Subontology Extraction Using Hyponym and Hypernym Closure on is-a Directed Acyclic Graphs

Vincent Ranwez, Sylvie Ranwez, Stefan Janaqi

Sub-ontology extraction using hyponym and hypernym closure on is-a directed acyclic graphs

Vincent Ranwez, Sylvie Ranwez, Stefan Janaqi

Abstract— Ontologies are successfully used as semantic guides when navigating through the huge and ever increasing quantity of digital documents. Nevertheless, the size of numerous domain ontologies tends to grow beyond the human capacity to grasp information. This growth is problematic for a lot of key applications that require user interactions such as document annotation or ontology modification/evolution. The problem could be partially overcome by providing users with a sub-ontology focused on their current concepts of interest. A sub-ontology restricted to this sole set of concepts is of limited interest since their relationships can generally not be explicit without adding some of their hyponyms and hypernyms. This paper proposes efficient algorithms to identify these additional key concepts based on the closure of two common graph operators: the *least common-ancestor* and *greatest common descendant*. The resulting method produces ontology excerpts focused on a set of concepts of interest and is fast enough to be used in interactive environments. As an example, we use the resulting program, called OntoFocus (http://www.ontotoolkit.mines-ales.fr/), to restrict, in few seconds, the large Gene Ontology (~30,000 concepts) to a sub-ontology focused on concepts annotating a gene related to breast cancer.

Index Terms— [II. Artificial Intelligence / IV. Knowledge Representation Formalisms and Methods]
[II. Discrete Mathematics / II. Graph Theory]
Keywords—Sub-ontology extraction, Ontology transformation, Directed acyclic graph, Least common ancestor, Greatest common descendant.

1 Introduction

With rapid technological progress, the scope of digital data is overwhelming, ranging from real-time data flows to digitalized documents through specialized databases. Many dedicated applications make use of them, and they often become the core of company decision-making processes. However, their number constitutes a bottleneck for human beings striving to get an overall grasp of information and an impediment to high performance computing. Efficient filtering tools are thus needed to provide the user with the right information at the right time.

Wondering about the Semantic Web, Tim Berners Lee imagined an environment where software agents and humans might cooperate using the same information [1]. Its architecture relies on machine readable semantic models (ontologies) as backbones to structure collections of information and sets of rules to conduct reasoning. This architecture was proposed for the Web, but also appears to be efficient for Information Systems (IS) dealing with large amounts of data [2]. Specialized ontologies endow many Web independent applications with a semantic guide while browsing through documents, filtering and retrieving relevant information or analyzing data in a given context.

However, ontologies, especially those shared and accepted as standards in a given domain, tend to grow beyond the human capacity to manage them (browse, visualize, edit, analyze and share.) This growth is problematic for a lot of applications that require user interactions such as document annotation or ontology modification/evolution. A natural solution to help users would be to provide them with a sub-ontology focused on their current concepts of interest. Of course, as mentioned by [3], individual concepts or terms may always be found by textual searches within the ontology, but links between concepts are tedious to look for. Moreover, given an initial set of concepts of interest, their relationships can generally not be explicit without adding some hyponyms and hypernyms.

In this paper, given a reference-ontology, a "good" sub-ontology is envisaged as the smallest excerpt containing concepts of interest, additional concepts to specify their interconnections and corresponding relationships. We identified (but were not limited to) three main applications:

- Conception/Extension of ontologies: when designing an ontology, it is often necessary to zoom in on a specific part of it. While modifying an ontology, people may consider a very restricted subset of concepts and analyze them through relationships that interconnect them before adding new concepts or new relations or reorganizing them.
- Ontology based information retrieval/filtering: given a set

- *Vincent Ranwez is with the laboratoire de Paléontologie, Phylogénie et Paléobiologie, Institut des Sciences de l'Evolution (UMR 5554 CNRS), Université Montpellier II, CC 064, 34 095 MONTPELLIER Cedex 05, France. E-mail: vincent.ranwez@univ-montp2.fr*
- *Sylvie Ranwez and Stefan Janaqi are with LGI2P/EMA Research Centre, Site EERIE, Parc scientifique G. Besse, 30 035 Nîmes cedex 1, France. E-mail {sylvie.ranwez, stefan.janaqi}@mines-ales.fr*

Manuscript received (insert date of submission if desired).

of concepts that annotate a set of documents, having relationships that link them may improve the user's understanding of the annotations.
- Browsing and display of ontologies and related indexations: human cognitive and perceptive limits underline the crucial need for suitable representation tools. Displaying the overall ontology is generally unconceivable and thus some views of it often have to be built.

Sub-ontology is a pre-requisite for ontology visualization which is essential to favor user interactions with ontology based systems. Indeed, we agree with [4] that "regardless of the ontology, presenting terms in a graphical context makes the relationships of ontology terms clear, provides context for annotations, and makes the examination of large annotation sets feasible."

Some sub-ontology extraction methods have been proposed in the Knowledge Engineering literature. They are discussed in the state of the art presented in section 2. Most of them highlight the importance of the *is-a* relationship that may be considered as the backbone of the ontology. This kind of relationship can be represented as a directed acyclic graph (*dag*). Given an initial set of nodes, this paper proposes a definition of relevant nodes using graph operators. Section 3 reviews useful graph definitions and introduces the notion of *least common ancestor* closure, denoted U_{lca} closure, to determine the searched subset of relevant concepts. Section 4 presents optimized algorithms to compute this closure. Having grasped the set of relevant concepts, Section 5 details the process for inferring relations among them.

The major contribution of this work is to propose a sub-ontology extraction method that significantly reduces the number of concepts in a very short time (an ontology having thirty thousand concepts may be restricted to a hundred concepts in few seconds on a standard desktop computer). It favors different displays corresponding to different views on the reference ontology. It is thus possible to embed it within interactive applications where domain experts may focus on relevant concepts of their specialized domain. Visualization of the corresponding ontology excerpt enables users to better comprehend the underlying semantics of their concepts of interest. This extraction algorithm was developed in a program called OntoFocus that takes the OWL reference ontology and a list of concepts of interest as input. Written in JAVA and using the Jena library, this software may be freely used on-line at the address http://www.ontotoolkit.mines-ales.fr/. An application in the life-science domain is detailed in Section 6. The results are analyzed and discussed and outcomes are presented. Section 7 concludes by recapitulating the approach and discussing its impact.

2 State of the art: different sub-ontology extraction approaches

The need for restricted ontologies, particularly in the Life Science domain, is underlined in [5] where different approaches are compared and their strengths and weaknesses are analyzed with respect to the restriction objectives. For authors, ontology restriction methods often involve a two-step process: identification of the set of concepts of interest and then the restriction phase. However, they argue that none can be applied universally. In what follows, the sub-ontology extraction context is given and some methods are presented, organized according to three main application contexts: 1) conception of an ontology, 2) reuse of existing ontologies, and 3) visualization. We then consider sub-ontology extraction in the light of graph theory and present related works in this domain.

2.1 Pruning and trimming ontologies during their conception

Concerning ontology conception, some studies have been conducted to try to automatically identify *conceptual spaces* from a corpus analysis using natural language processes [6], even during the querying phase [7]. A statistical approach is usually used to determine *potential concepts* based on their frequency in the corpus and relationships among them are created *de novo*. After having automatically collected and organized concepts from the textual corpus, such approaches often rely on a pruning and trimming phase [8, 9]. The trimming step proposed in [8] to reduce the built ontology is closely related to sub-ontology extraction. Yet this step is more a way to clean up *de novo* relationships and remove redundant, useless or too fine-grained potential concepts than a real sub-ontology extraction. One point must be underlined here. Some concepts, identified as belonging to high level ontology or domain ontology, cannot be removed and are always kept in their "sub-ontologies". We do not distinguish such levels in our approach and all nodes are considered in the same manner.

2.2 Sub-ontology extraction to favor their *recycling*

An alternative strategy is to reuse parts of existing ontologies. A Web ontology segmentation was proposed in [10] to extract customized ontologies, specific to given applications, from large ontologies. Unfortunately, by keeping all ascendants and descendants of concepts of interest, the size of the resulting sub-ontology might be very large. This customized ontology may be used alone or to enrich a project-specific ontology. In this latter context, partial ontologies are also called *modules* [11, 12]. When a project related ontology P is enriched with an external module Q, the safety of this enrichment must be ensured. Indeed, if a common label is used for distinct concepts in the two ontologies, these can lead to some erroneous reasoning. More formally, an enrichment of P, using Q, is said to be safe when it "produces exactly the same logical consequences over the vocabulary of Q as Q alone" [11, 12]. This property is not straightforward to check and may result in complex algorithms and high computation times. It should however be noted that *safety* is only related to ontology merging and not to sub-ontology extraction.

2.3 Sub-ontology extraction as visualization aid

Displaying large ontologies is challenging. This problem can be tackled by two complementary approaches: using infor-

mation visualization techniques suited for scalability (e.g. treemap, hyperbolic trees, etc.) and/or using a semantic filtering based on sub-ontology extraction. A complete overview of ontology visualization tools is provided in [13]. Most of them only propose functionalities to zoom in on a local part of the ontology hierarchy rather than displaying an explicit and semantic excerpt of it. Solutions to obtain such excerpts are discussed in what follows.

When focusing on a subset of concepts (e.g. those indexing a given document or those over represented in the index of a set of documents) a graphical representation of their hierarchical relationships is very helpful. The most widespread solution is to display concepts that are in this subset plus those that are ancestors of one of them. This straightforward solution is (manually) used in many publications (e.g. [4, 14]) as well as within Web-based tools[1].

An Australian team proposed a solution for what they call the "user-driven automatic extraction of valid sub-ontology" [15, 16]. The need for sub-ontology extraction is clearly highlighted in [16], where the authors point out the fact that an application focuses only on particular aspects of the whole ontology. For the authors, sub-ontologies are valid independent ontologies, known as materialized ontologies that are specifically extracted to meet certain needs – what they call *tailoring*. However, they concede that *tailoring* is time consuming and thus developed a distributed approach in order to be able to deal with ontologies embedding thousands of concepts. This work was supplemented in [3], where the authors state that traditional visualization techniques fail to achieve semantic representativeness and user friendliness, and propose a methodology for extracting a sub-ontology before applying visualization techniques. Therefore they reinterpret the notion of *views* that is used in the database domain to transform data upon request according to given specifications. For these authors, the resulting sub-ontology should be *valid* and *complete* itself [17]. In the same line, we think that the term "sub-ontology" implies that the restricted set of concepts and embedded relations constitute a *valid* ontology.

For us, sub-ontology extraction is definitely devoted to assisting end-users and enhancing their understanding of the interconnection between concepts. Visualization thus constitutes the main goal of this work.

2.4 From concept of interest selection to sub-ontology extraction

As previously mentioned, sub-ontology extraction starts with the selection of concepts of interest. "In large ontologies, which may contain thousands of elements, […] the selection method is not very efficient, because it requires human intervention and there are too many concepts involved in the process" [5]. In cases where documents are annotated with concepts of the domain ontology, the selection of concepts of interest according to a subset of documents can be automated. One can simply keep any concepts that annotate at least one document of interest. This may result in keeping numerous concepts. Several studies have been carried out to detect concepts that annotate more of these documents than expected by randomly sampling the same number of documents. Such tools are especially useful and widespread in life science applications [18, 19].

Pruning methods are evaluated in [5] according to: their degree of automation, the kind of relationships they consider and the ratio between the number of concepts that they keep and those they trim. Whereas methods presented in [5] only remove a few concepts from the original ontology, our algorithm considerably reduces the size of the ontology in a very short time. This leads to a core sub-ontology focused on end-users' concepts of interest that meet their needs.

Given a reference ontology and a set of concepts of interest, few methods allow the extraction of a corresponding sub-ontology meeting both objectives: finding a concise yet informative restriction (i.e. that can be grasped by a human operator), and being fast enough to be used in an interactive environment. As mentioned in section 2.3, the main sub-ontology extraction study that was designed according to these objectives was conducted by the Australian team of Carlo Wouters, Tharam Dillon, Wenny Rahayu and associates. Even though their objectives for constructing views are very similar to ours, the two solutions differ with respect to various points. In their proposal [3], concepts of the materialized ontology may be a compression (rename) of concepts of interest. They are obtained via optimization schemes, a logical grouping of rules, and actions of the user. Conversely, for us, each concept that appears in the sub-ontology is directly derived from the reference ontology. Moreover, they considered the possibility of extending each sub-ontology independently [17], whereas we propose a rapid solution to allow the creation of sub-ontologies on request. The latter may thus be considered as instantaneous views on the large ontology, which is the sole ontology that can be used for automated reasoning (e.g. during information retrieval) and that can evolve. This avoids redundancy and related maintenance difficulties. Moreover, while their method needs to be parallelized for handling large ontologies, ours can filter such ontologies in few seconds on a standard desktop computer.

Having the set of concepts of interest (or signature), our main objective is to identify hyponyms and hypernyms that must be added to clarify relationships among the remaining concepts and thus to obtain an explicit sub-ontology.

This problem was addressed (for hypernyms) by Nebot and Berlanga [20], who propose three main extraction strategies. The first two (SCA – Signature Common Ancestors, ASA – All Signature Ancestors), as acknowledged by the authors, can result in an excessive amount of irrelevant nodes in the output sub-ontology. The third one (ASA-ST – All Signature Ancsestors Spanning Tree) tries to improve this situation, but its output depends on the arbitrary choice of a tree spanning the initial ontology. See section 5.1 for an illustrative example.

Our solution provides many more concise sub-ontologies

[1] http://www.informatics.jax.org/GOgraphs/OrthoDisease/

than SCA and ASA strategies. Moreover, since it is based on a well defined lca-closure operator, its output does not depend on the arbitrary choice of a spanning tree.

2.5 A new graph based sub-ontology approach

In the following, sub-ontology extraction is tackled by using an algorithmic approach that is based on *is-a* relationships and two common operators: *least common ancestor* (*lca*) and *greatest common descendant* (*gcd*). When the reference-ontology is represented by a *Direct Acyclic Graph* (*dag*), the "explicit" sub-ontology will be a subgraph that is *closed* for the above mentioned operators.

Before detailing our solution, some choices have to be justified. As underlined in the introduction, the *is-a* relationship is of particular interest. It is often considered as the backbone of the ontology – perhaps because it is commonly used by people to organize things around them. This relationship is central to ontologies as it is the sole one that appears in formal ontology definitions ([21] or [22]). Moreover, previous works have been proposed to use the hierarchical domain structure to compute similarity [23] or in information retrieval systems [24, 25]. Within these contexts, adding other relations just adds noise. Therefore we decided to focus on the dag representing *is-a* relationships to identify relevant concepts. Other relationships are considered and added during post-processing, as described in section 5.3.

Basic filtering, which involves keeping ancestors of concepts of interest, may preserve too many concepts since some concepts may have numerous ancestors (e.g. Aspirin has 152 ancestors in UMLS2003AA). The choice of *lca* operator to identify relevant concepts is motivated and validated through end-user evaluations in [26], with the aim of assisting experts who have to choose an ancestor concept to summarize a group of concepts for adjusting indexation granularity. The resulting tool displays the whole ancestor hierarchy but emphasizes *lca* concepts to assist experts in making choices. Our approach goes a step further and filters the ancestor hierarchy to display only ancestral concepts that are least common ancestors of some other preserved concepts. Moreover, unlike existing solutions focused on ancestors, we also preserve greatest common descendent concepts. These two key improvements considerably reduce the number of displayed concepts while preserving enough of them to render relationships among initial concepts. As a counterpart, a straightforward algorithm based on numerous *lca* computations results in high computation times.

Some works have been proposed to efficiently find the *lca* of concepts [27, 28]. However, they often calculate *lca* only for a pair of concepts (and not for a set of them) and do not search for the possibly numerous *lcas* but only for one of them. The algorithm introduced in [28] for multilabeled trees (i.e. where a single term can label more than one node) can easily be adapted for *dag*; but this solution cannot be used for large ontologies due to its time complexity. The next section introduces an efficient algorithm to identify relevant concepts based on *lca*. The key idea is to exploit the *dag* structure so as to compute in a single graph, traversal *lca* of any pair of concepts within the extracted sub-ontology.

In the following section, based on the experience of the graph community, a solution is developed to build sub-ontologies.

3 Least common ancestor operator and its closure

3.1 Preliminary definitions

Restricting a tree to a subset of nodes is a well defined operation used in various computer science fields. Yet its extension to *dags* is not straightforward. This section introduces some basic graph notations and provides two equivalent formal definitions of *lca* closure in *dags*.

Before going any further, and to avoid misunderstandings, it is important to note that in most of the graph literature *dag* edges are oriented from parents toward children, whereas in the ontology literature edges of the *is-a dag* are instead oriented from children toward parents in order to stress the *is-a* semantic. Though the following results hold for any *dag*, we clearly worked in the ontology framework and thus adopt the corresponding edge orientation convention (from children towards parents).

The following definitions are provided to make the paper self-contained. For further definitions on graphs see [29]. The *is-a* relationship graph is a *dag* $G = (V, E)$. The *indegree* $d_G^-(v)$ (*outdegree* $d_G^+(v)$) of a vertex v is the number of edges with head v (tail v). When G contains a directed (v, u)-path, the vertex u is said to be an *ancestor* of v and the vertex v is a *descendant* of u. For a non-empty subset W of V, the subgraph of G whose vertex set is W and whose arc set is the set of arcs of G that have both ends in W is called the subgraph of G induced by W and is denoted $G[W]$.

Using these notations, it is now possible to define the *least common ancestor* operator that is extensively used in our sub-ontology extraction approach. Given a vertex v of the *dag* $G = (V, E)$, the set $A_G(v)$ denotes the subset of ancestors of v in G. The generalization of this definition to a set $S \subseteq V$ of vertices is straightforward, i.e. $A_G(S) = \bigcap_{v \in S} A_G(v)$. For simplicity, we will omit index G from the notations whenever there is no ambiguity.

Definition 3.1.1. ([30]) The *least common ancestors* $lca(S)$ of a vertex subset $S \subseteq \mathrm{V}$ with respect to a *dag* $G = (V, E)$ are vertices $u \in A(S)$, such that $d_H^-(u) = 0$ in the graph $\mathrm{H} = \mathrm{G}[A(S)]$ induced by $A(S)$.

This definition generalizes the widely known definition of *least common ancestor* (see [30-33]) for a couple of vertices, i.e. $lca(\{x, y\}) = lca(x, y)$. It follows immediately from Definition 3.1.1 that $x = lca(x, y)$ if there is a directed (y, x)-path. By extension, we define $lca(x, x) = x$ for all x.

Note that, unless the *dag* G is a tree, the existence of a pair of nodes $x, y \in S$ such that $lca(x, y) = lca(S)$ is not guaranteed. For instance, in the example presented in Fig. 1, $\mathrm{lca}(\{\mathrm{C1}, \mathrm{C2}, \mathrm{C3}\}) = \{\mathrm{A1}, \mathrm{A2}\}$ while $\mathrm{lca}(\mathrm{C1}, \mathrm{C2}) = \{\mathrm{B1}, A2\}$, $\mathrm{lca}(\mathrm{C1}, \mathrm{C3}) = \mathrm{B2}$ and $\mathrm{lca}(\mathrm{C2}, \mathrm{C3}) = \{\mathrm{A1}, \mathrm{B3}\}$.

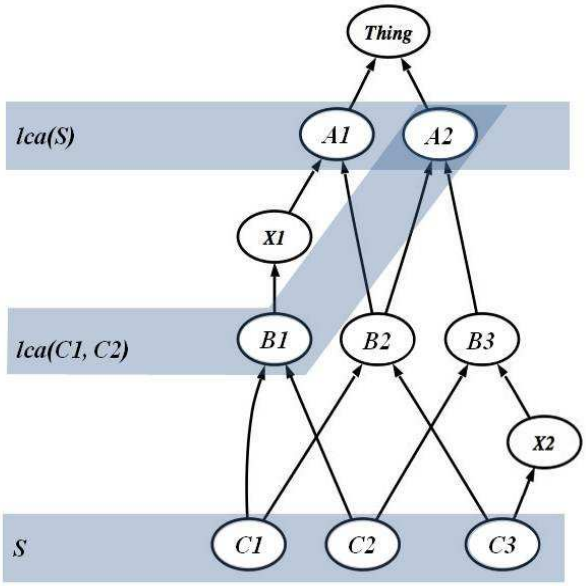

Fig. 1. Hypothetical illustration of the *lca* operator

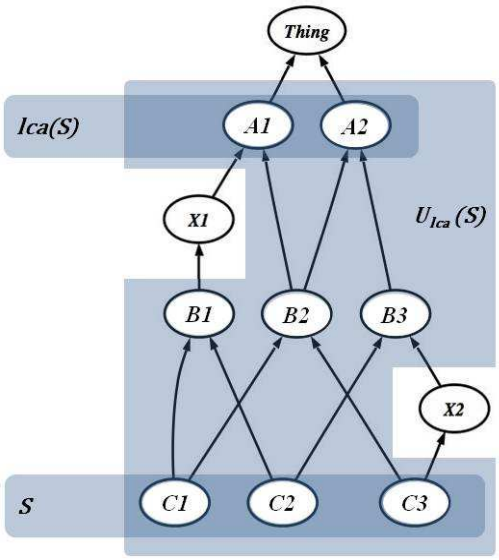

Fig. 2. Hypothetical example of the U_{lca} operator and the way it differs from the *lca* operator.

3.2 U_{lca} operator and two equivalent definitions of its closure in a *dag*

Closure operators are widely used in mathematics, especially in geometry. The best known example concerns convexity in a Euclidian space where a convex polygon can be obtained by giving a *finite* set of points and the segment operation $s(x, y)$ (that could correspond to $lca(x, y)$ operation).
In the previous section, we provided an intuitive and natural definition of the *least common ancestors* of a set of vertices denoted $lca(S)$. Yet, when considering a set of vertices S, every *least common ancestor* of a pair of vertices of S is a key vertex to gain insight into relationships between vertices of S. We thus now introduce a new operator that makes use of the $lca(x, y)$ operator and generalizes it to a well defined closure operator.

Definition 3.2.1. Let S be a subset of vertices of G. The U_{lca} operator on S is defined as:

$$U_{lca}(S) = \bigcup_{x,y \in S} lca(x, y) \quad (1)$$

It follows from this definition that $S \subseteq U_{lca}(S)$ and that the U_{lca} operator is monotonic, i.e.:

$$A \subseteq B \Rightarrow U_{lca}(A) \subseteq U_{lca}(B).$$

Fig. 2 illustrates the definition of the U_{lca} operator. This example also highlights the difference between the two *least common ancestor* operators. Given the set $S = \{C1, C2, C3\}$, $U_{lca}(S) = \{A1, A2, B1, B2, B3, C1, C2, C3\}$, while $lca(S) = \{A1, A2\}$.

In some cases, having only $U_{lca}(S)$ and/or $lca(S)$ is not enough to understand all relationships among concepts of S. This case is depicted in Fig. 3, where concept A is helpful for understanding $C1$ and $C2$ relationships but is neither included in $U_{lca}(S)$ nor in $lca(S)$. Concept A is of interest since A is the *lca* of $B1$ and $B2$, which in turn are *lcas* of two concepts of S. This leads us to the following definition of the U_{lca} closure of S.

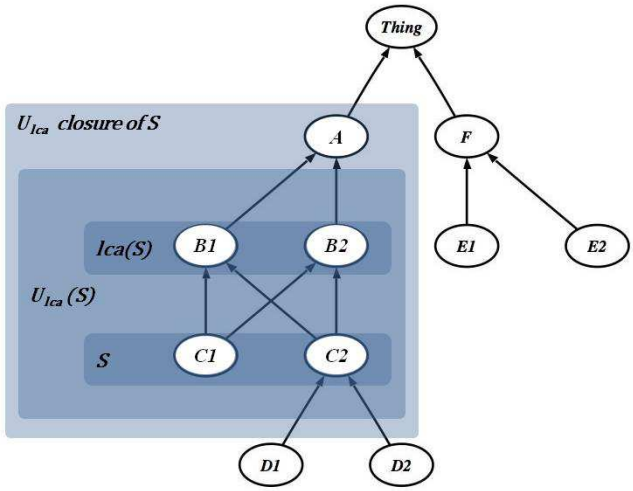

Fig. 3. Hypothetical illustration of U_{lca} closure

Let S be a subset of vertices of G. Let the increasing set sequence defined by: $S_0 = S$ and $S_{k+1} = U_{lca}(S_k), k = 0, 1, \ldots$. As G is finite, there is a number $c, 0 \leq c \leq |V|$ such that $\forall k \geq c, S_k = U_{lca}(S_k)$. This fixpoint (or fixset) is reached because of the monotonicity of the U_{lca} operator. In fact, once this relation holds for a given k, it holds for all greater values. So c and S_c are well defined.

Definition 3.2.2. The number c is called the *closure index* and the set S_c is called the U_{lca} *closure* of S and is denoted $\bar{S}$.

This definition provides a simple iterative algorithm to compute U_{lca} closure. The time complexity of this algorithm is related to the closure index. In the simple case where G is a tree, the closure index cannot be greater than 1. The following lemma shows that this is no longer true in the general case.

Lemma 3.2.1. For a *dag* $G = (V, E)$ and a set $S \subseteq V$, the number of iterations needed to obtain $\bar{S}$ is $O(|V|)$.

Proof. It is clear that S_k increases with at least one vertex at each iteration, hence proving that $c \leq (|V| - |S|)$. On the other hand, as shown by the example below, c can be as large as $(|V| - |S|)/2$. It follows that the number of iterations needed to obtain $\bar{S}$ is $O(|V|)$.

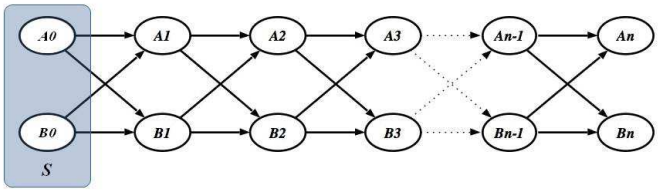

Fig. 4. Hypothetical example where the closure index is proportional to $|V|$. $S_0 = \{A_0, B_0\}$, at iteration k two vertices $\{A_k, B_k\}$ are added so that $S_k = \{A_0, \ldots, A_k, B_0, .. B_k\}$.

There is an alternative (descending) way to define U_{lca} closure in a *dag* $G = (V, E)$. For this, let the family of U_{lca} closed sets containing S be denoted by:

$$\mathcal{L}(S) = \{L \subseteq V \mid S \subseteq L \text{ and } L = U_{lca}(L)\} \quad (2)$$

Lemma 3.2.2. The family $\mathcal{L}(S)$ is non-empty and closed for the intersection, i.e. $M, N \in \mathcal{L}(S) \Rightarrow M \cap N \in \mathcal{L}(S)$.

Proof. $\mathcal{L}(S)$ is non-empty since V obviously belongs to $\mathcal{L}(S)$. Let us now prove that $M, N \in \mathcal{L}(S) \Rightarrow M \cap N \in \mathcal{L}(S)$.

- $S \subseteq M \cap N$. As $S \subseteq M$ and $S \subseteq N$, $S \subseteq M \cap N$.
- $M \cap N = U_{lca}(M \cap N)$.
  - $M \cap N \subseteq U_{lca}(M \cap N)$, by the U_{lca} operator definition.
  - $U_{lca}(M \cap N) \subseteq M \cap N$. This comes from the fact that the U_{lca} operator preserves monotonicity:
    $M \cap N \subseteq M \Rightarrow U_{lca}(M \cap N) \subseteq U_{lca}(M) = M$
    $M \cap N \subseteq N \Rightarrow U_{lca}(M \cap N) \subseteq U_{lca}(N) = N$
    $U_{lca}(M \cap N) \subseteq M \text{ and } U_{lca}(M \cap N) \subseteq N$
    $\Rightarrow U_{lca}(M \cap N) \subseteq M \cap N$. □

Definition 3.2.3. The U_{lca} closure of S is the set $\bar{\bar{S}} = \bigcap_{M \in \mathcal{L}(S)} M$.

Lemma 3.2.3. The above two definitions of U_{lca} closure are equivalent.

Proof. We have to show that $\bar{\bar{S}} = \bar{S}$.

- $\bar{\bar{S}} \subseteq \bar{S}$. By definition, $\bar{S} = U_{lca}(\bar{S})$, furthermore the monotonicity of the set sequence $S_0, S_1 \ldots, \bar{S}$ ensures that $S \subseteq \bar{S}$. Therefore $\bar{S} \in \mathcal{L}(S)$, thus proving that $\bar{\bar{S}} \subseteq \bar{S}$.
- $\bar{S} \subseteq \bar{\bar{S}}$. By definition, set S is included in every set of $\mathcal{L}(S)$ and thus in their intersections. It follows that $S_0 \subseteq \bar{\bar{S}}$, and therefore $U_{lca}(S_0) \subseteq U_{lca}(\bar{\bar{S}})$, which can be rewritten $S_1 \subseteq \bar{\bar{S}}$. Applying the U_{lca} operator to both terms leads to $S_2 \subseteq \bar{\bar{S}}$, $S_3 \subseteq \bar{\bar{S}}$ and so on until $S_c \subseteq \bar{\bar{S}}$, thus proving that $\bar{S} \subseteq \bar{\bar{S}}$. □

These definitions provide the framework for our sub-ontology extraction.

4 An Efficient algorithm to compute U_{LCA} closure

Since ontologies may contain several thousand concepts, efficient algorithms are needed to compute the U_{lca} closure of a set S of concepts of interest with respect to their underlying *is-a dag* $G = (V, E)$.

4.1 Straightforward algorithms to compute U_{lca} closure

We give below, as a lower bound on complexity, two straightforward solutions to calculate U_{lca} closure (the second being a slight improvement of the first, see Algorithm 2).

We denote $s_k = |S_k|$ and estimate the number N_{lca} of calls of the $lca(x, y)$-operator. In Algorithm 1, N_{lca} depends on the closure index c that determines how many times the *while* loop is done. As this number $c = O(|V|)$ (Lemma 3.2.1), the number N_{lca} of calls of the $lca(x, y)$-operator for this algorithm is:

$$N_{lca}(algo1) = \sum_{k=0}^{c} s_k^2 = \sum_{k=0}^{O(|V|)} O(|V|^2) = O(|V|^3) \quad (3)$$

Name: Straightforward_U_{lca}_closure
Input: a *dag* G and a set of nodes S of G.
Result: $\bar{S}$ the U_{lca} closure of S.

```
S_tmp ← S;
do
    S_k ← S_tmp;
    S_tmp ← ∅;
    for each (x, y) ∈ S_k × S_k
        S_tmp ← S_tmp ∪ lca(x, y);
    end
while S_k ≠ S_tmp
return S_k
```

Algorithm 1. Straightforward U_{lca}-closure algorithm

Name: Slightly_improved_U_{lca}_closure
Input: a *dag* G and a set of nodes S of G.
Result: $\bar{S}$ the U_{lca} closure of S.

```
S_k ← S; S_new ← S;
do
    S_tmp ← S_k;
    for each (x, y) ∈ S_k × S_new
        S_tmp ← S_tmp ∪ lca(x, y);
    end
    S_new ← S_tmp − S_k;
    S_k ← S_tmp;
while S_new ≠ ∅
return S_k
```

Algorithm 2. Slightly improved U_{lca}-closure algorithm.

This high complexity is due to the fact that some $lca(x, y)$ are computed several times, for instance those of S are computed c times. This is obviously useless, and Algorithm 2 computes $lca(x, y)$ only when needed, i.e. only if x and/or y

are/is in S_{new}. Yet the *lca* should be computed for any couple of vertices of $\bar{S}$. It follows that:

$$N_{lca}(algo2) = \sum_{x,y\in\bar{S}} 1 = O(|V|^2) \quad (4)$$

The time complexity of this second algorithm is $O(|V|^2)$ times the complexity of the $lca(x,y)$ subroutine.

Several optimized algorithms are available to retrieve *one* vertex $v \in lca(x,y)$. We stress the word "one" because, in our approach we need *all* the vertices of $lca(x,y)$. Most optimized solutions to retrieve *one* vertex $v \in lca(x,y)$ rely on the shortest path algorithm. After preprocessing, done in $(|V|^{\omega})$ with $\omega = 2.688$, a representative *lca* can be obtained in constant time for any couple of nodes (see [30-32]). An improved solution, with $\omega = 2.575$, is described in [34]. As the domain is active, [35] have given an algorithm that calculates *all* $lca(x,y)$ for *all* pairs of vertices with a mean time complexity $O(|V|^3 loglog(|V|))$ and worst time complexity $O(|V|^{3.3399})$.

Recall that we need the $U_{lca} - closure$ of a set S (denoted by $\overline{S}$) and Algorithm 2, even with the optimized subroutine [35] for computing *lca*, would have a complexity of at least $O(|V|^{3.3399})$ for computation time and $O(|V|^2)$ for memory space (in order to store pre-computed lca). We thus introduce an optimized dedicated solution that takes the topological vertex order induced by the *dag* into account. This solution has lower worst time complexity – $O(|\bar{S}||E|)$ – and space – $O(|\bar{S}||V|)$ – complexity. The advantage of this solution is even more relevant in practice, since for most real cases $|\bar{S}| \ll |V|$ and for most ontologies $|E| \ll |V^2|$.

4.2 Optimized algorithm to compute U_{lca} closure in $O(|\bar{S}||E|)$

This subsection details an optimized algorithm that determines the U_{lca} closure of S for a *dag* $G = (V,E)$ in $O(|\bar{S}||E|)$. The key idea of this algorithm is that, although there are $O(|V|^2)$ couples of vertices, at most $O(|V|)$ nodes can be added to S. Rather than computing the *lca* for each pair of vertices, we thus consider each node and decide greedily whether or not it must be added to S. This can be done efficiently by taking the topological vertex order induced by the *dag* into account. These are classical algorithms that we recall below so that the paper will be self-contained.

```
Name: postOrder
Input: a dag G
Result: the list of nodes of G in postOrder
    G.postOrder ← empty list
    for root in G.nodes()
        if root has no parent
            postOrderRec(root)
    end
    return G.postOrder
```

```
Name: postOrderRec
Input: a dag G, a node n of G
Result: add the list of desc(n) in post order to the postOrder list of G
    mark n as visited
    for s in children(n)
        if s has not been visited
            postOrderRec(s)
    end
    append n to the postOrder list of G
```

Algorithm 3. Post order implementation (recursive and iterative subroutines).

Our U_{lca} closure algorithm considers vertices in *post order*, i.e. a vertex is never considered before considering all of its descendants. Indeed, vertices of a *dag* can be ordered along a horizontal line such that all descendants of a vertex are placed to its left. We call this a *post order* since, as shown in [36], one can be efficiently obtained using the post-order indices of a depth-first search (Algorithm 3).

```
Name: U_lca closure
Input: a dag G, a set S containing nodes of interest.
Result: the U_lca closure of S.
    S^AL ← ∅
    P = postOrder(G)
    for n in P
        S^D(n) ← ∅          // S^D(n) is the set of descendants of
                            // n present in the U_lca closure of S
        maxS^D(n) ← 0       // maxS^D(n) is the maximal value
                            // |S^D(s)| with s a child of n
        for s in children(n)
            S^D(n) ← S^D(n) ∪ S^D(s)                          (*)
            maxS^D(n) ← max(|S^D(s)|, maxS^D(n))
        end
        if ( (n ∈ S) OR ( |S^D(n)| > maxS^D(n) ) )            (**)
            S^D(n) ← S^D(n) ∪ {n}
            S^AL ← S^AL ∪ {n}
        end
    end
    return S^AL
```

Algorithm 4. Computation of U_{lca} closure

Proposition 4.2.1. (*Proof of correctness*). Given the inputs $G = (V,E)$ and S, the set S^{AL} returned by Algorithm 4 is the closure of S with respect to G, $S^{AL} = \bar{\bar{S}} = \bar{S}$.

Proof. Let P denote the array of nodes of G sorted by the postOrder function. The U_{lca} closure algorithm goes through $P[1], \ldots, P[k], \ldots, P[|V|]$ gathering, for each k, a subset of S^{AL} denoted $S^{AL}(k)$. It is clear that $S^{AL} = S^{AL}(|V|)$. We show by induction that:

$$S^{AL}(k) = \bar{\bar{S}} \cap P[1..k] \; and$$
$$S^D(P[k]) = \bar{\bar{S}} \cap desc(P[k]), k = 1, 2, \ldots, |V|. \quad (5)$$

The statement (5) holds for $k = 1$.
Node $P[1]$ has no descendant. It is kept in S^{AL} if and only if $P[1] \in S \subseteq \bar{\bar{S}}$ (the line (**) of the algorithm.) Thus, $S^{AL}(1) = \bar{\bar{S}} \cap P[1..1]$ and $S^D(P[1]) = \bar{\bar{S}} \cap desc(P[1])$.

Assuming that (5) holds for indices $1, \dots, (k-1)$, then (1) also holds for indices $1, \dots, k$.
Let $n = P[k]$ be the current node and $\{s_1, \dots, s_p\}$ be the set of immediate descendants (children) of n. Since nodes are considered in post order, all children of n are at the left of n in array P. When the algorithm is considering n, all of its children $s_1, \dots, s_p$ have already been treated and set $S^D(s_i)$ has been recorded for each $s_i \in \{s_1, \dots, s_p\}$. At the point (**), the current recorded set for node n (see point (*)) is:

$$S^D(n) = S^D(s_1) \cup \dots \cup S^D(s_p) \qquad (6)$$

The test at the point (**) of the algorithm is used to decide whether or not n is in the closure and should be added to $S^D(n)$ and to S^{AL}.

- If $n \in$ S, n is added to $S^{AL}(k)$ as well as to $S^D(n)$ and evidently $n \in \bar{\bar{S}} \cap P[1..k]$ and $S^D(P[k]) = \bar{\bar{S}} \cap desc(P[k])$.
- If $n \notin$ S. In this case, by the induction hypothesis, at the point (**) any node of $S^D = S^D(s_1) \cup \dots \cup S^D(s_p)$ are in $\bar{\bar{S}}$ and we have $S^D = S^D(n) - \{n\} = \bar{\bar{S}} \cap desc(n)$. The only thing remaining to prove is that n will be included in $S^{AL}(k)$ and in $S^D(n)$ iff it is the least common ancestor of two nodes of S^D.
  - If $(|S^D(n)| > maxS^D(n))$ then there are at least two nodes z, t of S^D, such that $n = lca(z,t)$ and n should be added to $S^{AL}(k)$ as well as to $S^D(n)$. (see Fig. 5 for an example)
    As $|S^D(n)| > maxS^D(n)$, there are at least two nodes $z, t \in S^D$ such that $\{z,t\} \nsubseteq S^D(s_i)$, $i = 1, \dots, p$. It follows that there are two distinct children s_i, s_j of n such that $z \in S^D(s_i)$, $t \in S^D(s_j)$. By definition of the *lca*, $n \in lca(z,t)$ if and only if, $n \in A(z,t)$ and n has no descendant in the ancestor set $A(z) \cap A(t)$. The former assertion is obvious, let us prove the latter by supposing this is not the case (*reductio ad absurdum*). So, there is a node $n' \in A(z) \cap A(t)$ and a (n, n') directed path in G. This path necessarily goes through a child s_m of n and, by the induction hypothesis, $S^D(s_m) = \bar{\bar{S}} \cap desc(s_m)$. It follows that $\{z,t\} \subseteq S^D(s_m)$, which is impossible.
  - If the test (**) is not true, then n is not in the closure and is added neither to S^D nor to S^{AL}. The main thing to prove is that when $|S^D(n)| = maxS^D(n)$, there are not two nodes z, t of $S^D(n)$ such that $n = lca(z,t)$. (see Fig. 6 for an example). As $|S^D(n)| = maxS^D(n)$ then there is some $i \in \{1, \dots, p\}$ such that $S^D(n) = S^D(s_i)$. In this case, n cannot be the *lca* of a couple of nodes (z,t) because node s_i is (by construction) an ancestor of z, t and a descendant of n. It follows that $n \notin \bar{\bar{S}}$ and the proof is complete. □

Proposition 4.2.2. (*Time complexity of U_{lca} closure algorithm*) For a node set S in a *dag* $G = (V, E)$, the U_{lca} closure algorithm runs in $O(|\bar{S}||E|) = O(|V||E|)$.

Proof. Obtaining the postOrder vector of nodes is done through a classical depth first search traversal of the graph in $O(|E|)$. The complexity of the remaining part of the algorithm, made of two nested **for** loops, is obviously determined by the number of executions of line (*). This line computes the union of two sets of at most $|\bar{S}|$ elements and is executed for every child of every node, i.e. $O(|E|)$ times. It follows that the whole complexity of this algorithm is $O(|\bar{S}||E|)$. In the worst case, $|\bar{S}|$ equals $|V|$ leading to a complexity of $O(|V||E|)$. Note however that generally $|\bar{S}| \ll |V|$. □

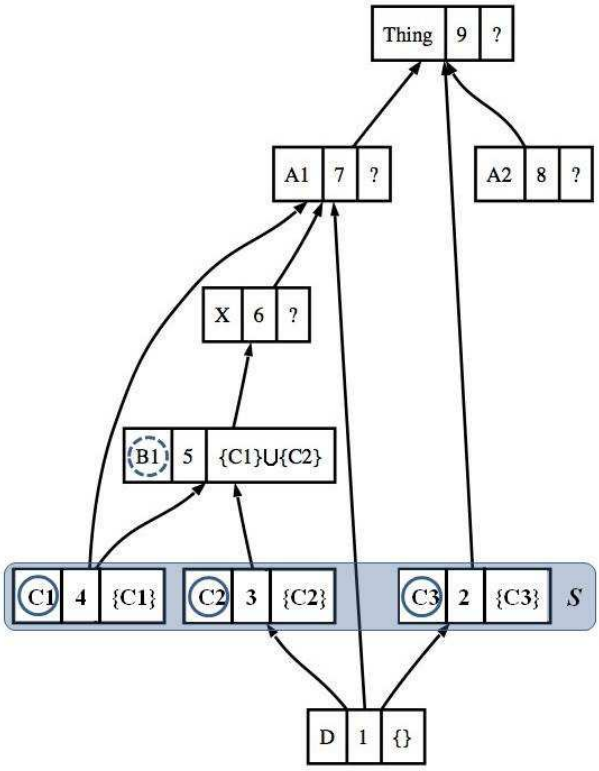

Fig. 5. U_{lca} closure algorithm: considering a node of the U_{lca} closure of S.
For each node n, the three following characteristics are displayed: its label, its rank in the postOrder vector and its current set $S^D(n)$. This figure displays information available at the point (**) while processing the node $B1$ (dotted circle). At this step the four sets $S^D(C1)$, $S^D(C2)$, $S^D(C3)$ and $S^D(D)$ have already been computed. Other S^D sets are not yet initialized (marked with '?'). $C1, C2$ and $C3$ have been identified as part of the U_{lca} closure of S (encircled) and the algorithm considers whether or not $B1$ is also part of this U_{lca} closure. At point (**) the current set $S^D(B1)$ is the union of the two sets $S^D(C1)$ and $S^D(C2)$. This union being larger than the two sets used to deduce it, $B1$ is identified as part of the U_{lca} closure of S and $S^D(B1)$ will be updated accordingly (see fig. 6).

Proposition 4.2.3. (*Space complexity of U_{lca} closure algorithm*) For a node set S in a *dag* $G = (V, E)$, the U_{lca} clo-

sure algorithm requires $O(|\bar{S}||V|) = O(|V|^2)$ memory space.

Proof. For each node of the outer "for" loop of algorithm 4, a subset of $\bar{S}$ is stored. In the worst case, $|\bar{S}| = |V|$ leading to a complexity of $O(|V|^2)$.
Note that in most cases $|\bar{S}| \ll |V|$. Moreover, when all parents of a node n have been treated, the subset of $\bar{S}$ attached to n becomes useless, which means that some memory space can be freed. This can be easily done by maintaining a counter for each node initialized to its number of parents. When treating a node, the counters of all of its children are decreased by one, and when a child reaches a zero value its memory is freed. This does not reduce the worst case complexity, since this optimization is useless when the *dag* is made of one node that has $|V| - 1$ children, but it significantly reduces the memory space needed in most real cases. □

The algorithms presented in this section are used for identifying a set of relevant concepts according to those of interest. Their relationships have to be rebuilt in order to obtain the intended sub-ontology.

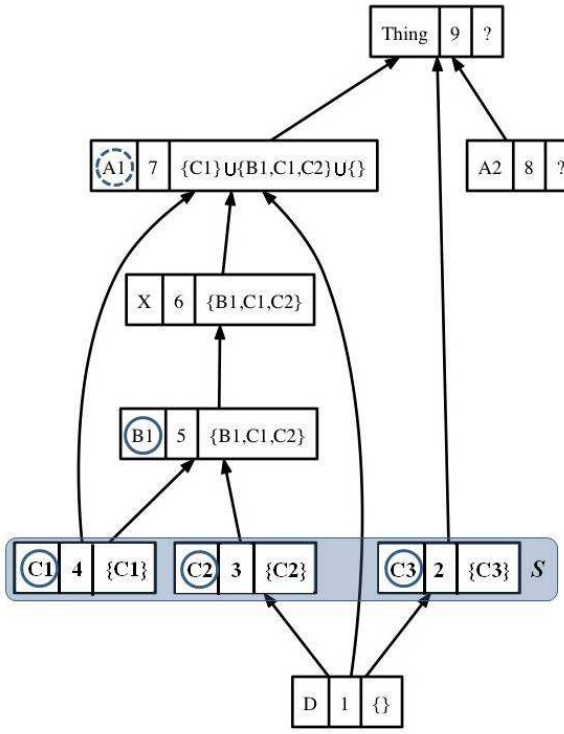

Fig. 6. U_{lca} closure algorithm: considering a node that is not part of the U_{lca} closure of S. (see Fig. 5 for legend).
The algorithm is considering $A1$. At the point (**), the set $S^D(A1)$ combines the three sets $S^D(C1)$, $S^D(X)$ and $S^D(D)$. With the resulting set being equal to $S^D(X)$ and $A1 \notin S$, $A1$ will not be added to the U_{lca}closure of S.

An efficient algorithm has been described to compute the U_{lca} closure of a set of concepts. Yet, when considering a set of vertices S, every greatest common descendant (U_{gcd}) of a pair of vertices of S is also a key vertex to grasp relationships between vertices of S. All results and algorithms of the previous section concerning *lca* can easily be transposed to *gcd*. This is due to the duality of the two operators. Let G be the *dag* (V, E) and let G^* the *dag* $G^* = (V, E^*)$ such that $(\mathrm{v}, \mathrm{u}) \in \mathrm{E}^*$ iff $(\mathrm{u}, \mathrm{v}) \in \mathrm{E}$. Then, for all x and y of V, an *lca* (resp. *gcd*) of x and y in G is a *gcd* (resp. *lca*) of x and y in G^*. $\bar{S}$ denotes the U_{lca} closure of S and, similarly, $\underline{S}$, denotes its U_{gcd} closure.

5 Building the Sub-Ontology

The previous section describes efficient algorithms to compute U_{lca} and U_{gcd} closures of a set of concepts. Both algorithms have been used within a broader process to extract the sub-ontology.

5.1 Using U_{lca} and U_{gcd} closures to define the set of relevant concepts.

Given a set S containing concepts of interest, we define the set S_r of *relevant* concepts kept in the sub-ontology as $S_r = \bar{S} \cup \underline{S}$. Note that this set is not closed with respect to *lca* or *gcd* operators, as illustrated by Fig. 7, where B and D belongs to S_r whereas their *least common ancestor* A1 does not. One could thus be tempted to use the set $\overline{\underline{S}}$ defined by the closure of both operators.

Definition 5.1.1. Let the increasing set sequence defined by: $\mathrm{S}_0 = \mathrm{S}$ and $\mathrm{S}_{k+1} = \overline{\mathrm{S_k}} \cup \underline{\mathrm{S_k}}, \mathrm{k} = 0, 1, \ldots$
Set $\overline{\underline{\mathrm{S}}}$ is equal to set $\mathrm{S_c}$ associated with index c such that $\forall \mathrm{k} \geq \mathrm{c}, \mathrm{S_k} = \overline{\mathrm{S_k}} \cup \underline{\mathrm{S_k}}$.

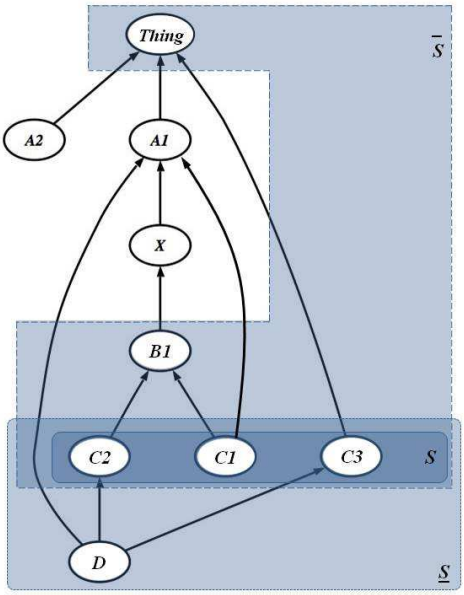

Fig. 7. Hypothetical illustration of $\boldsymbol{S_r = \bar{S} \cup \underline{S}}$

Using this constructive definition, set $\overline{\underline{\mathrm{S}}}$ can be obtained by iterating the U_{lca} closure and U_{gcd} closure algorithms until a fixed point (set) is obtained. This can be done in an $\mathrm{O}\left(|\overline{\underline{\mathrm{S}}}|^2|\mathrm{E}|\right)$ algorithm. Yet the concepts of $\overline{\underline{\mathrm{S}}}$ that are not in $\bar{\mathrm{S}} \cup \underline{\mathrm{S}}$ are obtained by applying both lca and gcd operators. For instance, concept A1 in Fig. 7 is present in $\overline{\underline{\mathrm{S}}}$ because $\mathrm{A1} = \mathrm{lca}(\mathrm{lca}(\mathrm{C1}, \mathrm{C2}), \mathrm{gcd}(\mathrm{C2}, \mathrm{C3}))$. The relationship between such concepts and the initial ones is thus hard to interpret and including them in the sub-ontology can be more confusing than helpful. We thus build the sub-ontology with

respect to S using only concepts within $\bar{S} \cup \underline{S}$ since this is faster and more meaningful than using $\underline{S}$.

Note that the set of concepts derived from the SCA and ASA strategies from Nebot and Berlanga [20], is $\{C1, C2, C3, B, X, A1, Thing\} \supset \bar{S}$. Indeed, though poorly informative, X and A1 are kept because they are common ancestors of C1 and C2. X is discarded by their third strategy (ASA-ST) while the A1 concept may be captured or not, depending on their choice of spanning tree.

5.2 Adding hyponymic and hypernymic relationships among relevant concepts

Once the set of relevant concepts S_r has been found, hypernymic (and hyponymic) relationships among them have to be identified. We want to preserve the hypernymic and hyponymic relationship between concepts in S_r even though intermediary concepts have not been kept in S_r.

Definition 5.2.1. Given the *dag* $G = (V, E)$, the relevant *subdag* $G_r = (V_r, E_r)$ is defined with:

- $V_r = \bar{S} \cup \underline{S}$
- $(u, v) \in E_r$ iff there is a directed path in G going from u to v without crossing any nodes of V_r

The set E_r of edges can be efficiently computed due to the topological order induced by the *dag*. This time we will consider a vertex u only after having considered all of its ascendants. Such an order, that we will call a *pre order*, can be obtained by simply considering the post order vector from tail to head. Let $V_{RRA}(u)$ be the set of *Relevant Reachable Ancestors* of u containing vertices that are present in V_r and can be reached from u through a path crossing no other nodes of V_r. When considering vertices in *pre order,* the set $V_{RRA}(u)$ of the current node u is simply the union of $V_{RRA}(f)$ sets of all of its parent nodes that are not in V_r, plus all its parent nodes that are in V_r. The set E_r is then constructed by adding, for each node u of V_r, edges (u, v) between u and any node v of $V_{RRA}(u)$ as detailed in Algorithm 5.

Name: subIsaDag
Input: a *dag* $G = (V, E)$ a set S containing nodes of interest
Result: G_r the relevant subDAG

```
V_r ← S̄ ∪ S
G_r ← (V_r, ∅)
for each u in reverse(postOrder(G))
    V_RRA(u) ← ∅
    for each f in parents(u)
        if (f ∉ V_r)
            V_RRA(u) ← V_RRA(u) ∪ V_RRA(f) (*)
        else
            V_RRA(u) ← V_RRA(u) ∪ f
    if (u ∈ V_r)
        for each v in V_RRA(u)
            G_r.addEdge(u, v)
return G_r
```

Algorithm 5. *Sub-isa-dag* algorithm

As for Algorithm 4, the key instruction, line (*), computes the union of two sets of at most $|V_r|$ elements and is executed for every parent of every node of the initial *dag* G i.e. $O(|E|)$ times. The overall complexity of this algorithm is thus $O(|\bar{S} \cup \underline{S}||E|) = O(|V||E|)$.

5.3 Adding other kinds of relationships between relevant concepts

While ontologies may include various kinds of concept relationships, most sub-ontology extractions identify relevant concepts only using the *is-a* relationship. Indeed this relationship is the only one shared by all ontologies and it constitutes their backbone. The key role of the *is-a* relationship is clearly explicit in the formal definition of the ontology proposed by [37] (p. 244-). Consequently, our method to select relevant concepts relies only on this relationship. However, other relationships may be added among those selected concepts. This section describes the strategy we used within this second phase.

Having the relevant *sub-isa-dag* $G_r = (V_r, E_r)$ obtained from the original ontology σ, one should at least enrich the relevant sub-ontology σ_r with any relationship uRv of σ concerning two concepts u, v of V_r.

Some additional relationships between two concepts u, v of V_r can be deduced from the original ontology even if the relation uRv is not explicitly present in σ. For instance, let Isa, Eq and Pof be the *is-a*, *equivalence* and *part-of* relationships, respectively. If we consider the case where $A\ Isa\ B$, $A\ Pof\ C$ and $A\ Eq\ D$, and $V_r = \{D, C\}$, the relationship $D\ Pof\ C$ can be added since $A\ Pof\ C$ and $A\ Eq\ D$.

Formalizing the relationship that should be kept is far from obvious and the semantics of the relationships at stake must be properly taken into account. Yet some general rules can be applied for the simplest cases. In particular, for any transitive relationship R (i.e. when $aRb\ and\ bRc \Rightarrow aRc$), we can add the relation uRv to σ_r for any pairs of concepts such that there is a sequence of relationships $uRa_1, a_1Ra_2, \ldots, a_nRv$ in σ and no a_i concept belongs to V_r. This is a straightforward generalization of the approach used to determine the *is-a* relationship of σ_r.

Algorithm 5, formulated to add the *is-a* relationship to σ_r, relies on the fact that the underlying graph representation of these relationships is a *dag*. Hence, this algorithm can only handle anti-symmetric relationships (that induce a *dag* structure) and is only meaningful for transitive relationships (otherwise it will induce wrong relationships). It can thus be used on *is-a* and *part-of* relationships but not for the equivalence one that is transitive but not anti-symmetric and may thus induce cycle ($A\ Eq\ B$, $B\ Eq\ C$ and $C\ Eq\ A$.)

All of the presented algorithms have been implemented in an application called OntoFocus using Java and the Jena library. OntoFocus takes an OWL ontology and a set of concepts of interest S as input, and an OWL sub-ontology focused on S as output. The current version adds any direct relationships existing between two relevant concepts and uses Algorithm 5 to add direct and induced *is-a* and *part-of* relationships between them. Applications are discussed in the following section.

6 APPLICATIONS, RESULTS AND PERSPECTIVES

The accumulation of digital information has become problematic in many domains, especially in Life Sciences where genome decryption, for example, leads to hundreds of databases and interacting applications. They often make use of ontologies to organize the information, to support semantic information retrieval and visual navigation through it. The size and scope of bio-ontologies have increased almost as rapidly as the biological data. However, in numerous cases, only a few of their concepts are useful at once. For example, molecular biologists often visualize and analyze a graph representation of the ontology relations of a subset of Gene Ontology concepts, but unfortunately this analysis is often done by hand [38-40].

OntoFocus has been used to restrict the Gene Ontology to the subset of concepts related to the BRCA1 gene associated with BReast CAncer susceptibility. More precisely, we downloaded the annotation of the Swiss-Prot protein P38398 (associated with BRCA1) via the Go Annotation (GOA) service provided by the European Bioinformatics Institute (http://www.ebi.ac.uk/GOA/.) We thus obtained 102 redundant annotations corresponding to a set of 50 individual GO concepts. The corresponding sub-ontology inferred by OntoFocus in about one minute contains 92 relevant concepts (instead of thirty thousand).

As illustrated in Fig. 8, the visualization is very comfortable and within human cognitive and perceptive limits. Here two of the three connected parts of the sub-ontology are presented: concepts that correspond to *molecular functions* (GO_0003674) and those that correspond to *cellular components* (GO_0005575). The last part corresponding to *biological processes* is not presented here. However, it contains 63 concepts, which also facilitates visualization.

Giving a specific view on the ontology may highlight part of it and clarify the fact that some concepts are refinements of others. Fig. 8 highlights, for example, that several annotations are refinements of the *protein binding function* (GO_0005515.) Such user-centered sub-ontologies may be useful for biological users exploiting annotations, especially for collaboratively updating gene annotations. This approach is also widespread in comparative genomic analysis in which gene function variation among species (gene annotations, in this case, are species dependant) is studied. An intuitive way to summarize this variation is to color concepts of an ontology excerpt. Such representations have been achieved on a large scale for genes related to human diseases [41] and made available on a Web site (http://www.informatics.jax.org/GOgraphs/OrthoDisease/). These excerpts have been obtained by keeping all ancestor concepts (hence including useless concepts while ignoring common descendants). This rough strategy can thus be advantageously replaced by the one presented in this paper and integrated in OntoFocus.

The same approach may be used to simultaneously apprehend annotations of several genes that share some biological characteristics (e.g. genes having similar expression profiles in microarray experiments.) In such cases, GO concepts are useful for giving biological meaning to the corresponding gene clusters. These annotating genes can be used to automatically define the set of concepts of interest that can serve as an OntoFocus input. The resulting sub-ontology highlights gene annotation relationships and helps to determine the cluster semantic. The additional concepts proposed by OntoFocus are also good candidates for annotating a cluster as a whole. Indeed, as underlined by [26], *least common ancestor* concepts are of particular interest for summarizing annotations.

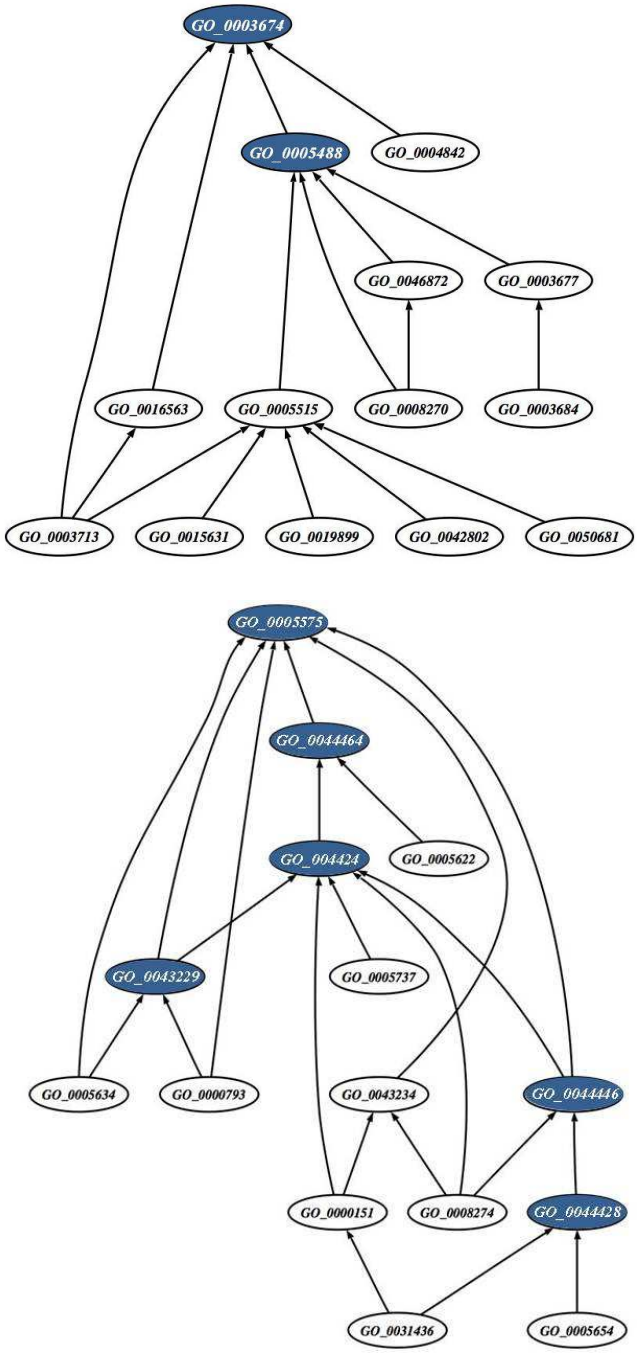

Fig. 8. Visualization of molecular function (GO_0003674) and cellular component (GO_0005575). GO-sub-ontologies constructed by OntoFocus using BRCA1 annotation (blue colored concepts were added by OntoFocus to make the sub-ontology explicit).

Our approach, illustrated here by a biological application, may be used in every Information System dealing with amounts of data and huge ontologies. Indeed, having sub-

ontologies may be helpful in any computer assisted ontology operation requiring a human expert (e.g. ontology design and evolution or visual filtering within conceptual maps).

To be effective, visual representations must be semantically guided so that the user may observe semantics while browsing data. Therefore, we plan to adapt sub-ontology extraction in order to enrich the sub-ontology with additional information such as the number of children/parents in the reference ontology and the original path length of an *is-a* edge. This additional information will allow us to compute the semantic distance separating two relevant concepts with respect to the reference ontology [42].

Concerning ontology visualizations, as mentioned in [13] many papers have been proposed but they all focus on hierarchy representation. "In general the main drawback is that the semantics entailed by the ontologies is not used fully" [3]. We assume that there is an alternative way of representing ontologies. In [43] a semantic distance has been proposed to evaluate similarity between concepts. This distance, combined with visualization techniques using multi-dimensional scaling, provides a different view of the concept hierarchy. Concerning the prospects of the work presented in this paper, we plan to insert OntoFocus in an environment that we have developed: OBIRS (ontology based information retrieval system) [44]. In response to a query built with concepts of a domain ontology, OBIRS builds semantic maps to display the retrieved documents. These maps may be used for navigating through, analyzing or viewing digital documents annotated by concepts of a given ontology.

7 Concluding Remarks

After justifying the crucial need for sub-ontology extraction methods for different purposes, this paper introduces a graph based definition of relevant concepts for highlighting relationships among concepts of interest and provides an optimized algorithm to identify them. According to a reference ontology and a set of concepts of interest, it computes U_{lca} and U_{gcd} closures on direct acyclic graphs (dag) induced by *is-a* relationships. It then infers relationships among these relevant concepts to provide a user-centered sub-ontology. The complexity of this algorithm is proportional to the number of kept concepts in addition to the number of reference dag edges.

The resulting OntoFocus program may be freely used online (http://www.ontotoolkit.mines-ales.fr/). Many applications may benefit from this algorithm. One of them, developed in this paper, concerns the life science domain and the use of gene ontology to analyze gene annotations.
Because of the very short calculation time (few seconds for about 30.000 concepts), calculations may be performed each time a user needs a specific view. This ensures that the sub-ontology is always compliant with the reference ontology and offers novel prospects for real-time user interactions. For instance, this provides the possibility of chaining several extractions, progressively refining the ontology so that the user can precisely tune the filtering process.

Future directions of our work include ontology visualization, organization of query results and dedicated applications coupled with Gene Ontology.

Acknowledgments

We would like to thank Pierre Jean for his help in deploying the OntoFocus Web server.

The authors wish to thank the reviewers for their many detailed comments and proposals to improve the paper.
This publication is the result of a collaboration between the "Institut des Sciences de l'Evolution de Montpellier" (UMR 5554 - CNRS) and the "LGI2P Research Centre from Ecole des Mines d'Alès". This work was supported by the French Agence Nationale de la Recherche 'Domaines Emergents' [ANR-08-EMER-011 'PhylAriane'] and is the contribution N°2011-067 of ISEM.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, pp. 29-37, 2001.

[2] A. Abecker and L. van Elst, "Ontologies for Knowledge Management," in *Handbook on Ontologies second edition*, *International handbooks on information systems*, S. S. a. R. S. (Eds.), Ed. Heidelberg: Springer, 2009, pp. 713-734.

[3] C. Wouters, T. Dillon, W. Rahayu, and E. Chang, "Large scale ontology visualisation using ontology extraction," *International Journal of Web and Grid Services*, vol. 1, pp. 113-135(23), 2005.

[4] M. E. Dolan and J. A. Blake, "Using ontology visualization to understand annotations and reason about them," presented at KR-MED 2006 "Biomedical Ontology in Action", Baltimore, Maryland, USA, 2006.

[5] J. W. Kim, J. Conesa Caralt, and J. K. Hilliard, "Pruning Bio-Ontologies," presented at 40th Annual Hawaii International Conference on System Sciences (HICSS'07), Big Island, Hawaii 2007.

[6] H. C. Chen, B. Schatz, T. Ng, J. Martinez, A. Kirchhoff, and C. T. Lin, "A parallel computing approach to creating engineering concept spaces for semantic retrieval: The Illinois Digital Library Initiative project," *Ieee Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, pp. 771-782, 1996.

[7] P. C. Smits and A. Friis-Christensen, "Resource Discovery in a European Spatial Data Infrastructure," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, pp. 85-95, 2007.

[8] R. Navigli, "Automatically extending, pruning and trimming general purpose ontologies," presented at IEEE International Conference on Systems, Man and Cybernetics Tunisy, 2002.

[9] M. Sabou, "Learning web service ontologies: an automatic extraction method and its evaluation," in *Ontology learning from text: methods, evaluation and applications*, vol. 123, *Frontiers in Artificial Intelligence and Applications*, P. Buitelaar, P. Cimiano, and B. Magnini, Eds. Amsterdam: IOS Press, 2005.

[10] J. Seidenberg and A. Rector, "Web ontology segmentation: analysis, classification and use," in *Proceedings of the 15th international conference on World Wide Web*. Edinburgh, Scotland: ACM, 2006.

[11] B. C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler, "Mod-

ular reuse of ontologies: Theory and practice," *Journal of Artificial Intelligence Research*, vol. 31, pp. 273-318, 2008.

[12] E. Jimenez-Ruiz, B. C. Grau, U. Sattler, T. Schneider, and R. Berlanga, "Safe and economic re-use of ontologies: A logic-based methodology and tool support," *Semantic Web: Research and Applications, Proceedings*, vol. 5021, pp. 185-199, 897, 2008.

[13] A. Katifori, C. Halatsis, G. Lepouras, C. Vassilakis, and E. Giannopoulou, "Ontology visualization methods - A survey," *Acm Computing Surveys*, vol. 39, pp. 10, 2007.

[14] M. E. Dolan and J. A. Blake, "Using Ontology Visualization to Coordinate Cross-species Functional Annotation for Human Disease Genes," in *Computer-Based Medical Systems, IEEE Symposium on*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 583-587.

[15] M. Bhatt, W. Rahayu, S. P. Soni, and C. Wouters, "OntoMove: A knowledge based framework for semantic requirement profiling and resource acquisition," presented at 2007 Australian Software Engineering Conference, ASWEC'07, 2007.

[16] M. Bhatt, A. Flahive, C. Wouters, W. Rahayu, D. Taniar, and T. Dillon, "A distributed approach to sub-ontology extraction," *18th International Conference on Advanced Information Networking and Applications, Vol 1 (Long Papers), Proceedings*, pp. 636-641, 2004.

[17] A. Flahive, W. Rahayu, D. Taniar, B. O. Apduhan, C. Wouters, and T. Dillon, "A service oriented architecture for extracting and extending sub-ontologies in the semantic grid," *21st International Conference on Advanced Networking and Applications, Proceedings*, pp. 831-838, 2007.

[18] S. Bauer, S. Grossmann, M. Vingron, and P. N. Robinson, "Ontologizer 2.0 - a multifunctional tool for GO term enrichment analysis and data exploration," *Bioinformatics*, vol. 24, pp. 1650-1651, 2008.

[19] S. Grossmann, S. Bauer, P. N. Robinson, and M. Vingron, "Improved detection of overrepresentation of Gene-Ontology annotations with parentchild analysis," *Bioinformatics*, vol. 23, pp. 3024-3031, 2007.

[20] V. Nebot and R. Berlanga, "Efficient retrieval of ontology fragments using an interval labeling scheme," *Information Sciences*, vol. 179, pp. 4151-4173, 2009.

[21] C. Ben Necib and J. C. Freytag, "Ontology based query processing in database management systems," *On the Move to Meaningful Internet Systems 2003: Coopis, Doa, and Odbase*, vol. 2888, pp. 839-857, 2003.

[22] N. Guarino, D. Oberle, and S. Staab, "What is an Ontology?," in *Handbook on ontologies*, *International handbooks on information systems*, S. a. S. Staab, Rudi, Ed. Heidelberg: Springer, 2009, pp. 1-17.

[23] P. Ganesan, H. Garcia-Molina, and J. Widom, "Exploiting hierarchical domain structure to compute similarity," *Acm Transactions on Information Systems*, vol. 21, pp. 64-93, 2003.

[24] M. Baziz, M. Boughanem, G. Pasi, and H. Prade, "An Information Retrieval Driven by Ontology: from Query to Document Expansion," presented at 8th International Conference RIAO 2007, Carnegie Mellon University, Pittsburgh, PA, USA, , 2007.

[25] L. Khan, D. McLeod, and E. Hovy, "Retrieval effectiveness of an ontology-based model for information selection," *The VLDB Journal The International Journal on Very Large Data Bases*, vol. 13, pp. 71-85, 2004.

[26] Y. Tao, E. A. Mendonca, and Y. A. Lussier, "A tool for abstracting relevant classes of concepts: The common ancestry summarizer," *Medinfo 2004: Proceedings of the 11th World Congress on Medical Informatics, Pt 1 and 2*, vol. 107, pp. 449-453, 1497, 2004.

[27] Z. Liu and Y. Chen, "Identifying meaningful return information for XML keyword search," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. Beijing, China: ACM, 2007.

[28] Y. Xu and Y. Papakonstantinou, "Efficient keyword search for smallest LCAs in XML databases," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. Baltimore, Maryland: ACM, 2005.

[29] J. A. Bondy and U. S. R. Murty, *Graph Theory with Appplications*. New York: American Elsevier Co., 1976.

[30] M. A. Bender, G. Pemmasani, S. Skiena, and P. Sumazin, "Finding least common ancestors in directed acyclic graphs," *Proceedings of the Twelfth Annual Acm-Siam Symposium on Discrete Algorithms*, pp. 845-854, 2001.

[31] R. Yuster, "All-pairs disjoint paths from a common ancestor in O (n(omega)) time," *Theoretical Computer Science*, vol. 396, pp. 145-150, 2008.

[32] D. Harel and R. E. Tarjan, "Fast Algorithms for Finding Nearest Common Ancestors," *Siam Journal on Computing*, vol. 13, pp. 338-355, 1984.

[33] B. Schieber and U. Vishkin, "On Finding Lowest Common Ancestors - Simplification and Parallelization," *Siam Journal on Computing*, vol. 17, pp. 1253-1262, 1988.

[34] A. Czumaj, M. Kowaluk, and A. Lingas, "Faster algorithms for finding lowest common ancestors in directed acyclic graphs," *Theory of Computer Science* vol. 380, pp. 37-46, 2007.

[35] S. Eckhardt, A. M. Mühling, and J. Nowak, "Fast lowest common ancestor computations in dags," in *Proceedings of the 15th annual European conference on Algorithms*. Eilat, Israel: Springer-Verlag, 2007.

[36] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, "Graph Algorithms," in *Introduction to Algorithms*, 6 ed. Cambridge: The MIT Press, 1992.

[37] A. D. Maedche, *Ontology Learning for the Semantic Web*, vol. 16. Boston ; Dordrecht ; London: Kluwer Academic, 2002.

[38] J. Lindberg, E. af Klint, A. I. Catrina, P. Nilsson, L. Klareskog, A. K. Ulfgren, and J. Lundeberg, "Effect of infliximab on mRNA expression profiles in synovial tissue of rheumatoid arthritis patients," *Arthritis Res Ther*, vol. 8, pp. R179, 2006.

[39] A. Alexa, J. Rahnenfuhrer, and T. Lengauer, "Improved scoring of functional groups from gene expression data by decorrelating GO graph structure," *Bioinformatics*, vol. 22, pp. 1600-7, 2006.

[40] A. del Pozo, F. Pazos, and A. Valencia, "Defining functional distances over gene ontology," *BMC Bioinformatics*, vol.

9, pp. 50, 2008.

[41] K. P. O'Brien, I. Westerlund, and E. L. Sonnhammer, "OrthoDisease: a database of human disease orthologs," *Hum Mutat*, vol. 24, pp. 112-9, 2004.

[42] W. N. Lee, N. Shah, K. Sundlass, and M. Musen, "Comparison of ontology-based semantic-similarity measures," *AMIA Annu Symp Proc*, pp. 384-8, 2008.

[43] S. Ranwez, V. Ranwez, J. Villerd, and M. Crampes, "Ontological distance measures for information visualisation on conceptual maps," *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, Pt 2, Proceedings*, vol. 4278, pp. 1050-1061, 2006.

[44] S. Ranwez, V. Ranwez, M.-F. Sy, J. Montmain, and M. Crampes, "User Centered and Ontology Based Information Retrieval System for Life Sciences," presented at Semantic Web Applications and Tools for Life Sciences, SWAT4LS, Berlin, Germany, 2010.

Vincent Ranwez is a research member of the ISEM laboratory team at the University Montpellier II (France). He holds a PhD in Computer Science from the University of Montpellier (France). He previously worked for Clinigetics SA, a biopharmaceutical firm that used DNA ship analyses to identify potential gene targets for new cardiovascular drugs. This work extensively relied on Gene Ontology based annotation. His main research now focuses on molecular evolution, especially phylogeny and supertree inferences. Since phylogenies are represented as trees or as DAG (in case of hybridizations) most of his work is related to graph algorithms.

Sylvie Ranwez is a research member of the LGI2P Research Center team at the Ecole des Mines d'Alès (France). She holds a PhD in Computer Science from the University of Montpellier (France), dealing with automatic composition of adaptive hypermedia documents based on ontologies and intentional user requests. Her research focuses on technical assistance in indexation, navigation and information search using semantic support (ontologies) and visualization. She has published research papers on semantic distance, conceptual maps, ontological and FCA based indexing and visual navigation.

Stefan Janaqi is a research member of the LGI2P Research Center team of the Ecole des Mines d'Alès (France). He holds a PhD in Computer Science from University Joseph Fourier, Grenoble (France), dealing with geometric properties of graphs. His research focuses on mathematical models for optimization, image treatement, evolutionary algorithms and convexity in discrete structures such as graphs. He has published research papers in differential evolution and matching on vision. He applies optimization models to blending in the petroleum industry.