OntoRefiner, a user query refinement interface usable for Semantic Web Portals

Brigitte Safar, Hassen Kefi, Chantal Reynaud

University of Paris Sud-CNRS (LRI), INRIA (Futurs),
LRI, Building 490, 91405 Orsay Cedex, France
{safar, kefi, cr}@lri.fr
http://www.lri.fr/~safar

Abstract. We present a user interface, the OntoRefiner[1] system, for helping the user to navigate numerous retrieved documents after a search querying a semantic portal which integrates a very important number of documents. Retrieved answers are filtered and the user could be provided only with the answers which are, according to him, the most relevant. The refinement process is based on two technologies, dynamic clustering close to Galois lattice structure combined to the use of a domain ontology. The Galois lattice structure provides a sound basis for the query refinement process. However, its construction as a whole is a very costly process. So, we propose an approach based on the use of a domain ontology, avoiding the construction of the whole Galois lattice. In the paper, we present the algorithm and experimental results.

1 Introduction

Web portals are a convenient way today to make information easily accessible for users. Such portals have been developed for some years in different application domains like tourism, medicine, biology, etc. As when using classical Information Retrieval engines, users of Web portals are often swamped by a mass of answers. To solve this problem, commonly referred to as information overload, classical ranking or clustering schemes may be used in order to help users finding the most relevant documents. We show, in this paper, that specific solutions which take benefit of the specificities of semantic Web portals are more efficient and convenient to the user.

A semantic Web portal is a framework allowing to make information search, whatever done by human beings or machines, more easy and to obtain better results, thanks to the semantic representation of the contents of the queried documents. It is different from the current Web which is essentially syntactic. The structure of the accessible documents is well defined but the content of the Web remains inaccessible to automatic reasoning.

Ontologies play a major role in the construction of the Web semantic. They are used each time modules have to be supported by semantic representations. In the context of semantic Web portals, they are used to provide the vocabulary and

[1] OntoRefiner is supported by France Telecom R & D under Picsel 2 contract.

the structure of metadata to annotate documents and also as an intermediate representation connecting heterogeneous data sources, providing the user with the vocabulary to use to query and then giving him the illusion that he queries a centralized and homogeneous system.

We consider semantic portals allowing users to query documents relative to a same application domain. These semantic portals are composed of an ontology describing this application domain into a hierarchy of terms and a set of documents annotated with terms of the hierarchy. We assume that the annotation process has been made earlier, either in an automatic or in a manual way. Whatever the case is, the annotation process is out of the scope of the paper. Queries are posed as in a boolean search engine using terms coming from the domain ontology. Given a query, the returned documents are documents with a mapping between their annotation and the terms in the query.

OntoRefiner can then be viewed as an interactive interface helping the user to navigate very numerous retrieved documents after a search. Retrieved documents are filtered and the user could be provided only with subcollections of answers which are, according to him, the most relevant.

Our approach combines both clustering techniques and the use of a domain ontology. The refinement process is based on dynamic clustering close to Galois lattice structure. The Galois lattice structure provides a sound basis for the query refinement process. This structure has already been used for supporting browsing retrieval in a database [6, 3, 5, 11]. It is also a very convenient structure for query refinement. Indeed, each node in the lattice can be seen as a query described by a set of terms, the terms which compose the query, and a set of documents, those returned by the query. Given a query represented as a node in the lattice, its refinements correspond to the queries represented by the child nodes. Navigating through a Galois lattice can then be assimilated to a navigation from a query to another, more or less specialized. We detail this point further in section 2. However, such kind of structure has a serious limitation. Its construction as a whole, each time the semantic portal is queried, is not efficient.

So, our approach is also based on the use of a domain ontology. The use of terms linked together by a subsumption relationship allows to **directly** construct only the most general nodes of the lattice, avoiding, that way, the construction of the whole Galois lattice. Moreover the domain ontology allows to identify semantic criteria to refine those nodes. These semantic criteria are terms, called in the paper **axes of refinement**, which annotate the documents composing the most general nodes. The user can select, in a first step, among the most general nodes of the lattice, the most interesting subset of documents which corresponds to a refined query. In a second step, by choosing an axis of refinement among the axes associated to this selected refined query, he adds another term in the query which becomes still more specific, and covers still less of documents. The process realizes, this way, two steps of refinement in the same time, and can be iterated on the set of selected documents until the user is satisfied with the obtained set. Consequently, the parts of the Galois lattice which are not relevant according to the choices of the user will never be built.

Let us take an example to illustrate more precisely an iteration of the refinement process from a user point of view. Given the query Q:{Hotel, AlwaysSunny}, let us assume that we obtain 50 000 answers, that is 50 000 documents accessible from the semantic portal concern both hotel and locations which are always sunny. Our aim is to avoid delivering these 50000 documents brutally with no organization. Consequently, we use OntoRefiner to deliver the answers gradually, and only the most relevant ones. At the first step of the first iteration, initially retrieved documents are separated into several groups. In the example, we may have a first group of documents which concerns hotels located in a sunny place **which is an isle**, a second group of documents which concerns hotels located in places which are **always sunny**, a third group which concerns hotels located in a sunny place, those documents also **containing information about available equipments on site**. At that point, we doesn't allow the user to choose the group of documents that are the most relevant for him.

In a second step, for each group of documents just identified, OntoRefiner searchs axes of refinement. For example, it may propose that among the documents in which the places are isles, there are documents with places which are isles **with a beach** and documents with places which are **sunny in summer**.

Once this two-step refinement is done, the result of the grouping is presented to the user. The user has to choose the set of documents he prefers from the groups obtained from the first refinement step and for this selected set, he must indicate the axis of refinement which corresponds the most to the documents he is interesting in. At the end of the iteration, the initial query is complemented with the terms which characterize the set of chosen documents and the axis of refinement. The second iteration starts from this new more precise query and the associated documents.

The paper is organized as follows. Section 2 presents the Galois lattice clustering method and related work for query refinement. Section 3 introduces the domain ontology and its use for bringing out different points of view in a document description. Then we show, in section 4, how the use of a domain ontology allows the gradual construction of only pertinent parts of the Galois lattice and, in section 5, our algorithm. We conclude with experimental results and a short discussion in section 6.

2 Galois Lattice and Related Work

Given two finite sets D (set of documents) and T (set of terms), and a binary relation $\mathcal{R}$ between these two sets, the Galois lattice[12, 6] is a particular set of nodes, in which each node is a pair, composed of a subset of documents $D' \subseteq D$, called **extent**, and a subset of terms $T' \subseteq T$, called **intent**. Each pair (D', T') must be a complete pair with respect to $\mathcal{R}$, meaning that T' must only contain the terms shared by all the documents in D', and the documents in D' must be precisely those sharing all the terms in T'.

$T' = f_1(D')$ where $f_1(D') = \{t' \in T' \mid \forall d' \in D', d'\mathcal{R}t'\}$

$D' = f_2(T')$ where $f_2(T') = \{d' \in D' \mid \forall t' \in T', d'\mathcal{R}t'\}$

The complete pair **centered around a term** t_i can be computed as $C_i = (f_2(t_i), f_1(f_2(t_i)))$ where $f_1(f_2(t_i))$ is the *closure* of the term t_i.

Example 1 Figure 1 shows an example containing 5 documents described by 7 terms, $\{a, b, c, d, e, f, g\}$. Document 1 is indexed by terms $\{a, c, f\}$. Term g appears in documents 2 and 3. To compute C_b, the complete pair centered around the term b, we compute $f_2(b) = \{4, 5\}$ then $f_1(\{4, 5\}) = \{b, c, f\}$. $C_b = (\{4, 5\}, \{b, c, f\})$.

	a	b	c	d	e	f	g
1	•		•			•	
2	•		•				•
3	•			•			•
4		•	•			•	
5		•	•		•	•	

Fig. 1. An example of relation

Given two pairs $C_1 = (D_1, T_1)$ and $C_2 = (D_2, T_2)$, a partial ordering relation $(<)$ is defined on this set of pairs by: $C_1 < C_2 \Leftrightarrow T_1 \subseteq T_2$ or $C_1 < C_2 \Leftrightarrow D_2 \subseteq D_1$. The partial order is used to generate the lattice graph in the following way: there exists an edge (C_1, C_2) if $C_1 < C_2$ and there is no other element C_3 in the lattice such that $C_1 < C_3 < C_2$. C_1 is called **parent** of C_2 and C_2 **child** of C_1. The graph reveals the generalization/specialization relationship between the pairs corresponding to the subset relationship between the D or T parts.

Example 2 In figure 2a showing the lattice graph corresponding to the relation of figure 1, we can see that the pair ({1,2,3},{a}) is more general than its child, the pair ({2,3},{a,g}): a subset of terms characterizes an overset of documents.

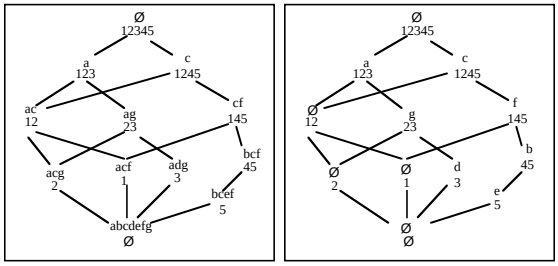

Fig. 2. fig.2a Galois lattice graph fig.2b $T' - ICL$ graph

The Galois lattice representation exhibits a lot of redundancy. For a pair $C = (D', T')$, D' will be present in every ancestor of C and T' will appear in all its descendants. Godin et al. showed in [7] that this redundancy may be eliminated without losing any information if the graph is maintained. For a pair $C = (D', T')$, let T'' be the non redundant element in T', $T'' = \{t' \in T' | t' \in f_1(D')$ and there is no other pair $C' = (X, Y) < C$ such that $t' \in Y\}$. Godin defines the $T'-$ *inheritance concept lattice* $(T' - ICL)$ using the set of pairs (D', T''), and shows that for a given pair C, the corresponding values of T' can be computed by taking the union of the T'' sets of the ancestors of C including C itself. The features of T' not in T'' are inherited from their ancestors. In our work, we will use the set of pairs (D', T''), too.

Example 3 Comparing fig. 2.a and 2.b, we can see that the intent of the pair ({1,2},{a,c}) of fig. 2.a, is empty in fig. 2.b. The intent of this pair can be computed by taking the union of the intent of its ancestors, the pairs ({1,2,3},{a}), ({1,2,4,5},{c}) and ({1,2,3,4,5},∅).

This clustering structure is especially relevant in the context of information retrieval because one cluster is directly described in intension by T' which is a simple set of terms easily readable and understandable by the user. Furthermore, this lattice structure allows overlap: a document may appear in different clusters reflecting different points of view. This is an advantage compared to hierarchical classification for example.

Godin [6] and Carpineto [3] have used this type of structure to index a document collection manually annoted by a set of terms. Once the lattice characterizing the whole database is built, user's queries are compared with the intent of the lattice pairs. Indeed, for each pair, its intent can be seen as a conjunctive query and its extent as the documents answering this query. In this setting, answering a user's query consists in looking for the more general pair whose intent contains all the terms of the considered query. Once this pair is found [2] the user is allowed to explore its direct neighbors in the lattice. The children of the pair correspond to a minimal refinement of the query (additional terms in the query that becomes then more specific) and the parents to a minimal enlargement. These additional terms are, for us, the axes of refinement of the query.

Example 4 In figure 2a, the pair ({2,3},{a,g}) has two children ({2},{a,c,g}) and ({3},{a,d,g}). These child nodes are two different refinements of the query composed of the term a and g respectively obtained by adding the term c or d. c and d are the two possible axes of refinement. The only enlargement of the query is represented by the pair ({1,2,3},{a}).

Ecklund and Cole [5] improve the refinement process using a hierarchy of annotation terms. When assigning terms to documents, only the most specific terms are used and all their subsumers are automatically added. This hierarchy allows to construct pairs at distinct abstraction levels as required by the user. Stojanovic [11] used a hierarchy of terms for computing the relevance of query refinements and ranking them according to user's preferences. [6] or [5] build their lattice structures from a static database. Our context is different since the response documents set is generated dynamically for a user's query: documents clustering is ephemeral and the construction of the whole Galois lattice for each set of answers would be too costly. As Carpineto [4] which only construct the direct neighbors of the user's query, we present in sections 4 and 5, an algorithm which allows to construct only a subpart of the original Galois lattice under the user guidance.

3 Domain Ontology and Description of Documents

We consider a domain ontology composed by a hierarchy of terms (denoted $\mathcal{D}_h$) and expressed by a set of rules of the form: $t_1 \rightarrow t_2$, where t_1 and t_2 are terms. Throughout the paper we will use a toy example as an illustration from the tourism products domain. For such an application, possible ontology terms could be AccomodationPlace, Localization, Equipment, Activities. Figure 3 shows a fragment of the

[2] If the query has no answer in the set of documents, the searched pair does not exist. The problem of cooperative answering is not studied here and the reader interested in this problem has to refer to [3], [1] or [9].

hierarchy describing the localizations according to different points of view like physical and geographical characteristics. The following two definitions establish the notion of subsumer of a term.

Definition 1 *A term t' is a* **direct subsumer** *of a term t if the rule $t \rightarrow t'$ is in $\mathcal{D}_h$. t is a direct specialization of t'.*

A term can have many direct subsumers and specializations.

Definition 2 *t' is a* **subsumer** *of a term t if t' is a direct subsumer of t or if there exists t" such that t' is a direct subsumer of t" and t" is a subsumer of t.*

Example 5 The direct subsumers of Reunion Island are AlwaysSunny, OutOfEurope and IslandWithBeach while other subsumers are PlaceWithBeach, Island, etc. The direct specializations of UnderTheSun are SunnyOnlySummer and AlwaysSunny while other specializations are Corsica, Sahara, etc.

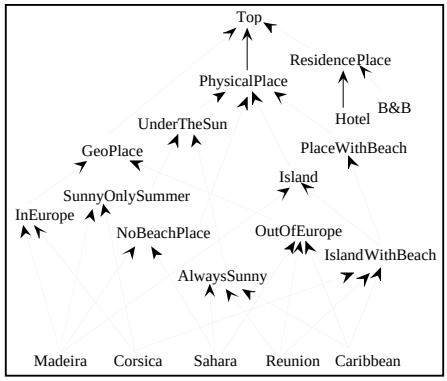

Fig. 3. A fragment of the hierarchy

Documents are supposed to be indexed with the most specialized terms of the domain ontology as in [5] but their description is enriched before clustering. All subsumers in the ontology of the terms indexing the documents are added. This process is called **saturation** and is defined as follows:

Definition 3 *Sat(T) is the* **saturated** *of the set of terms T, if for each term t in T, all subsumers of t are in Sat(T).*

Example 6 Given the hierarchy in figure 3, given a document focusing on a hotel situated on Reunion Island with the description T = {Hotel, Reunion}, Sat(T) = {Hotel, ResidencePlace, Reunion, AlwaysSunny, UnderTheSun, OutOfEurope, IslandWithBeach, PlaceWithBeach, Island, GeoPlace, PhysicalPlace}.

Conversely, we define **desaturation** of a set of terms T as the process of removing from this set all subsumers whose specializations are in T.

Definition 4 *DeSat(T) is the* **desaturated** *of the set of terms T, if it contains all t in T which have no specialization t" in T.*

The purpose of the saturation step is to make appear similar terms between descriptions that do not share any term in their initial description but that describe related documents. For example, the intersection of the descriptions for {Madere} and {Réunion} is empty, but the saturated descriptions share common subsumers {GeoPlace, UnderTheSun, Island}. Saturation allows also to show similarities

between documents at a more abstract level (using more general terms) and along different points of view represented in the ontology. Indeed, ontologies describe the characteristics of terms using various points of view relevant to a particular domain. In the tourism domain, for example, a place will be characterized by its facilities for nautic, swimming or other leisure activities, whereas in another settings, historical, political or geological perspectives will be expressed. When

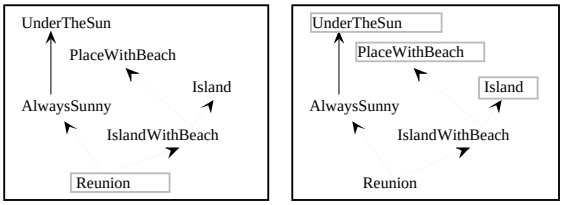

Fig. 4. fig.4a : Desat(T) fig.4b : Mgt(T)

looking for shared terms, we work on non-desaturated document descriptions and we focus on the **most general terms** which are more prone to be shared between descriptions. The most general terms of a set of terms are defined below.

Definition 5 *Mgt(T) is the set of the* **Most general terms** *of a set of terms T if for each term t in Mgt(T), t is in T and it does not exist t' in T such that t' is a subsumer of t.*

Example 7 Given the hierarchy in figure 3, given T = {Reunion, AlwaysSunny, UnderTheSun, IslandWithBeach, PlaceWithBeach, Island}, DeSat(T) = {Reunion} and Mgt(T) = {UnderTheSun, PlaceWithBeach, Island}, respectively represented in figure 4a and 4b in boxes.

We show now how we use these most general terms to only and directly construct the most general nodes of the lattice, the minimal refinements of the user's query.

4 Contribution of the Use of a Domain Ontology in the Construction of a Galois lattice

Our aim, in this section, is to show the impact of the domain ontology, particularly the set of subsumption links between terms, on the Galois Lattice to simplify its building process.

Given a query Q:{X}, let us assume that the retrieved documents are {12345} and the set of the terms which annote these documents is :{abcdefghijx}. Remember that all the terms which annotate the documents are terms in the domain ontology (see figure 5)[3] and let us remark that the most general terms of the annotation terms are Mgt = {a,b,c } which appear in boxes in figure 5. Only the most specialized terms (here, {efghijx}) have been used in the annotation process but the descriptions have been enriched by the saturation process just before computing the Galois lattice. This allows to make appeared new similarities between documents and to identify new groupings of documents.

[3] The term X does not appear in figure 5, because this term belongs to the initial query, then it annotates all the documents and does not allow to discriminate between them. It is the same with the term c in figure 7.b.

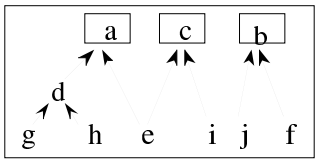

Fig. 5. Subsumption relationships and Mgt

Figure 6 presents first the annotations of the documents once the saturation process has been done and then the Galois lattice which has been built from these annotations. Let us remark in this figure that the nodes in the upper level of the Galois lattice, the most general nodes, the ones whose extent is not included in the extent of the other nodes, are nodes centered around terms belonging to the set of the most general terms (here, {a,c})[4]. The reason is that these most general terms are subsumers of the majority of the other terms. They are in all the documents annotated with one of their specializations, thus in an equal or greater number of documents.

	x	a	b	c	d	e	f	g	h	i	j
1	•	•	•		•		•	•			
2	•	•		•	•	•		•			
3	•	•	•	•		•				•	•
4	•			•						•	
5	•	•	•		•		•		•		

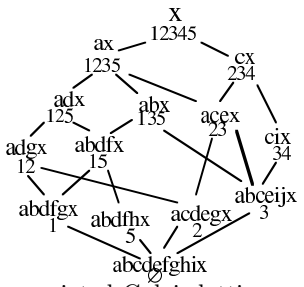

Fig. 6. Annotations of documents and associated Galois lattice

Our gradual construction of the lattice is then based on these most general terms. Let us suppose a set D of documents annotated by a set of terms T which belong to the hierarchy. Comparing performance of algorithms for generating concept lattices, Kuznetsov shows in [8] that in a classical top-down algorithm like, for example, the Bordat algorithm [2], the time complexity of the procedure which finds the most general nodes, the lower neighbours of the node that represents the query, is $O(|D|\times|T|^2)$.

In our approach, using subsumption links between terms, we know that those most general nodes will belong to the nodes centered around the terms in $Mgt(T)$. The time complexity of the algorithm that calculates the node centered around one term is $O(|D|\times|T|)$. As several terms in T have the same subsumers, the number of terms in $Mgt(T)$, $n = |Mgt(T)|$, is lower than $|T|$. Then, our algorithm is less costly since its time complexity is $O(|D|\times|T|\times n)$.

Let us show now that the general nodes obtained this way are the same as the ones calculated by a classical algorithm. Let us suppose there exists a term t such that $t \notin Mgt(T)$ for which there exists a node (X, Y) centered around t and built by a classical algorithm. As before clustering, the description of each

[4] The term b is a most general term too, but all documents annotated by this term are annotated by the term a too, and the extent of the node centered around the term b is included in the one of a.

document is enriched with all subsumers of the terms indexing the document, since $t \notin Mgt(T)$, in each description where t appears there exists a subsumer t' of t such that $t' \in Mgt(T)$. All descriptions of documents X containing t contain t' too which means $t' \in Y$ and the node (X, Y) has been built by our algorithm too, as the closure of t'.

This mechanism can be reused at distinct steps in the algorithm to compute the refinements of any node. Furthermore, as we know that the refined nodes of one node can be built from the most general terms appearing in the descriptions of the documents associated to this node, we can prevent the construction of the refined nodes all the time. A node and its most general terms, which are **possible refinements axes**, can be enough to make the user being aware of the terms he can add in the next step. Moreover, we will see in section 6, that when documents are very numerous, intents of the nodes are usually composed of a unique term, the term around which the node is centered. Then exact computations of the nodes are often useless, but costly.

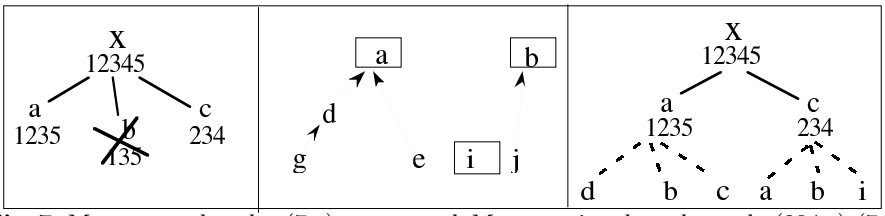

Fig. 7. Most general nodes (7.a), terms and Mgt associated to the node (234,c) (7.b) and axes of refinement of each general node (7.c)

We will precisely show, in the next section, how we are going to work, in an iteration, from one node, called **active node**, that will represent, at the beginning, the initial user's query and then the current refined query. First, the algorithm will really construct the refined nodes (see figure 7.a). Then, it will compute the most general terms associated with each refined node (figure 7.b) and display them (figure 7.c). This way, it provides the user, in the same time, with a vision of two steps of refinement of the active node. This is a means to prevent the exact and costly computation of all the nodes centered around axes of refinement, when only few of these axes are interesting for the user.

5 Algorithm

Given a query Q_0, the inputs of the algorithm are (1) the set D_0 of identifiers of documents with a mapping between their description and all the terms of the query and (2) for each d_i in D_0, a set of terms called **current description** of d_i, $Cdesc(d_i)$, initialized with the saturated of the set of terms describing the document d_i. After n iterations, the output is a refined query Q_n which contains all the terms of Q_0 and the set of terms added by the user at each iteration.

Example 8 Let us suppose that the user query is Q_0 = ResidencePlace. Let us suppose too (for a not too big example) that the search engine finds only four documents mapping this query and that the sets of terms describing these documents are:
d_1: {Hotel, Reunion}, d_2: {Hotel, Carribean}, d_3: {B&B, Sahara}, d_4: {B&B, Corsica}.

The current description of d_1 is then initialized as follows: Cdesc(d_1) = {Hotel, ResidencePlace, Reunion, AlwaysSunny, UnderTheSun, OutOfEurope, IslandWithBeach, PlaceWithBeach, Island, GeoPlace, PhysicalPlace}.

At each iteration, the algorithm realizes four tasks: i) the construction of the active node from the documents mapping the current user query, ii) the construction of its "refined" nodes, (nodes representing refined queries), iii) the identification of potential axes of refinement for each refined node. At this point, the user may decide to iterate the process. He may choose a refined node and an axis of refinement. Then the algorithm realizes the fourth task: iv) the construction of the new current query from the user choice, and goes back to the task i). If the user decides to stop the process, the final refined query is built from the last node chosen by the user.

5.1 Construction of the active node C_i from the set of document D_i

We show here the main steps of task i.

To compute the *Active Node* C_i from the set of documents D_i

1. Compute $C_i = (D_i, T_i)$ with $T_i \leftarrow f_1(D_i)$, the set of terms shared by all current descriptions of documents in D_i
2. Remove the terms shared by all documents from the description of each of them, $\forall d_j \in D_i$, $Cdesc(d_j) \leftarrow Cdesc(d_j) - T_i$

The current descriptions of all the documents in D_i share, at least, the terms belonging to the descriptions and mapping the terms of the query Q_i. This set of shared terms is going to make up the intent T_i of the active node C_i. The extent of C_i is the set of identifiers of the documents D_i. The shared terms are removed from the current description of the documents because they don't allow to discriminate between each of them. That way, we eliminate the redundant elements in T_i as in the Godin's $T' - ICL$.

Example 9 Current descriptions of the four documents of the example share the term $ResidencePlace$ of the query Q_0 and $\{UnderTheSun, GeoPlace, PhysicalPlace\}$ which are common subsumers of the locations appearing in all the documents. The active node is then C_0 = (1234, {ResidencePlace, UnderTheSun, GeoPlace, PhysicalPlace}).

5.2 Construction of the refined nodes C_{ij} of the active node C_i

This step clusters the documents from the active node C_i and creates its child nodes in the lattice. The originality is that using the specialization-generalization hierarchy of terms in the ontology, we obtain a more efficient algorithm than the traditional one.

To compute the *refined nodes* C_{ij} of the node C_i

3. Build the set UT_i of the distinct terms remaining in the current descriptions, $UT_i \leftarrow \bigcup_{j \in [1...n]} Cdesc(d_j)$
4. Compute the *Most General Terms* of UT_i, $Mgt_i \leftarrow Mgt(UT_i)$
5. $\forall t_{ij} \in Mgt_i$, compute the complete pair $C_{ij} = (D_{ij}, T_{ij})$ centered around t_{ij}
6. Keep only the most general nodes among C_{ij}. Their intent does not contain the intent of an other node.

We start by identifying the set of the distinct terms appearing in the documents of the active node C_i. This set, UT_i, is the union of the terms remaining in the current descriptions when shared terms identified in step 1 have been removed.

Example 10 UT_0 = {Hotel, B&B, Reunion, Caribbean, Sahara, Corsica, AlwaysSunny, SunnyOnlySummer, OutOfEurope, InEurope, IslandWithBeach, PlaceWithBeach, Island, NoBeachPlace} and Mgt_0 = {Hotel, B&B, AlwaysSunny, SunnyOnlySummer, OutOfEurope, InEurope, PlaceWithBeach, NoBeachPlace, Island}.

Then, we compute the **most general terms** of this set, Mgt_i, and for each term t_{ij} in Mgt_i, the algorithm constructs the complete pair (D_{ij}, T_{ij}) centered around t_{ij}. We only keep the most general nodes, their intent does not include the intent of an other node. The intent of each most general node is displayed to the user, providing an informative way of summarizing its contents and helping to choose the next node to explore.

Example 11 The complete pair centered around AlwaysSunny is: (123, {AlwaysSunny, OutOfEurope}). It is more general than the one centered around Hotel, which is: (12, {Hotel, AlwaysSunny, OutOfEurope, PlaceWithBeach}). Only the first one is kept. 3 nodes are finally contructed in this step: C_{01}=(34, {B&B}), C_{02}= (123, {AlwaysSunny, OutOfEurope}), C_{03}=(124, {PlaceWithBeach, Island, IslandWithBeach}).

5.3 Identification of axes of refinement Mgt_{ij} for a refined node C_{ij}

During the interactive node construction, as an additional help to the user, we propose some axes of refinement for each node. They are terms the user can choose to include in the future query.

To compute the *potential axes of refinement* Mgt_{ij} of one node C_{ij}

7. In each node $C_{ij} = (D_{ij}, T_{ij})$, remove the terms shared by all documents, $\forall d_k \in D_{ij}, Cdesc(d_k) \leftarrow Cdesc(d_k) - T_{ij}$
8. Build the set UT_{ij} of the distinct terms remaining in the current descriptions, $UT_{ij} \leftarrow \bigcup_{k \in [1...n]} Cdesc(d_k)$
9. Compute $Mgt_{ij} \leftarrow Mgt(UT_{ij})$

The three steps of this task are identical to steps 2, 3 and 4, but applied on each C_{ij}. As in step 2, each node is cleaned: the intent of the node is removed from each document description. Let us remark that a document may appear in different brother nodes. In that case, it has several current descriptions.

Example 12 At step 6, d_1 appears in the two brother nodes C_{02} and C_{03} and its current description is Cdesc(d_1) = {Hotel, Reunion, AlwaysSunny, OutOfEurope, IslandWithBeach, PlaceWithBeach, Island}. After step 7, in the two nodes, the new current descriptions of d_1 are now:
- for the node C_{02} = (123, {AlwaysSunny, OutOfEurope}), Cdesc(d_1) = {Hotel, Reunion, IslandWithBeach, PlaceWithBeach, Island}.
- for the node C_{03} = (124, {PlaceWithBeach, Island, IslandWithBeach}), Cdesc(d_1) = {Hotel, Reunion, AlwaysSunny, OutOfEurope}).

Each new set Mgt_{ij} of the *most General Terms* remaining in the new descriptions of a node C_{ij} is displayed to the user as the set of *refinement axes* of the node. For each term appearing in each Mgt_{ij}, the number of documents mapping this term is displayed too.

Example 13 For the node C_{02} = (123, {AlwaysSunny, OutOfEurope}), Mgt_{02} = {Hotel, B&B, PlaceWithBeach, Island, NoBeachPlace} and the displayed set of refinement axes is {Hotel 2, B&B 1, PlaceWithBeach 2, Island 2, NoBeachPlace 1}.

5.4 Interaction with the user

If the user decides to iterate the process, he chooses both a node and an axis of refinement. A new more precise query is built. The subset of documents with a current description including the chosen axis is identified. Then the algorithm goes back to step 1 in order to propose new refinements of the current query.

To create the *new current query* Q_{i+1} from user choices

10. Given a node C_{ij} and an axis $t_k \in Mgt_{ij}$, $Q_{i+1} \leftarrow T_{ij} \cup t_k$
11. $D_{i+1} \leftarrow \{d \in D_{ij} \mid Q_{i+1} \subseteq Cdesc(d)\}$, the set of documents whose description maps all the terms of the query Q_{i+1}.

Example 14 Let us suppose that the user chooses the node C_{02} = (123, {AlwaysSunny, OutOfEurope}), and the axis of refinement $PlaceWithBeach$. 1 and 2 are the only documents with a description including this term. Then the algorithm goes back to step 1 with these documents and their current descriptions.

If the user decides to stop the refinement process, he selects a node among all the displayed ones. The refined query is then built in a two-step process. First, we compute the union of the intent of the chosen node and of the intent of its parents in the lattice. Second, we compute the desaturated of the set previously obtained.

Example 15 If the user chooses to stop with the node C_{02} = (123, {AlwaysSunny, OutOfEurope}), the refined query Q_1 is {ResidencePlace, AlwaysSunny, OutOfEurope}.Indeed, T_{02} = {AlwaysSunny, OutOfEurope}, T_0 = {ResidencePlace, UnderTheSun, GeoPlace, PhysicalPlace} and $DeSat(T_{02} \cup T_0)$ = {ResidencePlace, AlwaysSunny, OutOfEurope}.

If the user is not satisfied with the proposition of the system, he can backtrack in order to explore alternative nodes. This process allows him to make refinement step by step as he could do in a whole Galois lattice.

6 Experimental Results

We tested the algorithm in the setting of the PICSEL project [10]. We built a repository which groups 80,000 descriptions of documents in the tourism domain. The domain ontology has been composed of a hierarchy of about 250 terms. The documents have been acquired using the search engine Google. Queries to Google were conjunctions of terms, these terms being the most specialized terms in the hierarchy. Answers given by Google were URLs of documents with a match between their description and the terms in the queries. As a same URL may be a result of different queries, the description of the documents in the repository was composed of the union of the terms belonging to all the queries with their URL as results. Moreover, our aim was to acquire documents with the most precise descriptions as possible. So, the repository only stores URLs obtained as results of at least two queries.

Example 16 If URL_1 is a result of both Q_1= {Hotel, Corsica, FiveStars, ViewOnToTheSea, BabyClub} and Q_2= {Hotel, Corsica, SwimmingPool, Golf, ArcheologicalSite}, its description in the repository will be {Hotel, FiveStars, Corsica, ViewOnToTheSea, BabyClub, SwimmingPool, Golf, ArcheologicalSite}.

Once the URLs and their description have been acquired, we built queries in order to interrogate the repository and to evaluate the algorithm. We studied the first level of nodes and analysed OntoRefiner's runtime according to the number of answers. The aim is to help the user to refine a query, step by step, in an acceptable time and to provide him with acceptable interaction conditions. We made several experiments. For each experiment, we noted time, the current OntoRefiner version took, at precise points in the algorithm which were the following:

• At the end of step 4 (cf. section 4.2). This time includes querying the repository, saturation, active node computation, identification of the most general terms. This point in the algorithm will be called further **phase 1**.

• At the end of step 6. This time includes time of phase 1 and computation of the most general nodes. We will call this point **phase 2** in the following.

• At the end of step 9. It includes time of phase 2 and computation of the refinement axes corresponding to each child node. This point will now be referred to as **phase 3** in the following.

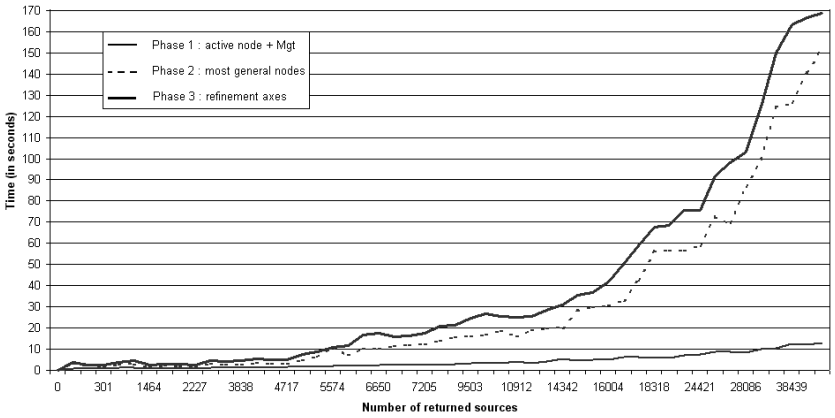

Fig. 8. Time of phase 1 to 3 according to the number of returned answers

We analyzed the experimental results giving by figure 8. This allowed us to specify the conditions where the algorithm is completely satisfactory and applicable as it is. Then we were able to identify the most costly phases and to propose optimizations depending on the number of answers, limited to some specific situations. That way, acceptable interaction conditions are proposed to the user whatever the number of answers is.

The results of the analysis that we have done are the following. First, we noticed that when the number of answers is smaller than 6,000, the algorithm is very efficient. It provides the user with very precise criteria, limiting greatly the number of answers. Time of each iteration is less than 10 seconds. It is very reasonable and quite acceptable for the user.

However, when the number of documents is greater than 6000, we noticed that runtime increases rather considerably. OntoRefiner spends most of the time computing the most general nodes (phase 2 Fig. 8). Consequently, we have an-

alyzed the number and the content of the created nodes, centered around the most general terms (child nodes C_{0i} of the first node C_0), when queries are general and when results are numerous (greater than 6000). Our analysis brought to the fore two significant phenomena.

First, the intent of the first child nodes is usually composed of a unique term. This term is the most general term t_{0i} around which the node C_{0i} is centered. This point is very interesting because, when computing the node centered around a term t_{0i}, $C_{0i} = (D_{0i}, T_{0i})$, the running cost is mainly the result of calculation of T_{0i}, the set of terms shared by all current descriptions of documents in D_{0i}, such that $T_{0i} \leftarrow f_1(D_{0i})$. To make OntoRefiner usable in such a situation, we propose an optimization. The proposed solution is to limit the computation of the child nodes C_{0i} of the first node C_0, by giving only an **approximation of these nodes**. All the documents D_{0i} which are annotated by a most general term t_{0i} will be identified. However, the intent T_{0i} of the associated node will not be precisely computed. We will consider (by approximation according to our heuristic) that it is only composed of t_{0i} as it is most of the time.

Second, we noticed in our experiments that axes of refinement computed for each refined node are often very similar. More precisely, axes of refinement of node C_{0i}, for example, are: the set Mgt_0 associated to its father node C_0, without the term t_{0i} (the term around which the node C_{0i} is centered), but including the specializations of t_{0i} which have no other subsumer in Mgt_0.

Example 17 Given a node C_0, which $Mgt_0 =$ {X, Y, Z, Leasure}, if $ChildLeasure, CulturalLeasure, SportLeasure$ are specializations of $Leasure$ in the domain hierarchy, the axes of refinement computed for the refined node centered around $Leasure$ will be = {X, Y, Z, ChildLeasure, CulturalLeasure, SportLeasure, ...} if for each of these terms there is at least one document with a description including it.

From that observation, we propose to use a second heuristic in order to calculate, once more, only an approximation. This time, the approximation is about the Mgt_{0i} which are associated to the approximated nodes C_{0i}. The approximation of Mgt_{0i} is obtained from the set Mgt_0 associated to its father node C_0, without the term t_{0i} (the term around which the node C_{0i} is centered), but it includes the specializations of t_{0i} which have no other subsumer in Mgt_0. To be sure that the approximated result does not contain terms which are in no document, the approximation process is followed by a verification phase. For each term in approximate Mgt_{0i}, we verify that there is at least one document in D_{0i} with a description including it.

So, to summarize, we studied the balance between runtime and acquisition for the user of new criteria in order to obtain less but more relevant answers. When the number of answers is smaller than 6,000 and leads to a time for phase 3 less than ten seconds, the algorithm is very satisfactory. It provides the user with precise criteria to limit the number of answers.

On the contrary, when the number of answers is greater than 6,000, optimizations techniques have been investigated. Two heuristics are proposed. That way, phases 2 and 3 are less costly while the user is all the same provided with useful and relevant filtering criteria which are approximations.

Thanks to these optimizations, OntoRefiner is now an interactive system, usable whatever the number of retrieved documents is.

7 Conclusion

In this paper, we present OntoRefiner, a system helping the user to refine a query posed onto a repository when the initial returned answers are too numerous. We describe the Galois lattice structure, each node in the lattice being assimilated, in our approach, to a query. We present the domain ontology, the description of the documents in the repository and the way the domain ontology is used to improve the refinement process. The four tasks of the refinement algorithm are presented, each step being detailed. This algorithm has been implemented in Java. It has been experimented to query a repository and also in the setting of the PICSEL mediator [10]. In this paper, we described the experiments coming from refinements of queries posed onto a repository of documents in the tourism domain. We analyzed OntoRefiner's runtime according to the number of answers. The experiments confirmed the feasibility of the approach and its efficiency up to 6,000 documents. To maintain acceptable conditions of user interaction beyond that, we proposed two additional different optimizations which consist to do approximations. The optimizations have been introduced in the approach and the tests give quite satisfactory results.

References

1. A. Bidault, Ch. Froidevaux, B. Safar, Repairing Queries in a Mediator approach. In *Proc. of ECAI'00*, 406-410, Berlin, August 2000.
2. J.P. Bordat, Calcul pratique du treillis de Galois d'une correspondance. In *Mathématiques, Informatiques et Sciences Humaines*, 24, 31-47, 1986.
3. C. Carpineto and G. Romano, Information Retrieval through hybrid navigation of lattice representations. *Inter. J. of Human-Computer Studies*, 45, 553-578, 1996.
4. C. Carpineto and G. Romano, Effective Reformulation of Boolean Queries with Concept lattices. In *Proc. of FQAS'98*, 83-94, 1998.
5. P. Eklund and R. Cole, Structured Ontology and Information Retrieval for Email Search and Discovery. In *Proc. of ISMIS'02*, LNAI 2366, pp.75-84, Lyon, June 2002.
6. R. Godin, R. Missaoui and A. April, Experimental comparison of navigation in a Gallois lattice with conventional information retrieval methods. *Int. Journal of Man-Machine Studies*, 38, 747-767, 1993.
7. R. Godin, R. Missaoui, H. Alaoui, Incremental Concept Formation Algorithms Based on Galois (Concept) Lattices. *Computational Intelligence,*11(2), 246-267, 1995.
8. S. Kuznetsov and S. Obiedkov, Comparing performance of algorithms for generating concept lattices. *Journal of Exp. and Theor. Artif. Intell.*, 14, 189-216, 2002.
9. J. Minker, An overview of Cooperative Answering in Databases. In *Proc. of FQAS'98*, 282-285, 1998.
10. M.-Ch. Rousset, A. Bidault, Ch. Froidevaux, H. Gagliardi, F. Goasdoué, C. Reynaud et B. Safar, Construction de médiateurs pour intégrer des sources d'information multiples et hétérogènes : le projet PICSEL, In *Revue I3,* Vol 2(1), 9-59, 2002.
11. N. Stojanovic, R. Studer, Lj. Stojanovic, An approach for ranking queries in Semantic Web, In *Proc. of ISWC'03*, LNCS 2870, 500-516, 2003.
12. R. Wille, Concept Lattices and Conceptual Knowledge Systems. *Computers and Mathematics with Applications*, 23, 403-515, 1992.