reflect preferences. Since no two concepts can have the same intents, any order defined on intents can be understood also as an order defined on concepts. Moreover, since the set of concept intents is a subset of $\mathcal{P}(\mathcal{A})$, any partial order defined on the latter carries over the former by inheritance. It follows that goal (a) above amounts to define a partial order between term sets.

Given two term sets $B, B' \subseteq \mathcal{A}$, B is said to be *preferred over* B', written as $B \preceq B'$, if for all terms $b \in B$ there exists a term $b' \in B'$ such that $(b, b') \in P$. It is not difficult to see that $\preceq$ is a pre-order on $\mathcal{P}(\mathcal{A})$: both its reflexivity and transitivity are implied by those of P. Antisymmetry does not hold for $\preceq$: it suffices to consider the term sets $\{a, b\}$ and $\{a, b, c\}$ such that (c, b) is in P.

Let us define two term sets B and B' *preferentially equivalent,* written $B \equiv B'$, if $B \preceq B'$ and $B' \preceq B$. Now $\equiv$ is an equivalence relation on $\mathcal{P}(\mathcal{A})$, whose induced equivalence classes we denote as $[B]$, *i.e.* $[B] = \{X \subseteq \mathcal{A} \mid X \equiv B\}$. $\preceq$ can be extended to the so defined equivalence classes as follows:

$$[B] \preceq [B'] \text{ iff } B \preceq B'.$$

As it can be verified, $\preceq$ is a partial order on equivalence classes. Thus by replacing the notion of term set with that of class of equivalent term sets we have a partial order on term sets, and have so accomplished goal (a) above. Let us resume the example on refinement developed in Section 5.3, whose result is shown in Table 1. In this example, we have

$$A_{ref} = \{\{A, C, D\}, \{C, D, E\}, \{C, D, F\}\}.$$

Assuming that the Hasse diagram of P includes only the pairs (A, E) and (A, F), the only comparable classes amongst those relevant to A_{ref} are:

$[\{A, C, D\}] \preceq [\{C, D, E\}]$ and
$[\{A, C, D\}] \preceq [\{C, D, F\}]$.

In order to accomplish goal (b), below we describe how the set A_{ref} of concept intents resulting from a *Refine* is partitioned, leaving aside for simplicity the other information returned to the user. The partition in question is given by the sets $A_1, A_2, \ldots$, defined as ($k \geq 1$):

$$A_k = \min_{\preceq} S_k$$

where S_k consists of the elements of A_{ref} which have not yet been inserted in any block, and is given by ($k \geq 1$):

$$S_1 = A_{ref}$$
$$S_{k+1} = S_k \setminus A_k$$

Thus, A_1 consists of the minimal elements of A_{ref}, A_2 of the minimal elements of A_{ref} after the removal of the minimal elements, and so on.

In our example:

$$S_1 = \{\{A, C, D\}, \{C, D, E\}, \{C, D, F\}\}$$
$$A_1 = \{\{A, C, D\}\}$$
$$S_2 = \{\{C, D, E\}, \{C, D, F\}\}$$
$$A_2 = \{\{C, D, E\}, \{C, D, F\}\}$$
$$S_i = A_i = \emptyset \text{ for } i \geq 3.$$

Accordingly, the user is first shown the first and third row of Table 1 and then the second row.

Formally, it can be shown that, for any set of term sets A_{ref} and partial ordering on $\mathcal{A}$, the following hold:

1. there exists $m \geq 1$ such that for all $j \geq m$, $S_j = A_j = \emptyset$.
2. $\{A_j \mid 1 \leq j < m\}$ is a partition of A_{ref}.
3. for all pairs of term sets $B, B' \in A_{ref}$ such that $[B] \preceq [B']$ and $[B] \neq [B']$, there exist i, j such that $B \in A_i$, $B' \in A_j$ and $i < j$.

The first statement can be made more precise by proving that m actually equals the length of the longest path in the graph $(G, \preceq|_G)$, where $G = \{[B] \mid B \in A_{ref}\}$.

We conclude by observing that the ordering criterion defined above preserves the tractability of the original method. Indeed, the computation of each element of the sequence $A_1, \ldots, A_m$ requires a polynomial amount of time in the size of the Hasse diagram of P, A_{ref} and $\mathcal{A}$. Since also the length m of the sequence, as just argued, is polynomially bound by the size of A_{ref}, we have the polynomial time complexity of the method.

6.2 Using preferences in query enlargement

The result of an *Enlarge* is (see Table 2) the set of the minimal super-concepts of the current concept, the intents of which represent enlargements of the underlying query. Evidently, also in this case the size of this set, hence the amount of information to be displayed to the user may be overwhelming, and ordering can result in a significant benefit. To this end, the same method described in the previous section can be applied.

7 Concluding remarks

We have seen an approach to query tuning that combines navigation and querying into a single process thus providing a more flexible and more user friendly interaction between the users and the information system.

In the traditional approach, the interaction proceeds by repeating the following two steps (in some order): (1) Query and Terminate, or (2) Navigate. In our approach, the interaction proceeds by repeating the following three steps (in some order) before terminating: (1) Query, (2) Refine, or (3) Enlarge. Here, Refine and Enlarge represent navigation steps that might be interleaved with the Query step and might be repeated several times before termination, *e.g.*, Query-Enlarge-Query-Refine-Query-Enlarge- ... -Terminate.

In order to support the user in selecting, or even just in examining, the possibly many alternative query tunings, we have used preferences. The adopted preferential relation is very liberal, thus resulting into a pre-ordering rather than into a partial ordering as customary for preferences. This problem had been circumvented by considering the ordering induced on equivalence classes.

All the involved problems have been shown to be computationally tractable.

Acknowledgements The authors gratefully acknowledge the DELOS Network of Excellence on Digital Libraries for partially supporting this work, under Task 2.10 "Modeling of User Preferences in Digital Libraries" of JPA2.

References

1. Formal concept analysis homepage. `http://www.upriss.org.uk/fca/fca.html`.
2. The open directory project. `http://dmoz.org/about.html`.
3. C. Carpineto and G. Romano. Information retrieval through hybrid navigation of lattice representations. *International Journal of Human-Computer Studies*, 45(5):553–578, 1996.
4. C. Carpineto and G. Romano. A lattice conceptual clustering system and its application to browsing retrieval. *Machine Learning*, 24(2):95–122, 1996.
5. C. Carpineto and G. Romano. Effective reformulation of boolean queries with concept lattices. In *Proceedings of International Conference on Flexible Query Answering Systems*, number 1495 in Lecture Notes in Artificial Intelligence, pages 83–94, Roskilde, Denmark, May 1998. Springer Verlag.
6. C. Carpineto and G. Romano. Order-theoretical ranking. *Journal of American Society for Information Science*, 51(7):587–601, 2000.
7. Hsinchun Chen, editor. *Journal of the American Society for Information Science. Special Issue: Digital Libraries: Part 1*, volume 51. John Wiley & Sons, Inc., 2000.
8. Hsinchun Chen, editor. *Journal of the American Society for Information Science. Special Issue: Digital Libraries: Part 2*, volume 51. John Wiley & Sons, Inc., 2000.
9. Jan Chomicki. Preference formulas in relational queries. *University at Buffalo, Buffalo, New York ACM Transactions on Database Systems*, 28(4), December 2003.
10. B.A. Davey and H.A. Priestley. *Introduction to lattices and order*, chapter 3. Cambridge, second edition, 2002.
11. B. Ganter and R. Wille. Applied lattice theory: Formal concept analysis. `http://www.math.tu.dresden.de/~ganter/psfiles/concept.ps`.
12. Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer Verlag, 1st edition, 1999.
13. R. Godin, J. Gecsei, and C. Pichet. Design of a browsing interface for information retrieval. In *Proceedings of SIGIR89, the Twelfth Annual International ACM Conference on Research and Development in Information Retrieval*, pages 32–39, Cambridge, MA, 1989.
14. Andrka H., Ryan M., and Schobbens P-Y. Operators and laws for combining preference relations. *Journal of Logic and Computation*, 12(1):13–53, February 2002.
15. Uta Priss. Lattice-based information retrieval. *Knowledge Organization*, 27(3):132–142, 2000.
16. P. Rigaux and N. Spyratos. Metadata inference for document retrieval in a distributed repository. In *Proceedings of ASIAN'04, 9th Asian Computing Science Conference*, number 3321 in LNCS, Chiang-Mai, Thailand, 8-10 December 2004. Invited Paper.