Extraction de sous-ontologies autonomes par fermeture des opérateurs hyponymie et hyperonymie

Vincent Ranwez, Sylvie Ranwez, Stefan Janaqi

Extraction de sous-ontologies autonomes par fermeture des opérateurs hyponymie et hyperonymie

Vincent Ranwez
Laboratoire de Paléontologie,
Phylogénie et Paléobiologie,
Institut des Sciences de l'Evolution
(UMR 5554 CNRS)
Université Montpellier II, CC 064,
F-34 095 MONTPELLIER Cedex 05,
+33 (0)467 143 697
vincent.ranwez@univ-montp2.fr

Sylvie Ranwez
LGI2P – EMA/Site EERIE,
Parc scientifique G. Besse,
F-30 035 Nîmes cedex 1
+33 (0)466 387 044
sylvie.ranwez@ema.fr

Stefan Janaqi
LGI2P – EMA/Site EERIE,
Parc scientifique G. Besse,
F-30 035 Nîmes cedex 1
+33 (0)466 387 005
Stefan.janaqi@ema.fr

ABSTRACT

Facing the exponentially increasing amount of data, new challenges consist in retrieving the good information at the right time, organizing and filtering data, visualizing them and using them in a specific decision context. For a decade, ontologies have been successfully used as semantic guides for these tasks. Nevertheless the size of ontologies that are shared and accepted as standards in a given domain may rapidly grow beyond the human capacity to grasp information. Dealing with large ontologies proved to be problematic for applications that require user interactions such as ontology-based document annotation or ontology modification/evolution. This problem can be partially overcome by providing the user with a sub-ontology focused on his/her task.

Focusing on "is-a" relationships, an ontology can be represented as a direct acyclic graph. Given a reference-ontology and a set of user's concepts of interest, this paper proposes a formal definition of relevant concepts. This set of relevant concepts is made of user's concepts of interest plus some of their hyponyms and hyperonyms. Those extra concepts are added for enlighten user's concepts relationships in the resulting sub-ontology to make it self-explanatory. The set of those relevant concepts is defined using the closure of classical graph operators i.e.: least common ancestor (*lca*) and greatest common descendant (*gcd*). Efficient algorithms are also provided to identify relevant concepts and to extract the corresponding self-explanatory sub-ontology. The resulting program, called OntoFocus, may be freely used at the address http://www.ontotoolkit.mines-ales.fr/.

As an example of application OntoFocus has been used to restrict the "Gene Ontology" (about 30.000 concepts) to the subset of 50 concepts related to the annotation of the BRCA1 gene (associated with breath cancer susceptibility). The resulting sub-ontology contains only 92 concepts and was obtained in about one minute. It may be useful for biological users, either for exploiting the BRCA1 annotation or for updating this annotation.

JFO 2009 December 3-4, 2009, Poitiers, France

Categories and Subject Descriptors

I.2.4 [**Knowledge Representation Formalisms and Methods**]
G.2.2 [**Graph Theory**]

General Terms

Algorithms, Performance, Theory.

Keywords

Sub-ontology extraction, ontology transformation, ontologies contextuelles, directed acyclic graph, least common ancestor, greatest common descendant.

1. INTRODUCTION

Avec l'essor des nouvelles technologies, les informations nous submergent. Qu'il s'agisse de flots de données en temps réel, de documents numérisés, ou de données contenues dans des bases spécialisées, elles sont au cœur des systèmes d'informations de nombreuses entreprises et sont souvent utilisées comme support au processus d'aide à la décision. Cependant, leur trop grand nombre empêche l'opérateur humain d'avoir une vision globale de l'information disponible et freine les performances logicielles. Des outils de filtrage sont nécessaires afin de fournir l'information pertinente au moment propice.

Lorsqu'il a imaginé le Web Sémantique, Tim Berners Lee a envisagé un environnement où les agents logiciels et humains pouvaient collaborer en partageant les mêmes informations [Berners-Lee et al. 2001]. Son architecture repose sur des modèles sémantiques (ontologies) interprétables par une machine et utilisables pour structurer l'information et permettre des raisonnements via un ensemble de règles. Cette architecture, initiée pour le Web, peut être appliquée à tout système d'information qui traite un grand nombre de données. Ainsi des applications indépendantes du Web et fortement spécialisées se sont, elles aussi, dotées d'ontologies du domaine pour servir de guide sémantique lors de la navigation dans la collection de documents, ainsi que pour améliorer les requêtes et trouver l'information pertinente ou encore pour analyser des données dans un contexte spécifique.

Cependant, ces ontologies, et plus particulièrement celles acceptées comme standard et partagées par de nombreux spécialistes d'un domaine, deviennent de plus en plus volumineuses. Le nombre de concepts qu'elles contiennent dépasse largement la capacité d'un opérateur humain à les appréhender dans leur globalité et à les manipuler (navigation, visualisation, édition, analyse

et partage). Cette croissance constitue une limite à nombre d'applications qui nécessitent une intervention humaine, comme l'annotation de documents ou encore l'évolution d'ontologies (enrichissement). Une solution naturelle consisterait à fournir à cet opérateur une sous-ontologie focalisée sur ses concepts d'intérêt, dans son contexte applicatif. Bien sûr, comme le propose [Wouters et al. 2005], il est toujours possible de rechercher et sélectionner les concepts ou les termes d'intérêt dans l'ontologie. Mais les liens entre ces concepts peuvent être difficiles à identifier. De plus, il n'est généralement pas suffisant de se limiter aux seuls concepts d'intérêt initiaux et il est préférable d'y adjoindre certains de leurs hyponymes ou hyperonymes pour mieux comprendre leur signification et les relations qui les unissent.

Nous avons identifié trois cas d'application dans lequel il est souhaitable de restreindre l'ontologie :

- Conception/extension de l'ontologie. Lors de la mise à jour de l'ontologie, il est souvent nécessaire de considérer un sous-ensemble de concepts et d'analyser leurs relations avant d'ajouter un nouveau concept, de modifier une relation ou de réorganiser les concepts.

- Recherche d'information et filtrage basés sur une ontologie. Etant donné l'ensemble des concepts utilisés pour annoter un (ou plusieurs) document(s), l'identification des relations entre ces concepts permet de mieux comprendre le sens de l'annotation.

- Navigation et visualisation supportées par une ontologie. Les limites cognitives et perceptives des opérateurs humains engendrent une forte demande en outils de représentation intuitifs et conviviaux.

La littérature du domaine de l'ingénierie des connaissances propose certaines méthodes d'extraction de sous-ontologies. Elles sont discutées dans l'état de l'art présenté dans la section 2. La plupart d'entre-elles insistent sur l'importance de la relation "*est-un*". L'ensemble des relations "*est-un*" d'une ontologie peut être représenté par un graphe orienté acyclique (direct acyclic graph – *dag*). Cet article s'appuie sur la théorie des graphes pour proposer une méthode qui détermine, à partir d'un ensemble de concepts d'intérêt, l'ensemble des concepts pertinents du *dag* qui doivent être conservés. La section 3 reprend certaines définitions utiles et introduit la notion de fermeture sur l'opérateur "plus petit ancêtre commun" (notée *lca* pour *least common ancestor*). Cette LCA-fermeture est ensuite utilisée pour déterminer l'ensemble des concepts pertinents à conserver. La section 4 présente des algorithmes optimisés qui calculent la LCA-fermeture. Une fois l'ensemble des concepts pertinents identifiés, la section 5 détaille le processus permettant d'inférer les relations qui les lient. Ces algorithmes ont été implémentés dans une application nommée OntoFocus qui prend en entrée une ontologie de référence, notée σ par la suite, (écrite en OWL) et une liste de concepts d'intérêt et fournit comme résultat une sous-ontologie σ_p pertinente (elle aussi écrite en OWL). Ce programme est écrit en Java et utilise la bibliothèque Jena. Il est possible de l'utiliser à l'adresse suivante : http://www.ontotoolkit.mines-ales.fr/. La section 6 détaille une application de ce programme dans le domaine des sciences du vivant. Les résultats obtenus sont ensuite analysés et discutés avant de présenter les perspectives ouvertes par ces travaux. La section 7 conclue en récapitulant l'approche et en discutant de son impact.

2. ETAT DE L'ART : DIFFERENTES APPROCHES DE L'EXTRACTION DE SOUS-ONTOLOGIES

Avant de décrire notre approche, il est nécessaire de définir précisément la notion de sous-ontologie. Cette section s'y attache et récapitule certains processus qui permettent d'inférer des sous-ontologies. Elle clarifie le vocabulaire utilisé et présente différents travaux publiés dans le domaine de l'informatique et plus particulièrement en ingénierie des connaissances.

Trois principaux besoins peuvent nécessiter l'extraction de sous-ontologies : *i)* pendant la conception d'une ontologie, il est souvent nécessaire de zoomer sur une partie spécifique de l'ontologie ; *ii)* lorsqu'on utilise l'ontologie comme filtre pour la recherche d'information ou le filtrage de données, l'utilisation d'une ontologie restreinte peut améliorer la pertinence des résultats ; et *iii)* la visualisation de l'ontologie dans sa globalité est souvent impossible et il est nécessaire de construire des vues spécifiques sur un sous-ensemble de concepts.

Concernant le premier point, la conception d'ontologies, les approches décrites dans la littérature proposent généralement de collecter et organiser automatiquement les concepts en utilisant des techniques de traitement de langage naturel. Les ontologies ainsi construites sont alors retaillées et élaguées pour obtenir des ontologies plus pertinentes [Navigli 2002; Sabou 2005]. Plus précisément, une approche statistique basée sur la fréquence d'apparition de termes dans le corpus est souvent utilisée pour identifier les concepts potentiels. Les relations sont ensuite construites manuellement. L'étape d'élagage proposée dans [Navigli 2002] pour réduire l'ontologie ainsi construite se rapproche de l'extraction de sous-ontologie. Toutefois la méthode proposée vise plus à 'nettoyer' l'ontologie et à supprimer des concepts redondants, non pertinents ou trop spécifiques ainsi que les relations associées, plutôt qu'à réellement extraire une sous-ontologie dédiée. Dans leur approche l'ontologie est définie sur trois niveaux et certains concepts qui appartiennent à l'ontologie de haut-niveau ne peuvent pas être supprimés. Il est à noter que dans notre approche, nous ne faisons aucune distinction entre les concepts et ils sont tous considérés de la même manière.

Concernant le filtrage et la recherche d'information (point *ii*), Dans [Kim et al. 2007] la nécessité de disposer de techniques d'élagage est soulignée, particulièrement dans le domaine des sciences du vivant, où les ontologies sont souvent volumineuses. Ses auteurs considèrent deux étapes successives : l'identification des concepts d'intérêt puis l'étape d'élagage. Pour eux, dans les ontologies volumineuses qui contiennent des milliers de concepts, la sélection des concepts d'intérêt n'est pas très efficace car, elle nécessite l'intervention humaine et le nombre de concepts considérés est trop important. Dans le cas où des documents sont annotés avec les concepts de l'ontologie du domaine, la sélection des concepts d'intérêt à partir d'un sous-ensemble de documents pertinents peut être automatisée. On peut simplement conserver chaque concept qui annote au moins un document. Mais dans ce cas, il se peut que le nombre de concepts conservés soit très grand. Plusieurs études ont été menées pour identifier les concepts surreprésentés dans les annotations d'un ensemble de documents donné par comparaison avec les annotations d'un même nombre de documents tirés aléatoirement. Ces techniques sont particulièrement répandues dans les applications biologiques [Bauer et al. 2008; Grossmann et al. 2007]. Une grande partie de [Kim, Conesa Caralt and Hilliard 2007] est consacrée à l'évaluation de différentes méthodes d'élagage en fonction de plusieurs critères : leur degré

d'automatisation, les types de relations qu'elles prennent en compte et le ratio entre le nombre de concepts sélectionnés et le nombre de concepts supprimés. Alors que dans les méthodes présentées, on constate que peu de concepts sont supprimés de l'ontologie de référence, il faut souligner que notre algorithme permet de réduire considérablement le nombre de concepts en un temps très court (de l'ordre de la minute) et de fournir ainsi à l'utilisateur une sous-ontologie précisément centrée sur ses concepts d'intérêt.

Les travaux de recherche qui se rapprochent le plus de nos objectifs sont ceux menés par les membres de l'équipe "E-Commerce & Web Technology" de l'université australienne de La Trobe. Ils ciblent les points *ii* et *iii* et proposent une solution pour ce qu'ils appellent l' "extraction automatique de sous-ontologies valides, dirigée par l'utilisateur" [Bhatt et al. 2007]. Le besoin d'extraction de sous-ontologie est clairement identifié dans [Bhatt et al. 2004], et les auteurs soulignent le fait que les applications sont souvent focalisées sur une partie restreinte de l'ontologie, sur certains points de vue. Pour eux, la sous-ontologie extraite, qu'ils nomment *ontologie matérialisée*, est *valide* si elle est indépendante de l'ontologie initiale et correspond exactement aux centres d'intérêt de l'utilisateur. Ils la qualifient de sous-ontologie *sur mesure*. Cependant, les auteurs insistent sur les temps de calculs important que représente une telle restriction sur des ontologies qui peuvent contenir des milliers de concepts. En réponse, ils proposent une approche répartie. Dans [Wouters, Dillon, Rahayu and Chang 2005] ce travail est complété et utilisé en amont de techniques de visualisation. En effet, pour proposer des affichages d'ontologies conviviaux et porteurs de sens, il est indispensable de limiter le nombre de concepts qui sont présentés, afin de ne pas surcharger l'écran. Les auteurs proposent une méthode d'extraction de sous-ontologies dans laquelle ils s'inspirent fortement de la notion de vues utilisée dans le domaine des bases de données. Dans ce cas, les données sont transformées, à la demande, en fonction de certaines spécifications. Les auteurs insistent sur le fait que la sous-ontologie obtenue doit être *valide* et *complète* [Flahive et al. 2007]. La notion de validité est sous-entendue, pour nous, dans celle de sous-ontologie. La définition qu'ils donnent de "complète" est assez subjective, aussi nous lui préférons le qualificatif "*autonome*". L'objectif est d'obtenir une *sous-ontologie* qui puisse elle-même être considérée comme une ontologie indépendante. Des hyponymes et des hyperonymes sont rajoutés aux concepts d'intérêt, de façon à obtenir une sous-ontologie dans laquelle on perde le moins de signification possible, tant au niveau des concepts conservés que de leurs relations.

Même si les approches qui viennent d'être discutées sont voisines de la nôtre, elles diffèrent cependant sur plusieurs points. Dans les travaux présentés, les concepts de l'ontologie matérialisée (la sous-ontologie), peuvent être constitués d'un agrégat de concepts et être renommés. Ils sont obtenus à partir de schémas d'optimisation (un ensemble logique de règles) et d'actions de l'utilisateur. A l'opposé, pour nous, il est important que chaque concept présent dans la sous-ontologie soit à l'identique de ce qu'il était dans l'ontologie de référence. Ce faisant, l'utilisateur n'est pas déstabilisé par de nouveaux concepts qui apparaissent. Pour réaliser l'extraction, nous voulons que le processus soit entièrement automatique. De plus, les auteurs de [Flahive, Rahayu, Taniar, Apduhan, Wouters and Dillon 2007] envisagent que les sous-ontologies puissent ensuite évoluer séparément. Or il nous semble primordial, pour éviter les problèmes de redondance et de mise à jour, de pouvoir créer "à la volée" une sous-ontologie chaque fois que c'est nécessaire, mais de conserver l'ontologie de référence comme seule… référence ! Pour cela nous avons besoin d'un programme capable de créer très rapidement ces sous-ontologies. Ces dernières peuvent être considérées comme des vues sur l'ontologie de référence. Alors que leur méthode nécessite une programmation répartie pour traiter de larges ontologies, notre méthode peut filtrer plusieurs milliers de concepts en moins d'une minute sur un poste de travail standard.

Etant donné une ontologie de référence, une "bonne" sous-ontologie peut être définie comme le plus petit extrait qui contienne tous les concepts d'intérêt et qui constitue une ontologie autonome. Dans la section suivante, nous développons notre solution en nous basant sur des notions issues de la théorie des graphes.

3. L'OPERATEUR DE PLUS PETITS ANCETRES COMMUNS ET SA FERMETURE

Dans cette section nous détaillons comment il est possible de définir les concepts pertinents pour la création d'une sous-ontologie dédiée en s'appuyant sur la relation "*est-un*" et sur les deux opérateurs usuels que sont le plus petit ancêtre commun (*lca*) et le plus grand descendant commun (*gcd*). Si l'on se restreint aux relations "*est-un*" de l'ontologie de référence, cette dernière peut alors être représentée par un graphe orienté acyclique, ou *dag*. On peut alors définir de manière formelle une sous-ontologie autonome comme un sous-graphe fermé pour les deux opérateurs cités plus haut.

3.1 Définitions préliminaires

La restriction d'un arbre à un sous-ensemble de ses sommets est une opération clairement définie et utilisée dans de nombreux domaines de l'informatique. Cependant cette définition ne se généralise pas de manière immédiate au cas des *dag*. Cette section introduit quelques notations de base de la théorie des graphes et fournit deux définitions équivalentes pour la fermeture de l'opérateur *lca* (*lca-fermeture*) dans le cas d'un *dag*.

Avant d'aller plus loin dans cette section, il est important pour éviter tout malentendu, de noter que dans la littérature liée à la théorie des graphes les arcs sont généralement orientés des pères vers les fils tandis que dans la littérature liée aux ontologies les arcs du *dag* "*est-un*" sont généralement orientées du fils vers le père pour mettre en avant la sémantique du lien "*est un*". Bien que les résultats de cette section soient valides pour n'importe quel *dag*, ce travail se place clairement dans le cadre des ontologies. Nous orienterons donc nos arcs en conséquence (i.e. du fils vers le père).

Les définitions fournies dans cette section sont celles nécessaires à la compréhension de cet article. Pour plus d'information sur la théorie des graphes nous renvoyons le lecteur à l'un des ouvrages de référence de ce domaine tel que [Bondy and Murty 1976]. Le graphe représentant les liens "*est un*" qui existent entre concepts est un *dag* $G = (V, E)$[1]. Le *degré entrant* $d_G^-(v)$ (*degré sortant* $d_G^+(v)$) d'un sommet v est le nombre d'arcs pointant sur v (sortant de v). Si G contient un chemin orienté (v, u), on dit que u est un *ancêtre* de v et que v est un *descendant* de u. Si W est un sous-ensemble non vide de V, le sous-graphe de G induit par W

[1] Nous conservons la notation usuelle G=(V, E) issue de la terminologie anglophone : *Vertices* (sommets) et *Edges* (arcs).

(noté $G[W]$) a pour sommets les éléments de W et pour arcs ceux de G dont les deux extrémités sont dans W.

En nous appuyant sur ces notations, il est maintenant possible de définir les notions *d'ancêtre* et de *plus petit ancêtre commun* qui sont centrales dans notre approche d'extraction de sous-ontologie. Etant donné un sommet v d'un *dag* $G = (V, E)$, l'ensemble $A_G(v)$ désigne le sous-ensemble de V constitué des ancêtres de v dans G. Cette définition se généralise facilement au cas d'un ensemble $S \subseteq V$ de sommets, i.e. $A_G(S) = \bigcap_{v \in S} A_G(v)$. Pour simplifier ces notations, nous omettrons l'indice G chaque fois qu'il n'y aura pas d'ambiguïté sur l'identité du graphe utilisé.

Définition 3.1.1. Les plus petits ancêtres communs, d'un sous-ensemble de sommets $S \subseteq \mathrm{V}$ par rapport à un *dag* $G = (V, E)$ donnent un ensemble de sommets $u \in A(S)$, tels que $d_H^+(u) = 0$ dans le graphe $\mathrm{H} = \mathrm{G}[A(S)]$ induit par $A(S)$. On notera cet ensemble $lca(S)$ en référence à la terminologie anglophone usuelle de *least common ancestors*.

Cette définition généralise celle couramment admise du plus petit ancêtre commun pour un couple de sommets, i.e. $lca(\{x, y\}) = lca(x, y)$. Elle s'appuie sur les degrés sortants d'un sous-graphe de G tout comme la définition proposée pour un couple de sommets dans [Bender et al. 2001; Yuster 2008] (une définition équivalente à cette dernière peut être obtenue sur la base d'un ordre partiel des ensembles de sommets [Aitkaci et al. 1989]).

Il découle directement de la Définition 3.1.1. que $x = lca(x, x)$ pour tout x et que $x = lca(x, y)$ dès qu'il existe un chemin orienté (x, y). On peut également noter que, hormis le cas où le *dag* G est un arbre, rien ne garantit l'existence d'une paire de sommets $x, y \in S$ telle que $lca(x, y) = lca(S)$. Par exemple, dans le cas de la Figure 1, on a $\mathrm{lca}(\{\mathrm{C1}, \mathrm{C2}, \mathrm{C3}\}) = \{\mathrm{A1}, \mathrm{A2}\}$ alors que $\mathrm{lca}(\mathrm{C1}, \mathrm{C2}) = \mathrm{B1}$, $\mathrm{lca}(\mathrm{C1}, \mathrm{C3}) = \mathrm{B2}$ et $\mathrm{lca}(\mathrm{C2}, \mathrm{C3}) = \mathrm{B3}$.

3.2 Deux définitions équivalentes de la fermeture de l'opérateur LCA

La définition précédente des plus petits ancêtres communs d'un ensemble de sommets est intuitive et naturelle. Cependant, lorsque l'on considère un ensemble de sommets S, les plus petits ancêtres communs de chaque paire de sommets de S sont tous des sommets utiles pour comprendre au mieux les relations entre les sommets de S. Dans cette section nous introduisons donc un nouvel opérateur, basé sur l'opérateur $lca(x, y)$, ainsi que la fermeture de ce nouvel opérateur.

Définition 3.2.1. Soit S un sous-ensemble de sommets de G. L'opérateur LCA sur S est défini comme :

$$LCA(S) = \bigcup_{x,y \in S} lca(x, y)$$

Il découle de cette définition que $S \subseteq LCA(S)$ et que l'opérateur LCA est monotone, i.e. :

$$A \subseteq B \Rightarrow LCA(A) \subseteq LCA(B).$$

L'exemple suivant illustre également la différence entre les deux opérateurs liés à la notion de plus petits ancêtres communs. Etant donné l'ensemble $S = \{C1, C2, C3\}$, $LCA(S) = \{B1, B2, B3, C1, C2, C3\}$, tandis que $lca(S) = \{A1, A2\}$.

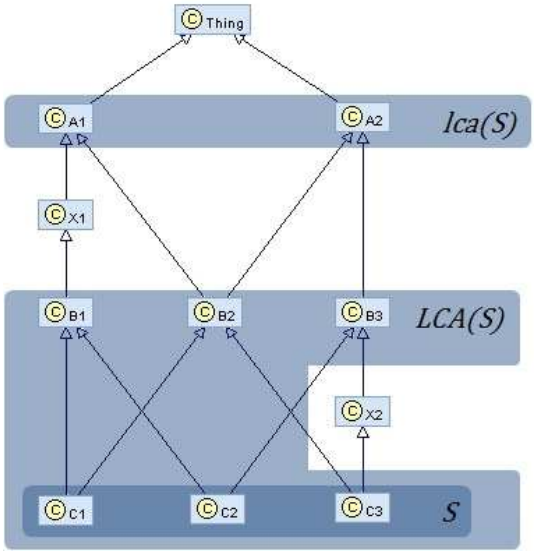

Figure 1: Un exemple[2] simple des opérateurs *lca* et *LCA*

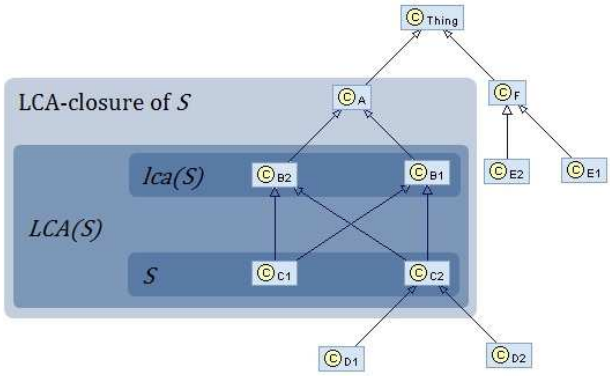

Figure 2 : Un exemple de fermeture pour l'opérateur LCA

Dans certains cas, le fait de prendre en compte les concepts de $LCA(S)$ et/ou $lca(S)$ n'est pas suffisant pour pleinement comprendre les relations qui existent entre les concepts de S. C'est par exemple le cas dans l'exemple de la Figure 2, où le concept A permet de mieux comprendre les relations entre les concepts $C1$ et $C2$ de S et cela bien que A n'appartienne ni à $LCA(S)$ ni à $lca(S)$. Le concept A est intéressant car il est le *lca* de $B1$ et $B2$, deux concepts qui eux sont inclus dans $lca(S)$. Cette remarque souligne l'intérêt de LCA-fermeture de S que nous définissons comme suit :

Définition 3.2.2. Soit S un sous ensemble des sommets de G. Soit la séquence croissante définie de la manière suivante : $S_0 = S$ et $S_{k+1} = LCA(S_k), k = 0, 1, \ldots$ L'ensemble S_c, associé à *l'indice de fermeture c* (*closure index)* tel que $\forall k \geq c, S_k = LCA(S_k)$, est appelé *LCA-fermeture* de S et il est noté $\bar{S}$.

[2] Les figures représentant des ontologies ont été créées en utilisant le logiciel RDF-Gravity (version 1.0) http://semweb.salzburgresearch.at/apps/rdf-gravity/

Cette séquence ne peut pas croitre indéfiniment, car G est un graphe fini. Il existe donc une valeur de k telle que $S_k = LCA(S_k)$. La monotonie de l'opérateur LCA garantit que si cette égalité est vraie pour une valeur de k donnée, elle le restera pour toutes les valeurs qui lui sont supérieures. Ce qui prouve que c (et donc $\bar{S}$) est bien défini de manière univoque.

Cette définition fournit un algorithme simple pour calculer la LCA-fermeture. La complexité en temps de cet algorithme dépend de la valeur de l'indice de fermeture. Dans le cas simple où G est un arbre, cet indice ne peut pas être plus grand que 1. Le lemme suivant montre que ce n'est malheureusement plus vrai dans le cas général.

Lemme 3.2.1. Pour un *dag* $G = (V, E)$ et un ensemble $S \subseteq V$, l'indice de fermeture c est inférieur ou égal à $(|V| - |S|)$ et peut valoir jusqu'à $\frac{1}{2}(|V| - |S|)$.

Preuve. Par définition $S_0 = S$ et, tant que k est inférieur à c, S_k contient au moins un sommet de plus à chaque itération, ce qui prouve que $c \leq (|V| - |S|)$. Par ailleurs, la figure ci-dessous présente un exemple où c vaut $\frac{1}{2}(|V| - |S|)$. Le nombre d'itérations nécessaire pour obtenir $\bar{S}$ est donc en $O(|V|)$.

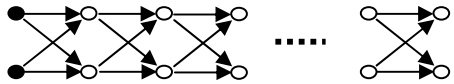

Figure 3 : Si S contient les deux sommets noirs les plus à gauche alors, à chaque itération l'ensemble S_k contient deux sommets en plus (ceux de la colonne suivante).

Il existe une formulation alternative pour définir, cette fois de manière descendante, la LCA-fermeture dans un *dag* $G = (V, E)$. Pour cela, on considère la famille $\mathcal{L}(S)$ des sur-ensembles de S fermés pour l'opérateur LCA :

$$\mathcal{L}(S) = \{L \subseteq V \mid S \subseteq L \text{ et } L = LCA(L)\}$$

Lemme 3.2.2. La famille $\mathcal{L}(S)$ est non-vide et fermée pour l'intersection, i.e. $M, N \in \mathcal{L}(S) \Rightarrow M \cap N \in \mathcal{L}(S)$.

Preuve. $\mathcal{L}(S)$ est non vide du fait que V appartient toujours à $\mathcal{L}(S)$. Il reste à prouver que $M, N \in \mathcal{L}(S) \Rightarrow M \cap N \in \mathcal{L}(S)$.

- $S \subseteq M \cap N$. Comme $S \subseteq M$ et $S \subseteq N$, $S \subseteq M \cap N$.
- $M \cap N = LCA(M \cap N)$.
  - $M \cap N \subseteq LCA(M \cap N)$.
    D'après la définition de l'opérateur LCA.
  - $LCA(M \cap N) \subseteq M \cap N$.
    Du fait que l'opérateur LCA est monotone :
    $M \cap N \subseteq M \Rightarrow LCA(M \cap N) \subseteq LCA(M) = M$
    $M \cap N \subseteq N \Rightarrow LCA(M \cap N) \subseteq LCA(N) = N$
    $\mathrm{LCA(M \cap N)} \subseteq \mathrm{M}$ and $\mathrm{LCA(M \cap N)} \subseteq \mathrm{N}$
    $\Rightarrow \mathrm{LCA(M \cap N)} \subseteq \mathrm{M \cap N}$. □

Définition 3.2.3. La LCA-fermeture de l'ensemble S est $\bar{\bar{S}} = \bigcap_{M \in \mathcal{L}(S)} M$.

Lemme 3.2.3. Les deux définitions de la LCA-fermeture sont équivalentes.

Preuve. Il faut prouver que $\bar{\bar{S}} = \bar{S}$.

- $\bar{\bar{S}} \subseteq \bar{S}$. Par définition, $\bar{S} = LCA(\bar{S})$, de plus la monotonie de la séquence $S_0, S_1 \ldots, \bar{S}$ garantit que $S \subseteq \bar{S}$. Il en découle que $\bar{S} \in \mathcal{L}(S)$, ce qui prouve que $\bar{\bar{S}} \subseteq \bar{S}$.
- $\bar{S} \subseteq \bar{\bar{S}}$. Par définition, l'ensemble S est inclus dans chaque ensemble de $\mathcal{L}(S)$, et donc dans leur intersection. On a $S_0 \subseteq \bar{\bar{S}}$, et par monotonie $LCA(S_0) \subseteq LCA(\bar{\bar{S}})$, ce que l'on peut réécrire de la manière suivante $S_1 \subseteq \bar{\bar{S}}$. Si l'on applique une nouvelle fois l'opérateur LCA aux deux termes de cette inclusion, on obtient $S_2 \subseteq \bar{\bar{S}}$ puis $S_3 \subseteq \bar{\bar{S}}$ et ainsi de suite jusqu'à $S_c \subseteq \bar{\bar{S}}$, ce qui prouve que $\bar{S} \subseteq \bar{\bar{S}}$. □

Ces définitions de la LCA-fermeture constituent le cadre théorique de notre méthode d'extraction de sous-ontologies autonomes.

4. UN ALGORITHME EFFICACE POUR CALCULER LA LCA-FERMETURE

Les ontologies peuvent contenir plusieurs milliers de concepts. Il est donc crucial d'avoir des algorithmes efficaces si l'on veut pouvoir calculer la LCA-fermeture d'un ensemble de concepts d'intérêt dans le *dag* "*est-un*" sous-jacent. Cette section commence par une description d'une solution naïve. Même si cette solution peut être légèrement améliorée, sa complexité en temps reste élevée conduisant à des temps de calculs prohibitifs. Nous décrivons ensuite une version optimisée qui s'appuie sur l'ordre des sommets induit par la topologie du *dag* $G = (V, E)$. La complexité en temps de ce nouvel algorithme est en $O(|\bar{S}||E|)$.

4.1 Algorithme naïf pour le calcul de la LCA-fermeture

Notons $s_k = |S_k|$ et N_{lca} le nombre d'appels de l'opérateur $lca(x, y)$. Dans la version la plus naïve, N_{lca} dépend de la valeur de l'indice de fermeture c qui détermine le nombre de passages dans la boucle *while*. Comme cet indice c est de l'ordre de $O(|V|)$ (Lemme 3.2.1), la complexité de l'algorithme est :

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

Cette complexité rédhibitoire est en grande partie due au fait que le calcul de $lca(x, y)$ est fait plusieurs fois pour un même couple de sommets. Par exemple, ce calcul est fait c fois pour tous les couples de sommets de S. Il est évidemment inutile de refaire ces calculs à chaque fois et la version moins naïve de l'algorithme ne calcule le lca de deux sommets (x, y) que lorsque cela est nécessaire, i.e. seulement lorsque x et/ou y sont/est dans S_{new}. Il reste cependant nécessaire de calculer le lca de tout couple de sommets de $\bar{S}$. Ce qui induit la complexité suivante :

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

Plusieurs algorithmes existent pour identifier un sommet $v \in lca(x, y)$. Des solutions optimisées s'appuyant sur une transformation de ce problème en une variante du plus court chemin ont été proposées. Elles permettent d'obtenir, pour n'importe quel couple de sommets (x, y), un des sommets appartenant à $lca(x, y)$ en $O(|V|^{2.7})$ [Bender, Pemmasani, Skiena and Sumazin 2001; Yuster 2008]. Cependant, ces solutions ne permettent pas

d'obtenir l'ensemble complet $lca(x, y)$ mais un seul représentant de cet ensemble. La complexité en temps de l'algorithme moins naïf est $O(|V|^2)$ multiplié par la complexité en temps permettant d'obtenir l'ensemble $lca(x, y)$ dont la complexité est au moins égale à $O(|V|^{2.7})$. Bien que meilleure que pour la version la plus naïve de l'algorithme, la complexité globale de cet algorithme calculant la LCA-fermeture reste assez élevée et inadaptée pour le traitement de grandes ontologies.

Name: LCA-fermetureNaïve
Input: un *dag* G et un sous-ensemble S des sommets de G.
Result: $\bar{S}$ la LCA-fermeture de S.

$S_k \leftarrow S_0$; $S_{new} \leftarrow S_0$;
do
  $S_{tmp} \leftarrow \emptyset$;
  for each $(x, y) \in S_k \times S_k$
    $S_{tmp} \leftarrow S_{tmp} \cup lca(x, y)$;
  end
  $S_{new} \leftarrow S_{tmp}$;
  $S_k \leftarrow S_k \cup S_{new}$;
while $S_{new} \neq \emptyset$
return S_k

Name: LCA-fermetureMoinsNaïve
Input: un *dag* G et un sous-ensemble S des sommets de G.
Result: $\bar{S}$ la LCA-fermeture de S.

$S_k \leftarrow S_0$; $S_{new} \leftarrow S_0$;
do
  $S_{tmp} \leftarrow \emptyset$;
  for each $(x, y) \in S_k \times S_{new}$
    $S_{tmp} \leftarrow S_{tmp} \cup lca(x, y)$;
  end
  $S_{new} \leftarrow S_{tmp}$;
  $S_k \leftarrow S_k \cup S_{new}$;
while $S_{new} \neq \emptyset$
return S_k

Algorithme 1 : Deux algorithmes basiques pour le calcul de LCA-fermeture

4.2 Un algorithme efficace en $O(|\bar{S}||E|)$ pour le calcul de la LCA-fermeture

Cette section détaille un algorithme optimisé qui détermine les sommets appartenant à la LCA-fermeture d'un sous-ensemble S des sommets d'un *dag* $G = (V, E)$ en $O(|\bar{S}||E|)$. L'idée clé de cet algorithme est que, bien qu'il y ait $O(|V|^2)$ couples de sommets, il n'y a que $O(|V|)$ sommets distincts qui peuvent être ajoutés à S. Au lieu de se demander pour chaque paire de sommets si son *lca* doit être conservé, il est plus efficace de se demander pour chaque sommet du *dag* s'il doit être ou non conservé. Cela peut se faire de manière efficace en utilisant une approche gloutonne, qui s'appuie sur l'ordre des sommets induit par la topologie du *dag*.
L'algorithme de LCA-fermeture que nous proposons traite les sommets du *dag* suivant l'ordre post-fixé, i.e. un sommet est toujours traité après que tous ses descendants l'aient été. En effet, les sommets d'un *dag* peuvent être positionnés le long d'une ligne horizontale de manière à ce que tous les descendants d'un sommet soient positionnés à sa droite. L'ordre sur les sommets qui résulte de ce positionnement horizontal est dit post-fixé, car il peut être obtenu efficacement à partir des indices post-fixés calculés lors du parcours en profondeur du *dag* décrit dans l'Algorithme 2 [Thomas H. Cormen et al. 1992].

Name: postOrder
Input: un *dag G*
Result: la liste des sommets de *G* en ordre post-fixé
  G.postOrder ← la liste vide
  for *root* **in** *G*.nodes() tel que *root* n'a pas de père
    postOrderRec(*root*)
  return *G*.postOrder

Name: postOrderRec
Input: un *dag G*, un sommet *n* of *G*
Result: ajoute, les descendants de *n* à la liste des sommets de *G* ordonnés en ordre post-fixé
  marquer *n* en tant que sommet visité
  for *s* **in** sons(*n*) tel que *s* n'est pas visité
    postOrderRec(*s*)
  ajouter *n* à la fin de la liste post-fixée des sommets de *G*

Algorithme 2 : Algorithmes permettant d'obtenir la liste des sommets de *G* suivant l'ordre post-fixé

Name: LCA-fermeture
Input: un *dag* G, un ensemble S de sommets d'intérêt.
Result: la LCA-fermeture de S.

$S^{AL} \leftarrow \emptyset$
Lpost = postOrder(G)
for *n* **in** *Lpost*
  $S^D(n) \leftarrow \emptyset$ // $S^D(n)$ ensemble des descendants de *n*
    // présents dans la LCA-fermeture de S
  $maxS^D(n) \leftarrow 0$ // $maxS^D(n)$ valeur maximale de
    // $|S^D(s)|$ pour les fils *s* de n
  for *s* **in** sons(*n*)
    $S^D(n) \leftarrow S^D(n) \cup S^D(s)$ (*)
    $maxS^D(n) \leftarrow \max(|S^D(s)|, maxS^D(n))$
  if $(n \in S)$
    $S^D(n) \leftarrow S^D(n) \cup \{n\}$ (**)
  if ($|S^D(n)| > maxS^D(n)$)
    $S^{AL} \leftarrow S^{AL} \cup \{n\}$ (***)
return S^{AL}

Algorithme 3 : Calcul de la LCA-fermeture

Property 4.2.1. (*Preuve de correction*). Etant donnés le *dag* $G = (V, E)$ et l'ensemble $S \subseteq V$, l'ensemble S^{AL} renvoyé par l'Algorithme 3 est la LCA-fermeture de S dans G, i.e. $S^{AL} = \bar{\bar{S}} = \bar{S}$.

Preuve. Soit P le tableau contenant les sommets de G suivant l'ordre post-fixé. L'Algorithme 3 traite successivement $P[1], \ldots, P[k], \ldots, P[|V|]$ et construit pour chaque valeur de *k*, le sous-ensemble S^{AL} noté $S^{AL}(k)$. Il est clair qu'au final $S^{AL} = S^{AL}(|V|)$ et nous allons montrer par récurrence que :

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

Les égalités de (1) sont vraies pour $k = 1$.
Par définition de l'ordre *post-fixé*, $P[1]$ n'a pas de descendant. Il n'est donc conservé dans S^{AL} que si $P[1] \in S \subseteq \bar{\bar{S}}$ (ligne (**) de l'algorithme). On a donc bien, $S^{AL}(1) = \bar{\bar{S}} \cap P[1..1]$ et $S^D(P[1]) = \bar{\bar{S}} \cap desc(P[1])$.

Si les égalités de (1) sont vraies pour les indices $1, \dots, (k-1)$, alors elles le sont également pour les indices $1, \dots, k$.
Soit $n = P[k]$ le sommet courant et $\{s_1, \dots, s_p\}$ l'ensemble des descendants immédiats (fils) de n. Comme les sommets sont considérés suivant l'ordre *post-fixé*, tout les fils de n sont à sa gauche dans le tableau P. A l'étape k, quand l'alogrithme considère le sommet n, tous ses fils $s_1, \dots, s_p$ ont déjà été traités et l'ensemble $S^D(s_i)$ a été sauvegardé pour chaque $s_i \in \{s_1, \dots, s_p\}$. L'ensemble enregistré pour n est donc :

$$S^D(n) = \begin{cases} S^D(s_1) \cup \dots \cup S^D(s_p) & si\ n \notin S \\ S^D(s_1) \cup \dots \cup S^D(s_p) \cup n & si\ n \in S \end{cases}$$

Le test de la ligne (***) de l'algorithme compare $|S^D(n)|$ avec $maxS^D(n) = \max\{|S^D(s_i)|, i = 1, \dots, p\}$.

- Si $n \in S$. Il est clair que $|S^D(n)| > maxS^D(n)$ et n est ajouté à $S^{AL}(k)$. On a, de manière évidente, $n \in \bar{\bar{S}} \cap P[1..k]$ et $S^D(P[k]) = \bar{\bar{S}} \cap desc(P[k])$.
- Si $n \notin S$. Dans ce cas, l'hypothèse de récurrence garantit que, tout sommet de $S^D = S^D(s_1) \cup \dots \cup S^D(s_p)$ est dans $\bar{\bar{S}}$ et que $S^D = S^D(n) - \{n\} = \bar{\bar{S}} \cap desc(n)$. La seule chose qui reste à prouver est que n n'est inclus dans $S^D(n)$ que s'il existe au moins deux sommets S^D dont il est le plus petit ancêtre commun.
  - Si le test (***) conduit à ajouter n dans S^{AL}, alors il existe au moins deux sommets z, t de S^D, tels que $n = lca(z, t)$.
    Suite au test (***) le sommet n est ajouté uniquement si $|S^D(n)| > maxS^D(n)$, ce qui ne peut arriver que s'il existe au moins deux sommets $z, t \in S^D$ tels que $\{z, t\} \nsubseteq S^D(s_i),\ i = 1, \dots, p$. Dans ce cas, il existe donc deux fils s_i, s_j de n qui sont distincts et tels que $z \in S^D(s_i),\ t \in S^D(s_j)$. D'après la définition du *lca*, $n \in lca(z, t)$ si et seulement si, $n \in A(z, t)$ et n n'a aucun descendant qui soit dans $A(z) \cap A(t)$. Le premier point est trivial et nous allons prouver le second par l'absurde. Supposons qu'il existe un sommet $n' \in A(z) \cap A(t)$ et un chemin orienté (n, n') dans G. Ce chemin passe nécessairement par l'un des fils de n. D'après l'hypothèse de récurrence, on sait que pour ce fils s_m l'égalité $S^D(s_m) = \bar{\bar{S}} \cap desc(s_m)$ est vérifiée, ce qui implique que $\{z, t\} \subseteq S^D(s_m)$. Ce qui est impossible par la définition de z et t.
  - Si le test (***) conduit à ne pas ajouter n dans S^{AL}, c'est qu'il n'existe aucun couple de sommets z, t de $S^D(n)$ tel que $n = lca(z, t)$.
    Suite au test (***) le sommet n n'est pas ajouté à S^{AL} si $|S^D(n)| = maxS^D(n)$. Ce qui ne peut se produire que s'il existe un indice $i \in \{1, \dots, p\}$ tel que $S^D(n) = S^D(s_i)$. Dans ce cas, n ne peut être le *lca* d'aucun couple de sommets (z, t). En effet, pour tout couple (z, t), s_i est (par construction) un ancêtre de z, t et un descendant de n. Ce qui prouve que dans ce cas $n \notin \bar{\bar{S}}$ et termine la preuve de correction de notre algorithme. □

Propriété 4.2.2. (*Complexité en Temps de l'algorithme* LCA-fermeture) Pour un sous-ensemble S de sommets du *dag* $G = (V, E)$, l'Algorithme 3 s'exécute en $O(|\bar{S}||E|) = O(|V||E|)$.

Preuve. Le tri des sommets suivant l'ordre post-fixé est obtenu par un parcours en profondeur du *dag* qui se fait en $O(|E|)$. La complexité du reste de l'algorithme, composé de deux boucles **for** imbriquées est dominée par le temps d'exécution lié à la ligne (*). Cette instruction calcule l'union de deux ensembles contenant au plus $|\bar{S}|$ éléments et cela pour chaque fils de chaque sommet du *dag* G, i.e. $O(|E|)$ fois. La complexité globale de l'algorithme est donc en $O(|\bar{S}||E|)$. Dans le pire des cas, $|\bar{S}|$ est égal à $|V|$ ce qui aboutit à une complexité en $O(|V||E|)$.

Propriété 4.2.3. (*Complexité en espace de l'algorithme de* LCA-fermeture) Pour un sous-ensemble S de sommets du *dag* $G = (V, E)$, l'Algorithme 3 nécessite un espace mémoire en $O(|\bar{S}||V|) = O(|V|^2)$.

Preuve. Un sous-ensemble de $\bar{S}$ est stocké pour chaque sommet de G. Dans le pire des cas, $|\bar{S}| = |V|$ et l'espace nécessaire est donc en $O(|V||E|)$.

Cependant, dans la pratique, on a généralement $|\bar{S}| \ll |V|$. De plus, une fois que tout les pères d'un nœud n ont été traités, le sous-ensemble $\bar{S}$ associé à n devient inutile, et l'espace mémoire associé peut donc être libéré. Ceci peut être réalisé efficacement en maintenant un compteur pour chaque sommet qui est initialement égal au nombre de pères de ce sommet. Chaque fois que l'on traite un sommet, les compteurs de ses fils sont décrémentés d'une unité, et si l'un d'eux atteint zéro la mémoire associée à ce fils de n est alors libérée. Cette optimisation ne change pas la complexité dans le pire des cas. En effet, elle est inefficace dans le cas où le *dag* est constitué d'un nœud racine ayant $(|V| - 1)$ fils. Cependant, elle réduit considérablement la mémoire nécessaire dans la plupart des cas réels.

Les algorithmes décrits dans cette section permettent d'identifier l'ensemble des concepts pertinents pour appréhender un ensemble de concepts d'intérêt. Néanmoins cela ne suffit pas, il faut également identifier les relations qui vont lier ces concepts dans la sous-ontologie.

5. CONSTRUCTION DE LA SOUS-ONTOLOGIE

La section précédente décrit un algorithme efficace permettant de calculer la LCA-fermeture d'un ensemble de concepts. Il faut noter que lorsqu'on considère un ensemble S de sommets, tous les *plus grands descendants communs* ou GCD (Greatest Common Descendants) d'un couple de sommets de S sont également pertinents pour mieux comprendre les relations entre les concepts de S. Les résultats et les algorithmes de la section précédente portant sur les *lca* se transposent aisément aux *gcd.* Cela se comprend facilement au vue de la dualité de ces deux opérateurs. Soit G le *dag* (V, E) et G^* le *dag* $G^* = (V, E^*)$ tel que $(v, u) \in E^*$ si et seulement si $(u, v) \in E$. Alors, pour tout x et y de V, un lca (resp. gcd) de x et de y dans G est un gcd (resp. lca) de x et de y dans G^*. Ayant noté $\bar{S}$ la LCA-fermeture de S, nous noterons, par analogie, $\underline{S}$ la GCD-fermeture.

5.1 Utilisation des LCA et GCD-fermetures pour définir un ensemble de concepts pertinents

Etant donné un ensemble S de concepts d'intérêt, l'ensemble de concepts pertinents conservés dans la sous-ontologie centré sur S est l'ensemble $S_p = \bar{S} \cup \underline{S}$. Comme le montre l'exemple de la Figure 4, cet ensemble n'est fermé ni pour l'opérateur lca ni pour le gcd. Dans cet exemple B1 et D appartiennent tout deux à S_p alors que leur plus petit ancêtre commun (le sommet A1) n'appartient pas à cet ensemble. Il peut alors sembler tentant de bâtir la sous-ontologie à partir de l'ensemble $\overline{\underline{S}}$ défini par la fermeture des deux opérateurs. De manière plus formelle, considérons la séquence (croissante) d'ensembles de sommets définie par : $S_0 = S$ et $S_{k+1} = \overline{S_k} \cup \underline{S_k}, k = 0, 1, \ldots$ L'ensemble $\overline{\underline{S}}$ est égal à l'ensemble S_c pour l'indice c tel que $\forall k \geq c, S_k = \overline{S_k} \cup \underline{S_k}$.

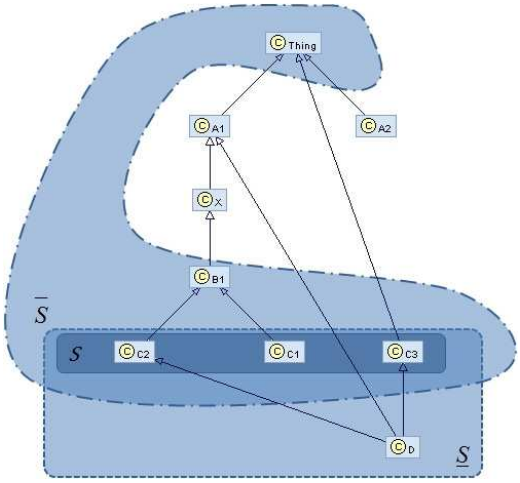

Figure 4 : Un exemple simple de l'ensemble de concepts pertinents $S_p = \bar{S} \cup \underline{S}$

Cette définition constructive fournit un algorithme permettant de calculer l'ensemble $\overline{\underline{S}}$ en construisant chaque ensemble S_k en appelant successivement l'algorithme de LCA-fermeture puis celui de GCD-fermeture. L'algorithme résultant a une complexité en $O\left(\left|\overline{\underline{S}}\right|^2 |E|\right)$. Il n'y a donc pas de difficulté particulière à utiliser l'ensemble de concepts $\overline{\underline{S}}$ au lieu de $\bar{S} \cup \underline{S}$. Si l'on considère ces deux ensembles de plus près, on peut noter que $\bar{S} \cup \underline{S} \subseteq \overline{\underline{S}}$. De plus les concepts de $\overline{\underline{S}}$ absents de $\bar{S} \cup \underline{S}$ sont ceux obtenus en utilisant à la fois l'opérateur lca et l'opérateur gcd. Par exemple, le concept A1 de la Figure 4 est présent dans $\overline{\underline{S}}$ du fait que $A1 = \text{lca}(\text{lca}(C1, C2), \text{gcd}(C2, C3))$. La relation entre un tel concept et les concepts de départ est de ce fait difficile à interpréter. Les inclure dans la sous-ontologie risque d'ajouter plus de confusion que de sémantique. Nous avons donc choisi de bâtir la sous-ontologie centrée sur S en n'utilisant que les concepts de $\bar{S} \cup \underline{S}$ ce qui est à la fois plus rapide et plus porteur de sens que d'intégrer tout les concepts de $\overline{\underline{S}}$.

5.2 Ajouter les relations d'hyponymie et d'hyperonymie entre concepts pertinents

Une fois que l'ensemble de concepts pertinents S_p est identifié, il reste à identifier les relations d'hyponymie et d'hyperonymie entre ces concepts. En effet, il est souhaitable de préserver ces relations entre les concepts de S_p même si certains concepts intermédiaires n'ont pas été gardés.

Plus précisément, étant donné un *dag* $G = (V, E)$, le sous-dag pertinent $G_p = (V_p, E_p)$ est défini comme suit :

- $V_p = \bar{S} \cup \underline{S}$
- $(u, v) \in E_p$ si et seulement si il existe un chemin orienté dans G de u à v sans passer par un sommet de V_p

L'ensemble des arcs E_p peut être obtenu efficacement en utilisant l'ordre des sommets induit par la topologie du *dag*. Cette fois-ci, un sommet u sera toujours traité après ses ascendants. Un tel ordre, que nous appellerons ordre prefixé, s'obtient simplement en considérant les sommets du vecteur post-fixé en partant de la fin. Soit $V_{APA}(u)$ l'ensemble des *Ancêtres Pertinents Atteignables* de u, i.e. l'ensemble contenant tous les sommets de V_p qui peuvent être atteints en partant de u en suivant un chemin qui ne passe par aucun autre sommet de V_p. En traitant les sommets selon l'ordre prefixé, l'ensemble $V_{APA}(u)$ du sommet courant est simplement l'union des ensembles $V_{APA}(f)$ de ses pères (father) qui ne sont pas dans V_p, et de ses pères qui sont dans V_p. L'Algorithme 4 détaille comment l'ensemble E_p peut ainsi être construit, en ajoutant pour chaque sommet u de V_p, les arcs (u, v) entre u et tout sommet v présent dans $V_{APA}(u)$.

Name: sousDagPertinent
Input: un DAG $G = (V, E)$ et un ensemble S de sommets d'intérêt
Result: G_p le sous-dag pertinent

$V_p \leftarrow \bar{S} \cup \underline{S}$
$G_p \leftarrow (V_p, \emptyset)$
for each *u* **in** reverse(postOrder(*G*))
  $V_{APA}(u) \leftarrow \emptyset$
  for each *f* **in** *fathers(u)*
    if $(f \notin V_r)$
      $V_{APA}(u) \leftarrow V_{APA}(u) \cup V_{APA}(f)$ (*)
    else
      $V_{APA}(u) \leftarrow V_{APA}(u) \cup f$
  if $(u \in V_r)$
    for each v **in** $V_{APA}(u)$
      $G_p.ajouterArete(u, v)$
return G_p

Algorithme 4 : Algorithme de construction du *sous-dag* pertinent

La complexité de cet algorithme est semblable à celle de la LCA-fermeture (Algorithme 3). Comme pour ce dernier, l'instruction clé est celle de la ligne (*), qui calcule l'union de deux ensembles contenant au plus $|V_p|$ éléments et qui est exécutée pour chaque père de chaque sommet du *dag* initial G, soit $O(|E|)$ fois.

5.3 Ajouter les autres relations entre les concepts pertinents

Bien que la relation "*est-un*" puisse être vue comme l'ossature des ontologies, la plupart d'entre elles permettent de spécifier

d'autres types de relation entre leurs concepts. Une fois le sous-dag $G_p = (V_p, E_p)$ pertinent obtenu à partir des relations de type "*est-un*" de l'ontologie de référence, il est légitime d'enrichir cette sous-ontologie σ_p en incluant au moins toute relation uRv entre deux concepts u, v de V_p qui est spécifiée dans σ.

Il est possible qu'une relation uRv entre deux concepts u, v de V_p puisse être déduite de l'ontologie originale σ même si cette dernière ne contient pas explicitement cette relation. Par exemple, si l'on note respectivement $EstUn$, $EstEq$ et $PartieDe$ les relations "*est-un*", "*est-equivalent-à*" et "*est-une-partie-de*" et que l'on considère le cas où σ contient les relations $A\ EstUn\ B$, $A\ PartieDe\ C$ et $A\ EstEq\ D$, et où $V_p = \{D, C\}$, la relation $D\ PartieDe\ C$ peut être incluse dans la sous-ontologie du fait qu'elle se déduit des relations $A\ PartieDe\ C$ et $A\ EstEq\ D$.

La formalisation des relations qui doivent être ajoutées à la sous-ontologie est loin d'être évident, notamment car cela nécessite de prendre spécifiquement en compte la sémantique de chaque type de relation. On peut mettre en évidence certaines règles, permettant de gérer les cas les plus simples et d'enrichir significativement la sous-ontologie. En particulier, pour chaque relation transitive R (i.e. telle que $aRb\ et\ bRc \Rightarrow aRc$), il est possible d'ajouter la relation uRv à σ_p pour tout couple de concepts u et v présents dans V_p et tel qu'il existe dans σ une série de relations $uRa_1, a_1Ra_2, \dots, a_nRv$ dont aucun des concepts a_i n'appartient à V_p. Cette règle est une simple généralisation de l'approche utilisée pour déterminer les relations de type "*est-un*" présentes dans σ_p.

L'algorithme décrit pour ajouter les relations de type "*est-un*" à l'ontologie σ_p s'appuie uniquement sur le fait que le graphe utilisé de manière sous-jacente pour représenter cette relation est un *dag*. Cet algorithme peut donc s'appliquer à toute relation transitive que l'on peut représenter sous forme de *dag*. C'est notamment le cas de la relation de subsomption ("*est-un*") et de la relation de composition (appelées aussi Tout-Parties – "*est-une-partie-de*") ; mais ce n'est pas le cas de la relation *est-equivalent-à* dont le graphe sous-jacent peut contenir des cycles ($A\ Eq\ B$, $B\ Eq\ C$ et $C\ Eq\ A$). Dans sa version actuelle OntoFocus rajoute les relations directes entre concepts, telles que les relations "*est-un*" et "*est-une-partie-de*" et utilise également l'Algorithme 4 pour rajouter les relations qu'elles induisent.

Les algorithmes décrits dans cette section sont implémentés dans l'application OntoFocus développée en JAVA à l'aide de la librairie Jena. Cette application prend en entrée une ontologie au format OWL et un ensemble S de concepts d'intérêt et fournit comme résultat la sous-ontologie centrée sur S. La section suivante détaille un cas d'utilisation concret de cette application.

6. APPLICATIONS, RESULTATS ET PERSPECTIVES

L'énorme quantité de documents numériques accumulés constitue une limite pour nombre d'applications. Les sciences de la vie sont particulièrement sensibles à ce problème. Les projets de séquençages de génomes ont, par exemple, conduit à la création de centaines de bases de données dédiées et au développement de nombreuses applications permettant d'interagir avec ces bases de données. Les ontologies sont souvent utilisées pour structurer ces informations, permettre la mise en place de requêtes sémantiques élaborées et fournir des outils de visualisations. La taille et le domaine d'application de ces bio-ontologies augmente presque aussi rapidement que les données elles mêmes. Cependant, dans la plupart des cas, seule une très petite partie des concepts de l'ontologie est réellement pertinente pour l'utilisateur. C'est donc un domaine où les outils développés dans cet article sont particulièrement utiles.

Nous avons utilisé OntoFocus pour restreindre la *Gene Ontology* (qui contient autour de 30 000 concepts) et obtenir une sous-ontologie centrée sur les concepts liés au gène BRCA1 (BReast CAncer) associé aux risques de cancer du sein. Pour ce faire, nous avons recherché dans la base de données Swiss-Prot l'identifiant de la protéine produite par le gène BRCA1. Nous avons ensuite téléchargé les annotations associées à cette protéine (dont l'identifiant est P38398) à l'aide du service web GO Annotation (GOA) fourni par l'European Bioinformatics Institute (http://www.ebi.ac.uk/GOA/). Nous avons ainsi obtenu une liste de 50 termes distincts de la Gene Ontology associés au gene BRCA1. A partir de ces 50 termes, OntoFocus a construit, en moins d'une minute, une sous ontology de GO contenant 92 concepts pertinents.

Comme on peut le constater sur la Figure 5, la visualisation de cette ontologie restreinte est possible de manière simple et elle peut facilement être appréhendée dans sa globalité. Cette sous-ontologie, tout comme la Gene Ontology, est constituée de trois parties distinctes. La Figure 5, se focalise sur deux de ces trois parties qui regroupent d'une part des concepts liés aux fonctions moléculaires (*molecular functions*) et d'autre part les concepts liés aux composants cellulaires (*cellular components*). La troisième partie de l'ontologie, correspondant aux concepts liés aux processus biologiques (*biological processes*), n'est pas représentée dans notre figure pour des questions de place. Cette dernière partie de notre sous-ontologie, bien que plus large que les deux autres, ne contient au total que 63 concepts et peut donc aussi être facilement visualisée dans son intégralité.

On peut souligner plusieurs utilisations possibles de ce type de sous-ontologies. Tout d'abord, le fait de disposer d'une sous-ontologie autonome centrée sur l'annotation d'un gène peut permettre au biologiste de mieux exploiter ces annotations. Par exemple, dans le cas du gène BRCA1, cela met en évidence le fait que plusieurs de ses annotations sont des raffinements du concept *protein binding function* (GO_0005515). Cette approche peut également être utilisée pour un ensemble de gènes ayant des caractéristiques communes (e.g. des profils d'expression similaires lors d'expériences de puces ADN). Dans ce cadre, les annotations de la gene ontology sont précieuses pour donner un sens biologique aux clusters de gènes mis en évidence. Pour ce type d'application il est possible d'automatiser complètement la sélection des gènes d'intérêt.

Ce type de sous-ontologie peut également être utile pour mettre à jour l'annotation d'un gène. La disposition d'une vue de l'ontologie centrée sur l'annotation que l'on modifie permet de mettre en évidence les concepts de cette annotation et les relations de spécialisation qui peuvent exister entre eux.

OntoFocus peut également être utilisé pour produire une sous-ontologie qui, bien que beaucoup plus petite, sera suffisante pour fournir le cadre sémantique d'une base de données particulière. En effet, GO est une ontologie très vaste et de nombreuses applications ou des bases de données sont centrées sur des aspects très particuliers ne recouvrant qu'une faible partie des concepts présents dans GO (e.g. la base de données IMGT dédiée au système immunitaire des vertébrés, http://imgt.cines.fr/).

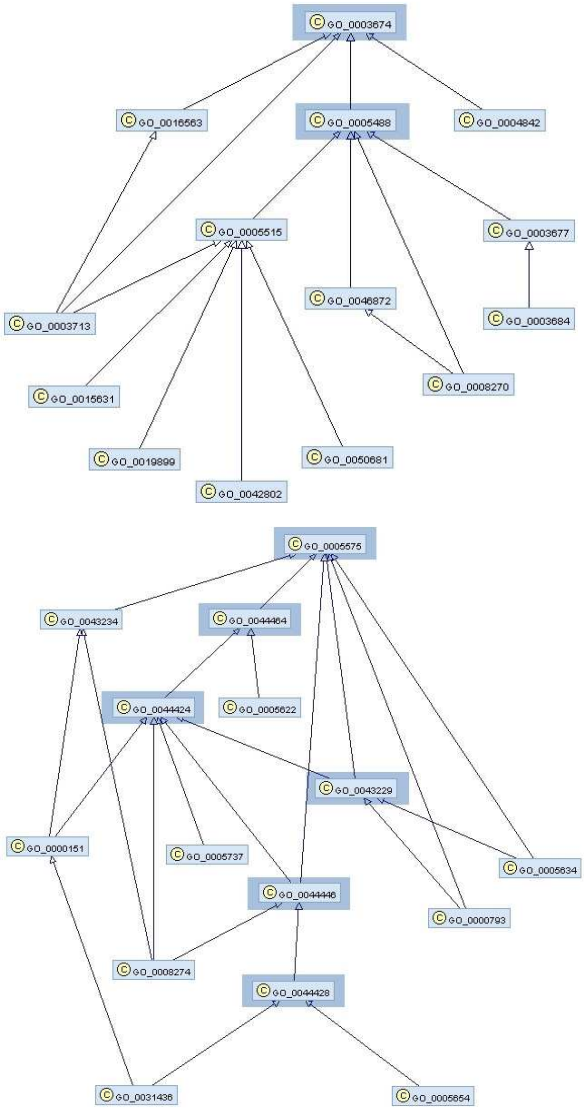

Figure 5 : Visualisation des parties *molecular function* (GO_0003674) et *cellular component* (GO_0005575) de la sous-ontologie de GO centrée sur les concepts utilisés pour annoter le gène BRCA1. (Les concepts encadrés sont ceux ajoutés par OntoFocus pour obtenir une ontologie autonome)

Bien que nous ayons utilisé un cas d'étude biologique pour illustrer l'intérêt de notre approche, elle peut évidemment s'avérer utile dans tout système d'information qui traite de grandes quantités d'information en s'appuyant sur de grandes ontologies. Le fait de disposer de sous-ontologies thématiques est un plus appréciable pour toutes les tâches qui nécessitent une interaction avec un expert humain et notamment pour la conception et l'évolution de l'ontologie ou pour le rendu visuel sur une carte conceptuelle.

Comme le souligne [Katifori et al. 2007], il existe de nombreux travaux sur la visualisation d'ontologies mais tous se concentrent sur une représentation hiérarchique de l'ontologie. Pour paraphraser [Wouters, Dillon, Rahayu and Chang 2005], on peut dire qu'avec ces représentations, "l'inconvénient majeur est que la sémantique portée par l'ontologie n'est, en général, pas totalement exploitée". Nous pensons qu'il existe d'autres façons de représenter graphiquement une ontologie. Nous avons récemment proposé une distance sémantique qui permet d'évaluer la proximité sémantique de deux concepts sur la base du *dag* "*est-un*" qui structure l'ontologie [Ranwez et al. 2006]. Cette distance permet d'obtenir une visualisation originale de l'ontologie en utilisant des techniques classiques de visualisation, telle que le *multidimensional scaling*. Nous envisageons de combiner l'utilisation d'OntoFocus et de cette distance sémantique pout développer dans notre environnement de visualisation (appelé MolAge) un outil capable de créer à la demande des cartes sémantiques centrées sur les concepts d'intérêt de l'utilisateur. De telles cartes pourraient être utilisées pour naviguer graphiquement dans une collection de documents annotés ou pour analyser son contenu. En effet, pour être réellement utile à l'utilisateur il est important que la représentation graphique prenne en compte la sémantique de sorte que l'exploration de documents à travers cette représentation lui soit intuitive. Dans ce but, nous envisageons d'enrichir la sous-ontologie avec des informations complémentaires telles que le nombre de fils/pères que possède chaque concept dans l'ontologie initiale et la longueur du chemin "*est-un*" que représente chaque arc de l'ontologie restreinte. Le but de ces ajouts est qu'il soit possible, en ne considérant que l'ontologie restreinte, de calculer pour tout couple de concepts de l'ontologie restreinte la distance sémantique qui les sépare au sens de l'ontologie d'origine. Cette capacité de zoomer sur une partie précise de très grosses ontologies permet, par exemple, de comparer visuellement la proximité sémantique d'un ensemble de concepts dans les différentes ontologies qui les contiennent.

7. CONCLUSION

Après avoir décrit le rôle important de l'extraction de sous-ontologie dans de nombreuses applications et la nécessité qui en découle de développer des outils efficaces pour cette tâche, cet article introduit une définition originale de la notion de concepts d'intérêt basée sur la théorie des graphes et un algorithme efficace pour les identifier. A partir d'une ontologie de référence et d'un ensemble de concepts d'intérêt, cet algorithme calcule leur LCA-fermeture et leur GCD-fermeture dans le graphe orienté acyclique induit par la relation "*est-un*" de l'ontologie. Il calcule ensuite, à partir des relations de l'ontologie d'origine, celles de l'ontologie restreinte. La complexité en temps de cet algorithme est proportionnelle au nombre de concepts conservés multiplié par le nombre d'arcs de l'ontologie d'origine.

Le programme OntoFocus qui implémente cet algorithme peut être utilisé gratuitement à partir de notre site web (http://www.ontotoolkit.mines-ales.fr/). Nous pensons que de nombreuses applications peuvent bénéficier d'un tel algorithme de restriction d'ontologies. Nous détaillons par exemple comment dans les sciences du vivant où la *gene onotlogy* est très utilisée, cette approche permet notamment de mieux comprendre l'annotation associée à un gène.

Nous avons accordé un soin particulier à l'optimisation afin que nos algorithmes aient de faibles complexités théoriques et que, en conséquence, nos logiciels puissent traiter de grandes ontologies en un temps raisonnable. Cela permet de créer de nouvelles sous-ontologies à la demande chaque fois que l'utilisateur à besoin d'une vue spécifique sur l'ontologie de référence. Cela garantit que les sous-ontologies soient toujours cohérentes avec l'originale et offre de nouvelles perspectives pour le développement d'outils

interagissant avec l'utilisateur en temps réel. On peut, par exemple, envisager la possibilité d'enchaîner plusieurs extractions pour que l'utilisateur puisse affiner progressivement sa requête jusqu'à obtenir une vue qui corresponde précisément à ses attentes.

Nous envisageons de poursuivre ce travail dans plusieurs directions et notamment d'utiliser cette approche pour compléter les outils de visualisation d'ontologies et pour structurer les documents renvoyés lors de requêtes en particulier dans le cadre de l'utilisation de la *gene ontology*.

8. REMERCIEMENTS

Nous tenons à remercier tout particulièrement Pierre Jean pour son aide précieuse dans la mise en place du site OntoToolkit.

Cette publication est le résultat d'une collaboration entre l'Institut des Sciences de l'Evolution de Montpellier (UMR 5554 - CNRS) et le centre de recherche LGI2P de l'Ecole des Mines d'Alès.

9. REFERENCES

[1] Aitkaci, H., Boyer, R., Lincoln, P. AND Nasr, R. 1989. Efficient Implementation of Lattice Operations. *Acm Transactions on Programming Languages and Systems 11*, 115-146.

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

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

[4] Berners-Lee, T., Hendler, J. AND Lassila, O. 2001. The Semantic Web. *Scientific American*, 29-37.

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

[6] Bhatt, M., Rahayu, W., Soni, S.P. AND Wouters, C. 2007. OntoMove: A knowledge based framework for semantic requirement profiling and resource acquisition. In *2007 Australian Software Engineering Conference, ASWEC'07*, 137-146.

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

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

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

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

[11] Kim, J.W., Conesa Caralt, J. AND Hilliard, J.K. 2007. Pruning Bio-Ontologies. In *Proceedings of the Proceedings of the 40th Annual Hawaii International Conference on System Sciences*2007 IEEE Computer Society.

[12] Navigli, R. 2002. Automatically extending, pruning and trimming general purpose ontologies. In *IEEE International Conference on Systems, Man and Cybernetics* Tunisy, 631- 635.

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

[14] Sabou, M. 2005. Learning web service ontologies: an automatic extraction method and its evaluation. In *Ontology learning from text: methods, evaluation and applications*, P. BUITELAAR, P. CIMIANO AND B. MAGNINI Eds. IOS Press, Amsterdam.

[15] Thomas H. Cormen, Charles E. Leiserson AND Ronald L. Rivest 1992. Graph Algorithms. In *Introduction to Algorithms* The MIT Press, Cambridge.

[16] Wouters, C., Dillon, T., Rahayu, W. AND Chang, E. 2005. Large scale ontology visualisation using ontology extraction. *International Journal of Web and Grid Services 1*, 113-135(123).

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

Annexe. Mémento des principales notations

En l'absence d'ambiguïté sur le graphe utilisé, celui-ci pourra être omis pour simplifier les notations

- **Graphes et sous-graphes**
  - $\boldsymbol{dag\ G = (V, E)}$ graphe orienté acyclique de sommets V et d'arêtes E.
  - $\boldsymbol{G[W]}$ le sous-graphe de G induit par $W \subseteq V$.
- **Degré des nœuds**
  - $\boldsymbol{d_G^+(v)}$ nombre d'arcs sortant du sommet v.
  - $\boldsymbol{d_G^-(v)}$ nombre d'arcs pointant sur le sommet v.
- **Ancêtres et plus petits ancêtres communs**
  - $\boldsymbol{A_G(v)}$ les ancêtres de v dans G. $\boldsymbol{A_G(S)} = \bigcap_{v \in S} A_G(v)$.
  - $\boldsymbol{lca(S)} = \{u | u \in A(S)\ et\ d^+_{G[A(S)]}(v) = 0\}$
  - $\boldsymbol{LCA(S)} = \bigcup_{x,y \in S} lca(x, y)$
- **Fermetures pour $\boldsymbol{G = (V, E)}$**
  - $\boldsymbol{\mathcal{L}(S)} = \{L \subseteq V \mid S \subseteq L\ et\ L = LCA(L)\}$
  - $\boldsymbol{\overline{S}} = \bigcap_{M \in \mathcal{L}(S)} M$, la LCA-fermeture de l'ensemble $S \subseteq V$.
  - $\boldsymbol{\underline{S}}$ la GCD-fermeture de l'ensemble $S \subseteq V$.
  - $\boldsymbol{\overline{\underline{S}}}$ la fermeture par LCA et GCD de l'ensemble $S \subseteq V$.