D 2 CP et computeBCov . Un prototype et un algorithme pour la découverte de services web dans le contexte du web sémantique

Christophe Rey

D²CP et *computeBCov*

Un prototype et un algorithme pour la découverte de services web dans le contexte du web sémantique

Christophe Rey

LIMOS – CNRS UMR 2239
Université Blaise-Pascal Clermont-Ferrand II
24, avenue des Landais
F-63177 Aubière cedex
rey@isima.fr

RÉSUMÉ. *Les services web représentent un nouveau paradigme prometteur pour le e-commerce. Jusqu'à ce jour, peu de systèmes existants fournissent des implémentations basées sur le web sémantique pour leur traitement. Cet article détaille le prototype* D²CP*, une des premières implémentations d'un tel système, permettant à la fois la définition et le stockage des services web dans des ontologies XML, et, au travers de l'algorithme* computeBCov *et de ses variantes, la découverte dynamique de combinaisons de services web pour répondre à une requête d'un utilisateur. Après un exemple de découverte dynamique,* computeBCov *est expliqué : on montre en particulier son lien avec le problème classique de recherche des transversaux minimaux dans un hypergraphe. Les fonctionnalités de* D²CP *sont ensuite présentées (génération aléatoire d'ontologies, choix de l'algorithme de découverte), ainsi que les résultats des tests qui ont permis de le valider qualitativement et quantitativement.*

ABSTRACT. *Web services are a new promising paradigm for e-commerce. Until now, very few systems apply semantic web principles to process them. This paper is about such a prototype called* D²CP. D²CP *allows both the definition and the storage of web services into XML-based ontologies, and, thanks to the* computeBCov *algorithm and to its variants, the dynamic discovery of combinations of web services to answer a user query. After an example of dynamic discovery,* computeBcov *is explained: in particular, its close link to the classical problem of the generation of minimal transversals of a hypergraph is shown. Then are presented* D²CP *capabilities (random ontologies generation, selection of the discovery algorithm), and finally tests results that validated* D²CP *are given.*

MOTS-CLÉS : *services web, services web sémantiques, découverte dynamique, ontologie, web sémantique, logiques de description, prototype, transversaux minimaux, hypergraphe.*

KEYWORDS: *web services, semantic web services, dynamique discovery, ontology, semantic web, description logics, prototype, minimal tranvserals, hypergraph.*

1. Introduction

Internet est en train de révolutionner la façon dont les entreprises interagissent avec leurs fournisseurs, partenaires et autres clients. Durant la dernière décennie, le nombre et la diversité des services se sont considérablement accrus sur le web, ce qui a abouti à l'apparition du e-commerce. Un nouveau paradigme pour le e-commerce a récemment été défini dans lequel les applications sont encapsulées et présentées sous la forme de services électroniques intégrés : les services web (ou e-services) [WEI 01, CAS 01c]. Un service web peut être défini comme une application rendue disponible sur internet par un fournisseur, et accessible à des clients [CAS 01b]. Des services de réservation en ligne, de gestion de comptes bancaires, ou même des applications métiers entières en sont des exemples actuellement courants. Ce qui rend la notion de service web si attirante est qu'il semble que les services web soient en mesure de permettre une meilleure automatisation dans la gestion des applications, par exemple en permettant leur découverte dynamique, en facilitant la communication et la négociation, voire en permettant leur composition en des services plus complexes [CAS 01a, WEI 01].

Dans ce contexte, le problème de la découverte dynamique des services web est le suivant : en supposant que l'on a un certain nombre de descriptions de services web (provenant de fournisseurs divers et concernant par exemple le domaine du tourisme) ainsi qu'une requête d'un utilisateur (cherchant par exemple un vol et une chambre d'hôtel), comment découvrir parmi tous les services, ceux qui correspondent le plus à la requête de l'utilisateur ? Et ceci dynamiquement puisque l'utilisateur ne sait pas *a priori* quels services il désire obtenir et puisque l'offre de services peut changer très rapidement (pour une même requête, à des instants différents, les services découverts ne seront pas toujours les mêmes, ceux-ci pouvant à tout moment être modifiés, supprimés ou d'autres pouvant être proposés).

Actuellement, ce problème est traité au sein de registres qui permettent la définition, le stockage et la découverte de services web [CAS 01b]. Ce sont par exemple les registres UDDI utilisant le langage de description de services WSDL, ou des registres ebXML utilisant UML[1]. Les algorithmes de découverte y sont néanmoins limités par le fait qu'ils sont basés sur une recherche par mots-clés, ou par tables de correspondance de couples (clé, valeur) [BER 02]. De plus, pour une requête, ces algorithmes découvrent des services web séparés, et non des combinaisons de services web couvrant l'ensemble de la requête.

Dans le cadre du web sémantique, dont le but est de baser les traitements d'informations du web sur une définition de leur sémantique, de nouveaux langages pour une description formelle plus riche et précise des services web ont été proposés : les logiques de description (LD), famille de langages issus de la logique du premier ordre limitée à deux variables, y occupent ainsi une place centrale, grâce aux différents niveaux d'expressivité qu'elles permettent et au nombre important de mé-

1. Voir http ://www.uddi.com, http ://www.w3.org/TR/wsdl, http ://openebxml.sourceforge.net

canismes d'inférence (appelés raisonnements) dont elles disposent[2]. Ces langages, comme par exemple DAML-S[3], permettent ainsi de définir la sémantique des services web, c'est-à-dire de les décrire en utilisant des termes qui ont eux-mêmes une définition logique précise et qui sont regroupés dans des dictionnaires structurés appelés ontologies [HOR 02b, HOR 02a, FEN 02, LUT 02, ANK 02]. Grâce aux raisonnements disponibles, on peut alors aller au-delà des techniques syntaxiques classiques de découverte, basées sur des recherches de mots-clés, et envisager d'utiliser des critères de correspondance sémantique entre les requêtes et les services disponibles : on peut les rapprocher sur la base de points communs inférés à partir de leurs définitions logiques. De telles méthodes de découverte de services ont été définies dans [BER 02, SYC 02, PAO 02, GON 01, TRA 02, LI 03]. A l'instar de ces travaux, nous avons proposé, sous la forme d'un nouveau raisonnement pour les LD, une découverte qui a l'avantage d'être la seule à notre connaissance à permettre la découverte de *combinaisons* de services web pour répondre à une requête, et à fournir la différence entre la requête et les combinaisons découvertes sous la forme d'un concept réutilisable dans d'autres raisonnements pour d'autres traitements (initiation d'un dialogue système-utilisateur par exemple). Nous présentons ces résultats dans [HAC 02b] et [HAC 02c].

Dans cet article, nous présentons l'algorithme appelé *computeBCov* qui permet ce nouveau raisonnement de découverte dynamique de combinaisons de services web, étant donné une ontologie de services web et une requête utilisateur au format XML, ainsi que le système D^2CP, qui implémente *computeBCov*. Nous présentons aussi la conception, les expérimentations et les évaluations de ce système qui, à notre connaissance, est un des premiers prototypes qui va aussi loin dans l'implémentation d'un algorithme de découverte fondée sur l'approche web sémantique des services web et utilisant les LD.

Cet article est organisé de la manière suivante. La section 2 résume, à partir d'un exemple, le problème de la découverte dynamique de services web. Puis l'algorithme *computeBCov* est détaillé à la section 3. Le système D^2CP et ses principales caractéristiques sont présentés à la section 4. A la section 5, nous discutons des tests effectués avec D^2CP. Nous positionnons nos résultats par rapport aux autres travaux à la section 6. Enfin, nous concluons en envisageant les travaux à venir.

2. Découverte dynamique de services web

Au travers d'un exemple détaillé, nous rappelons, à la section 2.1, les résultats donnés dans [HAC 02b, HAC 02c]. Puis, à la section 2.2, nous nous positionnons par rapport aux autres découvertes de services existant dans le contexte du web sémantique.

---

2. Pour plus de détails sur les logiques de description, se référer au *Description Logics Handbook* [BAA 03].
3. Voir http ://www.daml.org/services/

Ontologie générale

$Heure$	$\equiv$	$\forall heures.Entier \sqcap$ $\forall minutes.Entier$	Conjonction d'un entier pour les heures et d'un entier pour les minutes.
$Date$	$\equiv$	$\forall num\_jour.Entier \sqcap$ $\forall num\_mois.Entier \sqcap$ $\forall num\_annee.Entier \sqcap$ $\forall jour\_sem.Chaine\_car \sqcap$ $\forall nom\_mois.Chaine\_car$	Conjonction d'un entier pour le jour, un pour le mois, un pour l'année, d'une chaîne pour le jour de la semaine et pour le nom du mois.

Ontologie du domaine du tourisme

$Voyage$	$\equiv$	$\forall lieu\_dep.Chaine\_car \sqcap$ $\forall lieu\_arr.Chaine\_car \sqcap$ $\forall moyen\_transport.Chaine\_car \sqcap$	Conjonction d'une chaîne pour le lieu de départ, une pour le lieu d'arrivée et une pour le moyen de transport.
$Logement$	$\equiv$	$\forall lieu\_sejour.Chaine\_car \sqcap$ $\forall premier\_jour.Date$	Conjonction d'une chaîne pour le lieu et d'une $Date$ pour le premier jour.

Ontologie de services web

$Hotel$	$\equiv$	$Logement \sqcap$ $\forall nb\_lits.Entier \sqcap$ $\forall categorie.Chaine\_car \sqcap$ $\forall equip\_loisirs.Television$	Sorte de $Logement$ avec TV, un nombre de lits et une chaîne indiquant la catégorie de l'hôtel.
$Appartement$	$\equiv$	$Logement \sqcap$ $\forall nb\_chambres.Entier \sqcap$ $\forall categorie.Chaine\_car$	Sorte de $Logement$ avec un nombre de chambres et une chaîne pour la catégorie d'appartement.
$Horaire\_Depart$	$\equiv$	$Voyage \sqcap$ $\forall heure\_dep.Heure \sqcap$ $\forall date\_dep.Date$	Sorte de $Voyage$ avec une $Heure$ de départ et une $Date$ de départ.
$Horaire\_Arrivee$	$\equiv$	$Voyage \sqcap$ $\forall heure\_arr.Heure \sqcap$ $\forall date\_arr.Date$	Sorte de $Voyage$ avec une $Heure$ d'arrivée et une $Date$ d'arrivée.

Une requête possible

"J'arriverai à Paris le lundi 29 octobre et je voudrais un logement avec piscine."	$Requete\ Q$	$\equiv$	$\forall lieu\_arr.Chaine\_car \sqcap$ $\forall date\_arr.Requete\_partie\_1 \sqcap$ $Logement \sqcap$ $\forall equip\_loisirs.Piscine$
	$Requete\_partie\_1$	$\equiv$	$\forall num\_jour.Entier \sqcap$ $\forall jour\_sem.Chaine\_car \sqcap$ $\forall nom\_mois.Chaine\_car$

Figure 1. *Une petite ontologie $\mathcal{T}$ du tourisme et une requête Q possible où chaque définition est composée d'un nom de concept, qu'on appellera aussi concept défini, du signe $\equiv$ qui signifie l'équivalence logique et d'une description de concept qui est l'expression dans le langage $\mathcal{FL}_0$ qui définit le concept défini*

2.1. *Exemple*

Nous détaillons ici un exemple de découverte dynamique de services web, dans le domaine du tourisme. La figure 1 nous donne la définition dans la logique de description $\mathcal{FL}_0$ d'une ontologie $\mathcal{T}$ contenant des définitions de concepts et de services relatifs au tourisme ainsi qu'une requête Q possible. La figure 2 présente de manière informelle ces services ainsi que la requête.

Description des services web

Hôtel	Permet de consulter une liste d'hôtels dont les chambres sont équipées d'une télévision, en donnant quelques informations comme le lieu, la date du premier jour, le nombre de lits ou la catégorie de l'hôtel.
Appartement	Permet de consulter une liste d'appartements en donnant quelques informations comme le lieu, la date du premier jour, le nombre de chambres et la catégorie d'appartement.
Horaire_Départ	Permet de consulter une liste de voyage étant donné les lieux de départ et arrivée et l'heure et la date de départ.
Horaire_Arrivée	Permet de consulter une liste de voyage étant donné les lieux de départ et arrivée et l'heure et la date d'arrivée.

Une requête possible

Q	"J'arriverai à Paris le lundi 29 octobre et je voudrais un logement avec piscine." Demande d'une combinaison de services web contenant un lieu d'arrivée décrit par une chaîne de caractères, associé avec une date d'arrivée décrite par un nom de jour, un numéro de jour et un nom de mois, et un logement, associé avec un équipement de loisir qui est une piscine.

Figure 2. *Description informelle des services et de la requête présentés à la figure 1*

$$
\begin{aligned}
Requete\ Q \equiv\ & \forall lieu\_arr.Chaine\_car \sqcap \\
& \forall date\_arr.\forall num\_jour.Entier \sqcap \\
& \forall date\_arr.\forall jour\_sem.Chaine\_car \sqcap \\
& \forall date\_arr.\forall nom\_mois.Chaine\_car \sqcap \\
& \forall lieu\_sejour.Chaine\_car \sqcap \\
& \forall premier\_jour.\forall num\_jour.Entier \sqcap \\
& \forall premier\_jour.\forall num\_mois.Entier \sqcap \\
& \forall premier\_jour.\forall num\_annee.Entier \sqcap \\
& \forall premier\_jour.\forall jour\_sem.Chaine\_car \sqcap \\
& \forall premier\_jour.\forall nom\_mois.Chaine\_car \sqcap \\
& \forall equip\_loisirs.Piscine
\end{aligned}
$$

Figure 3. *La requête normalisée (après extension de sa description)*

D'après la figure 1, l'ontologie est organisée en trois couches : une pour les définitions de concepts généraux comme $Heure$ et $Date$, une pour les concepts sur le tourisme comme $Voyage$ et $Logement$, et une pour les services web. Chaque couche est définie en fonction des concepts définis dans les autres couches (les définitions récursives sont interdites), en fonction de concepts de base appelés concepts atomiques ($Chaine\_car$, $Entier$ par exemple) et en fonction des rôles atomiques qui représentent les liens entre concepts (comme par exemple $lieu\_dep$, pour « lieu de départ », qui relie le concept défini $Voyage$ au concept atomique $Chaine\_car$). Comme cela est expliqué dans [COR 03], cette architecture à trois couches est une particularité de notre approche : cela permet de définir des services web indépendamment de tout fournisseur, ce qui rend la plate-forme de e-commerce bien plus réactive face à la grande mouvance du monde des services web.

A partir de l'ontologie et de la requête données à la figure 1, le processus de découverte se poursuit en 4 étapes. L'étape 1 consiste en la normalisation de l'ontologie et de la requête : toutes les descriptions sont « étendues », c'est-à-dire que tous les concepts définis par une description et apparaissant dans la description d'un autre concept défini sont remplacés par leur description. Comme l'ontologie ne présente pas

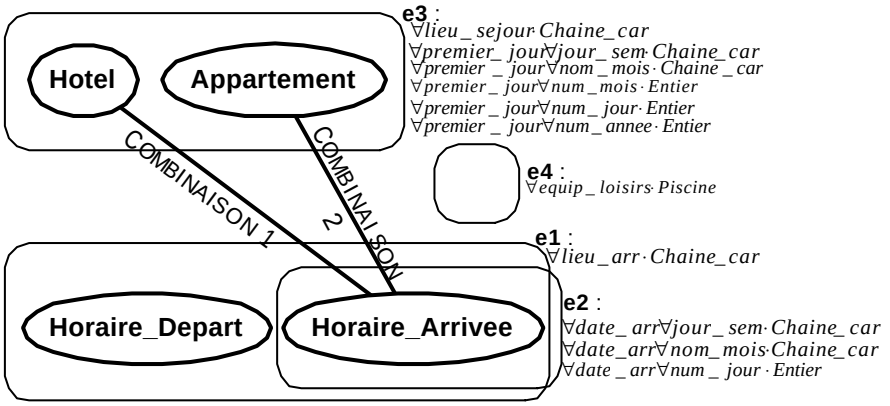

Figure 4. *Hypergraphe $\mathcal{H}_{\mathcal{T}Q}$ construit à partir de l'ontologie $\mathcal{T}$ et de la requête Q de la figure 1, et les transversaux minimaux associés (combinaisons 1 et 2)*

de définition récursive, ce processus se termine toujours, au pire en un temps exponentiel par rapport à la taille de l'ontologie (donnée par le nombre de concepts définis et la longueur maximale des descriptions). Ainsi chaque concept défini normalisé n'est exprimé qu'en fonction des concepts atomiques, c'est-à-dire des noms de concepts qui ne sont associés à aucune description. La figure 3 montre le résultat de la normalisation de la requête. L'étape 2 consiste en la construction d'un hypergraphe $\mathcal{H}_{\mathcal{T}Q}$ à partir des clauses de la requête normalisée, c'est-à-dire des termes de la conjonction (termes séparés par le symbole $\sqcap$) obtenue après extension de la description de la requête. Les sommets de cet hypergraphe sont les services web, et les arêtes e_i sont les clauses de la requête. Un sommet est dans une arête si le service correspondant contient la clause correspondante. La figure 4 donne l'hypergraphe correspondant à l'exemple courant. Les transversaux minimaux de cet hypergraphe sont ensuite calculés durant l'étape 3, afin de trouver les combinaisons de services qui maximisent l'information commune avec la requête. Dans l'exemple courant, les transversaux minimaux de cet hypergraphe sont, d'après la figure 4, les combinaisons 1 et 2, soit respectivement $\{Horaire\_Arrivee, Hotel\}$ et $\{Horaire\_Arrivee, Appartement\}$. Enfin, l'étape 4 consiste en l'identification parmi ces combinaisons de celles qui minimisent l'information apportée en plus par rapport à la requête : pour chaque combinaison, on fait la somme des tailles des clauses qui n'apparaissent pas dans la requête, et on garde les combinaisons pour lesquelles cette valeur est la plus petite. Dans notre exemple, $\{Horaire\_Arrivee, Appartement\}$ est ainsi plus proche de la requête que $\{Horaire\_Arrivee, Hotel\}$ puisqu'$Hotel$ contient une clause en plus qui est $\forall equip\_loisirs.Television$. La meilleure combinaison de services pour répondre à la requête est donc $\{Appartement, Horaire\_Arrivee\}$.

Comme nous pouvons le voir sur la figure 5, l'utilisateur peut connaître la partie de sa requête qui n'a pas été comprise (*i.e.* qui ne correspond à aucun service web),

Meilleure combinaison de services	Rest : partie de la requête absente de tout service	Miss : partie de la combinaison absente de la requête
$\{Horaire\_Arrivee, Appartement\}$	$\forall equip\_loisirs.Piscine$	$\forall categorie.Chaine\_car$ $\forall lieu\_dep.Chaine\_car$ $\forall moyen\_transport.Chaine\_car$ $\forall heure\_arr.\forall heures.Entier$ $\forall heure\_arr.\forall minutes.Entier$ $\forall date\_arr.\forall num\_mois.Entier$ $\forall date\_arr.\forall num\_annee.Entier$ $\forall nb\_chambres.Entier$

Figure 5. *Résultats de la découverte dynamique de services web*

c'est le « rest », et la partie de la combinaison des services web qui n'a pu être mise en correspondance avec aucun élément de la requête, c'est le « miss ». Les étapes 3 et 4 vues précédemment correspondent en fait à la minimisation respective des tailles de ces *rest* et *miss*. Afin que les services web découverts puissent être exécutés, il faut que l'utilisateur renseigne toutes les informations des services web pour ne plus avoir de *miss*. Le système peut ainsi se baser sur ce *miss* pour établir un dialogue avec l'utilisateur pour qu'il complète sa requête.

Dans [HAC 02a, HAC 02b, HAC 02c] sont donnés les résultats théoriques qui justifient cette approche. Nous avons d'abord formalisé la découverte dynamique de services web, dans le contexte du web sémantique, en définissant un nouveau raisonnement complexe pour les logiques de description appelé la « recherche des meilleures couvertures d'un concept en utilisant une terminologie » (*i.e.* en utilisant une ontologie dans le contexte LD). Ce nouveau raisonnement a été défini pour les logiques de description à subsomption structurelle selon la définition de Teege [TEE 94] (dont $\mathcal{FL}_0$ fait partie). Puis, nous avons montré que ce nouveau raisonnement était en fait une nouvelle instance du cadre formel du problème de réécriture de concepts en utilisant une terminologie, cadre formalisé dans [BAA 00a, BAA 00b]. Nous avons enfin montré que la recherche des meilleures couvertures d'un concept est équivalente au problème **NP**-difficile de recherche des minimaux transversaux de coût minimal dans un hypergraphe.

2.2. *Autres approches web sémantique de la découverte de services*

On peut constater que la découverte de services présentée à la section précédente se fait avec des descriptions de services (et de requêtes) abstraites et non concrètes, c'est-à-dire qu'un service est décrit par les notions qui le définissent conceptuellement, et non en termes de processus, d'enchaînement d'opérations ou de flots de données. Par exemple, le service $Hotel$ est décrit comme un logement d'une certaine catégorie ayant un nombre de chambre et un équipement de loisir qui est une télévision, et non comme un service qui serait composé d'une opération de recherche de chambre selon des critères, d'une demande de renseignements et enfin d'une transaction bancaire pour la réservation.

Ainsi, dans le contexte du web sémantique, les approches existantes de découverte de services web dans lesquelles ces derniers sont décrits abstraitement sont données dans [GON 01, TRA 02], dans [PAO 02] et dans [LI 03][4]. La figure 6 les compare précisément avec la nôtre. Les critères de comparaison sont les suivants : la nature des correspondances découvertes (services séparés ou combinaisons de services), les catégories de description abstraite utilisées (globale comme pour le service $Hotel$ ou en termes d'entrées-sorties), les différentes définitions de correspondance entre requête et services, le degré de correspondance (classement des réponses possibles) et les spécificités de certaines approches.

La conclusion générale de cette étude comparative est que notre approche est un peu plus flexible, renvoie des solutions plus variées et permet une mesure plus fine de la distance entre la requête et les réponses, au prix de l'utilisation de langages de représentation de connaissances moins expressifs.

3. L'algorithme *computeBCov*

Nous présentons maintenant l'algorithme *computeBCov* pour le calcul des transversaux minimaux de coût minimal, appliqué à l'hypergraphe $\mathcal{H}_{\mathcal{T}Q}$ construit comme indiqué précédemment à partir d'une ontologie $\mathcal{T}$ de services et d'une requête Q, le tout exprimé dans le langage $\mathcal{FL}_0$. Cet algorithme est fondé sur l'algorithme classique de calcul des transversaux minimaux, optimisé, et adapté pour calculer les transversaux de coût minimal. L'algorithme classique est l'algorithme 1 présenté à la section 3.1. L'algorithme optimisé est l'algorithme 2 : il résulte du théorème 3.1 présenté à la section 3.2 et évalué en termes de complexité à la section 3.3. L'algorithme optimisé et adapté est l'algorithme 3, appelé *computeBCov* : il résulte de l'utilisation d'une technique d'optimisation combinatoire dite de séparation et évaluation (ou Branch and Bound, BnB, en anglais) présentée à la section 3.5. De plus, un petit état de l'art sur le problème des transversaux minimaux est donné à la section 3.4.

3.1. *Algorithme classique de calcul des transversaux minimaux d'un hypergraphe*

Commençons par rappeler quelques définitions utiles concernant les hypergraphes.

Définition 3.1 (Hypergraphe et transversaux) [BER 89, EIT 95]
Un *hypergraphe* $\mathcal{H}$ est un couple (Σ, Γ) composé d'un ensemble $\Sigma = \{v_1, \ldots, v_n\}$ et d'un ensemble $\Gamma = \{e_1, \ldots, e_m\}$ de sous-ensembles de Σ. Les éléments de Σ sont nommés les sommets et les éléments de Γ les arêtes. La taille $|\mathcal{H}|$ de $\mathcal{H}$ est le nombre m de ses arêtes. Cependant on parle aussi de la taille combinée de l'hypergraphe qui représente le couple des nombres n de sommets et m d'arêtes.

4. La terminologie anglo-saxonne évoque les notions de « services discovery », « selection », « matching » ou encore « matchmaking ».

	Correspondance entre une requête Q et ...
Notre approche	une combinaison E de services. Correspondance 1..N
[GON 01, TRA 02]	1 service S. Correspondance 1..1
[LI 03]	1 service S. Correspondance 1..1
[PAO 02]	1 service S. Correspondance 1..1
Conclusion	Notre approche est plus flexible car explore et renvoie des solutions plus variées.
	Type de description des services et langage utilisé
Notre approche	Description abstraite globale des services. Dans [BEN 03], nous avons montré que notre découverte et l'algorithme associé s'appliquaient aussi à des descriptions abstraites de services en termes d'entrées et de sorties. Langages $\mathcal{FL}_0$ et toutes les LD à subsomption structurelle.
[GON 01, TRA 02]	Description abstraite globale des services. Langage DAML+OIL (basé sur la LD $\mathcal{SHIQ}$)
[LI 03]	Description abstraite globale des services. Langage DAML-S (basé sur DAML+OIL et donc sur la LD $\mathcal{SHIQ}$)
[PAO 02]	Description abstraite des services en termes d'entrées et de sorties. Langage DAML-S (basé sur DAML+OIL et donc sur la LD $\mathcal{SHIQ}$)
Conclusion	Les langages de notre approche sont moins expressifs, même s'ils restent tout à fait utilisables en pratique pour décrire des services.
	Définition de la correspondance
Notre approche	Existence d'une partie commune entre Q et E.
[GON 01, TRA 02]	Q ne doit pas contredire S.
[LI 03]	Q ne doit pas contredire S.
[PAO 02]	Chaque sortie de Q doit correspondre à une sortie de S et chaque entrée de S doit correspondre à une entrée de Q.
Conclusion	Même niveau de flexibilité pour toutes les approches.
	Degré de correspondance entre Q et E ou S.
Notre approche	Les solutions E sont classées selon la taille du rest, puis selon la taille du miss. Cette classification inclus en particulier l'ordre suivant, du meilleur au moins bon : – exact : Q et E sont strictement équivalents – subsume : Q subsume E (i.e. Q est plus générale que E) – plug-in : E subsume Q (i.e. E est plus générale que Q) – si pas de relation de subsomption entre Q et E : E acceptable si pas de meilleure solution – Q disjoint de E (i.e. Q contredit E) : E n'est pas acceptable
[GON 01, TRA 02]	Les solutions S sont classées selon l'ordre suivant, du meilleur au moins bon : – exact : Q et S sont strictement équivalents – subsume : Q subsume S (i.e. Q est plus générale que S) – plug-in : S subsume Q (i.e. S est plus générale que Q) – si S et Q sont subsumés par un même concept (i.e. il existe un concept C plus général que S et Q) : S est acceptable, si pas de meilleure solution – Q disjoint de S (i.e. Q contredit S) : S n'est pas acceptable
[LI 03]	Les solutions S sont classées selon l'ordre suivant, du meilleur au moins bon : – exact : Q et S sont strictement équivalents – plug-in : S subsume Q (i.e. S est plus générale que Q) – subsume : Q subsume S (i.e. Q est plus générale que S) – Q disjoint de S (i.e. Q contredit S) : S n'est pas acceptable
[PAO 02]	Les solutions S sont classées selon l'ordre suivant, qui s'applique d'abord aux sorties puis aux entrées de Q et S, du meilleur au moins bon : – exact : Q et S sont strictement équivalents – plug-in : S subsume Q (i.e. S est plus générale que Q) – subsume : Q subsume S (i.e. Q est plus générale que S) – Q disjoint de S (i.e. Q contredit S) : S n'est pas acceptable
Conclusion	Les degrés de correspondance qui classent les solutions sont un peu plus fins dans notre approche puisqu'ils englobent les principaux cas des autres approches.
	Spécificités
Notre approche	Au travers des rest et miss, notre approche donne une description précise de la distance qui existe entre Q et les solutions E. Cette mesure est un plus par rapport aux autres travaux qui ne donnent que la nature de cette distance. On peut ainsi utiliser ces rest et miss pour initier un dialogue avec l'utilisateur (par exemple créer un formulaire HTML à partir des informations manquantes des services obtenus pour qu'il les complète).
[PAO 02]	L'utilisateur peut choisir le degré minimal de correspondance qu'il désire pour effectuer la découverte.

Figure 6. *Comparaison avec les approches existantes de découverte dynamique de services décrits abstraitement*

Un *hypergraphe simple* ne possède aucun couple d'arêtes (e_1, e_2) avec $e_1 \subseteq e_2$ et $e_1 \neq e_2$. Un ensemble $X \subseteq \Sigma$ est un *transversal* de $\mathcal{H}$ si, pour chaque $e_i \in \Gamma$, on a $X \cap e_i \neq \emptyset$. Un transversal X de $\mathcal{H}$ est *minimal* si et seulement si aucun sous-ensemble strict X' de X n'est un transversal de $\mathcal{H}$. L'ensemble des transversaux d'un hypergraphe $\mathcal{H}$ est noté $tr(\mathcal{H})$. L'ensemble des transversaux minimaux d'un hypergraphe $\mathcal{H}$ est noté $Tr(\mathcal{H})$.

L'algorithme classique de calcul des transversaux minimaux d'un hypergraphe est donné dans [MAN 94]. Il est basé sur une propriété donnée dans [BER 89] et dans [EIT 95]. Cette propriété permet de calculer les transversaux minimaux d'un hypergraphe $\mathcal{H}$ à partir des transversaux minimaux de $\mathcal{H}'$, avec $\mathcal{H}'$ égal à $\mathcal{H}$ privé d'une arête. Ainsi, en partant de $\mathcal{H}$ privé de toutes ses arêtes, on obtient $Tr(\mathcal{H})$ en ajoutant itérativement toutes ses arêtes et en calculant à chaque itération les nouveaux transversaux minimaux à partir de ceux obtenus à l'itération précédente.

Soit l'itération j. On a calculé $Tr({\mathcal{H}'}_{j-1})$ et grâce à cela, on va pouvoir calculer $Tr({\mathcal{H}'}_{j})$, où ${\mathcal{H}'}_{j}$ est égal à ${\mathcal{H}'}_{j-1}$ plus l'arête e_j. Le calcul se fait en deux étapes. Il y a d'abord la génération d'un ensemble de transversaux candidats par calcul de toutes les unions possibles entre un transversal minimal $X \in Tr({\mathcal{H}'}_{j-1})$ et un sommet $v_i \in e_j$ (ligne 3 de l'algorithme 1). Puis, les transversaux candidats qui ne sont pas minimaux sont supprimés (ligne 4).

Algorithme 1 *Algorithme classique de calcul des transversaux minimaux d'un hypergraphe [MAN 94]*

Entrée(s) : Un hypergraphe simple $\mathcal{H} = (\Sigma, \Gamma)$.
Sortie(s) : $Tr(\mathcal{H})$
1: $Tr := \emptyset$;
2: **pour chaque** arête $e_j \in \Gamma$ **faire**
3: $\quad Tr_1 := \{X \cup \{v_i\} \mid X \in Tr$ et $v_i \in e_j\}$;
4: $\quad Tr_2 := \{X \in Tr_1 \mid X$ est minimal par rapport à $\subseteq\}$;
5: $\quad Tr := Tr_2$;
6: **fin pour**
7: Renvoyer Tr ;

3.2. *Théorème des persistants*

Dans cette section, nous présentons une amélioration de l'algorithme classique 1 de génération des transversaux minimaux. Nous avons vu que cet algorithme était itératif et que chaque itération était composée d'une étape de génération de transversaux candidats suivie d'une étape de suppression des candidats non minimaux. Le principe de l'amélioration proposée ici est d'éviter la deuxième étape en générant directement les transversaux minimaux. Le théorème 3.1 formalise cette idée. L'algorithme qui en découle est l'algorithme 2.

Supposons que l'on étudie l'itération k : à partir des transversaux de $\mathcal{H}$ générés à l'itération $k-1$ et à partir de l'arête e, nous devons générer les transversaux minimaux de $\mathcal{H}' = \mathcal{H} \cup e$ au cours de cette itération k. L'idée du théorème 3.1 est de classer les

transversaux minimaux obtenus à l'itération $k-1$ en plusieurs catégories. D'abord, il y a les « persistants », c'est-à-dire ceux qui ont une intersection non vide avec e : il est clair que ces transversaux, qui sont minimaux à l'itération $k-1$, le seront aussi à l'itération k (a. du théorème 3.1). C'est pour cette raison qu'on les nomme « persistants ». Parmi les persistants, nous distinguons les « n-persistants » qui ont au moins deux sommets en commun avec l'arête e, et les « 1-persistants » qui n'ont qu'un sommet en commun avec e. Ainsi, les éventuels nouveaux transversaux minimaux de l'itération k seront construits comme unions d'un sommet de e et d'un des autres transversaux minimaux de l'itération $k-1$, ceux qui ont une intersection vide avec e et que l'on nomme les « non-persistants ». Le théorème 3.1, ou « théorème des persistants », donne une condition nécessaire et suffisante (en b.) pour caractériser les non-persistants dont l'union avec un sommet c_j de e générera un transversal non minimal de l'itération k.

Théorème 3.1 (Théorème des persistants)
Soit $\mathcal{H}$ un hypergraphe et $Tr(\mathcal{H}) = \{X_i \mid i \in \{1,..,m\}\}$ l'ensemble de ses transversaux minimaux. Soit $e = \{c_j \mid j \in \{1,..,n\}\}$ une arête supplémentaire construite avec les sommets de $\mathcal{H}$. Soit $\mathcal{H}' = \mathcal{H} \cup e$.
$\forall i \in \{1,..,m\}$, on a :

a. Cas des persistants : Si $X_i \cap e \neq \emptyset$:
$\forall j \in \{1,..,n\}$:

- $(c_j \notin X_i \cap e) \rightarrow (X_i \cup \{c_j\}$ est un transversal de $\mathcal{H}'$ qui n'est pas minimal$)$
- $(c_j \in X_i \cap e) \rightarrow (X_i \cup \{c_j\} = X_i$ est un transversal minimal de $\mathcal{H}')$

b. Cas des non-persistants : Si $X_i \cap e = \emptyset$:
$\forall j \in \{1,..,n\}$:

$(X_i \cup \{c_j\}$ est un transversal de $\mathcal{H}'$ qui n'est pas minimal$)$ $\quad\leftrightarrow\quad$ $(\exists X_k \in Tr(\mathcal{H}) \mid X_k \cap e = \{c_j\}$ et $X_k \setminus \{c_j\} \subset X_i)$

Démonstration

Les principes de la démonstration, donnée en annexe, sont les suivants : le a) découle directement de la définition d'un transversal minimal d'un hypergraphe, et le b) provient des définitions d'un transversal minimal et d'un transversal non minimal, appliquées au membre de gauche de l'équivalence (l'explicitation de ces définitions par des manipulations algébriques simples permet de restreindre la propriété du membre de gauche exprimée sur $\mathcal{H}'$, à une propriété équivalente dans le membre de droite exprimée sur $\mathcal{H}$). □

L'algorithme 2 est une version de l'algorithme 1 utilisant le théorème 3.1 :

– ligne 1 : on initialise l'ensemble de transversaux minimaux avec tous les sommets formant une arête à eux seuls, car ces sommets se retrouvent au final dans tous les transversaux minimaux (puisqu'ils définissent certaines arêtes) ;

– ligne 2 : on itère sur les arêtes de cardinalité supérieure ou égale à 2 ;

– lignes 3 à 29 : génération des transversaux minimaux de l'itération courante :

- lignes 3 à 24 : implémentation du théorème 3.1 des persistants : aux lignes 6 et 7, on examine l'intersection de chaque transversal candidat avec l'arête courante ; aux lignes 8 à 11 et 21 à 24, on conserve les 1- et n- persistants ; et aux lignes 12, 13 et 16 à 20, on identifie les cas où un non-persistant générera un transversal non minimal à la fin de l'itération ;

- lignes 25 à 29 : on génère les nouveaux transversaux minimaux ;

– pas de phase de suppression des non-minimaux puisque l'on n'a généré aucun non-minimal d'après le théorème des persistants.

Algorithme 2 *Algorithme classique de génération des transversaux minimaux d'un hypergraphe amélioré avec le théorème 3.1 des persistants.*

Entrée(s) : Un hypergraphe simple $\mathcal{H} = (\Sigma, \Gamma)$.
Sortie(s) : $Tr(\mathcal{H})$

```
 1: Tr := {{v ∈ Σ | ∃e ∈ Γ | e = {v}}} ;
 2: pour chaque arête e ∈ Γ | |e| ≥ 2 faire
 3:    Tr_np := ∅ ;
 4:    Tr_1p := ∅ ;
 5:    Tr_notp := ∅ ;
 6:    pour chaque X ∈ Tr faire
 7:       I := e ∩ X ;
 8:       si |I| ≥ 2 alors
 9:          Tr_np := Tr_np ∪ {X} ;
10:       sinon si |I| = 1 alors
11:          Tr_1p := Tr_1p ∪ {(X, I)} ;
12:       sinon
13:          Tr_notp := Tr_notp ∪ {(X, e)} ;
14:       fin si
15:    fin pour
16:    pour chaque (X, I) ∈ Tr_1p faire
17:       pour chaque (Y, e) ∈ Tr_notp|X\_I ⊂ Y faire
18:          e := e\_I ;
19:       fin pour
20:    fin pour
21:    Tr := Tr_np ;
22:    pour chaque (X, I) ∈ Tr_1p faire
23:       Tr := Tr ∪ {X} ;
24:    fin pour
25:    pour chaque (Y, E) ∈ Tr_notp faire
26:       pour chaque vertex s ∈ E faire
27:          Tr := Tr ∪ {Y ∪ {s}} ;
28:       fin pour
29:    fin pour
30: fin pour
```

La vérification du théorème des persistants implique un nombre non négligeable de tests d'inclusion et d'opérations d'intersection dans la phase de génération des candidats. Cependant, elle permet d'éviter la phase de suppression des non-minimaux. La section 3.3 montre que, globalement, ce théorème implique un gain notable de performances sur l'ensemble des deux phases, par rapport à l'algorithme 1. C'est pourquoi on peut considérer ce théorème comme une amélioration de l'algorithme 1.

$H_1 = \{\{1,2\},\{3,4\},\{5,6\},\{7,8\},\{9,10\},\{11,12\},\{13,14\},\{15,16\},\{17,18\},$
$\{19,20\},\{21,22\},\{23,24\},\{25,26\}\}$
$H_2 = \{\{1,2,3,4\},\{3,4,5,6\},\{5,6,7,8\},\{7,8,9,10\},\{9,10,11,12\},$
$\{11,12,13,14\},\{13,14,15,16\},\{15,16,17,18\},\{17,18,19,20\},$
$\{19,20,21,22\},\{21,22,23,24\},\{23,24,25,26\},\{25,26,27,28\}\}$

Figure 7. *Deux pires cas : H_1 pour l'algorithme 1 et H_2 pour l'algorithme 2*

3.3. *Etude de complexité*

Dans [REY 03] est menée une étude de complexité détaillée, basée sur la construction de deux pires cas pour les algorithmes 1 et 2. Ces cas sont des hypergraphes, nommés H_1 et H_2, qui ont la particularité d'engendrer un nombre de transversaux minimaux exponentiel en fonction de leur taille, ainsi que de maximiser le nombre d'opérations élémentaires lors de l'exécution des algorithmes 1 et 2. Ces opérations élémentaires sont les tests d'inclusion et les calculs d'intersections ensemblistes d'une part, et les générations de transversaux d'autre part.

H_1 et H_2 sont donnés à la figure 7. H_1 est un pire cas pour l'algorithme 1 et H_2 pour les deux algorithmes. La figure 8 donne, en plus du nombre de transversaux générés à la fin de chaque itération, l'évolution du nombre de tests d'inclusion pour chaque algorithme, appliqué à H_1 et à H_2. On vérifie que, à l'instar du nombre de transversaux générés à chaque itération, le nombre de tests d'inclusion évolue de manière exponentielle, et que l'algorithme 2 est bien nettement plus performant que l'algorithme 1 même dans le cas de H_2 qui est un pire cas pour lui. La figure 9 résume les différents résultats théoriques obtenus dans [REY 03] pour les pires cas H_1 et H_2 des algorithmes 1 et 2, et donne ainsi un moyen de comparer l'efficacité théorique réciproque et au pire des deux algorithmes : là encore, on voit que l'algorithme 2 est meilleur. La section suivante évoque les principaux travaux concernant le problème de calcul des transversaux minimaux.

3.4. *Autres travaux sur les transversaux minimaux*

La recherche des transversaux minimaux d'un hypergraphe est centrale dans de nombreux domaines de l'informatique [EIT 95, EIT 02a]. En termes de complexité, certains hypergraphes prouvent que le nombre de transversaux minimaux peut être exponentiel en fonction de la taille de l'hypergraphe. Dès lors, la question qui se pose est de savoir s'il existe un algorithme polynomial en fonction de la somme des tailles de l'entrée (*i.e.* de la taille de l'hypergraphe) et de la sortie (*i.e.* la cardinalité de l'ensemble des transversaux minimaux). Ce problème est encore à notre connaissance un problème ouvert.

Dans [MIC 96], il est montré que la génération des transversaux minimaux d'un hypergraphe peut être effectuée en temps incrémental sub exponentiel (presque poly-

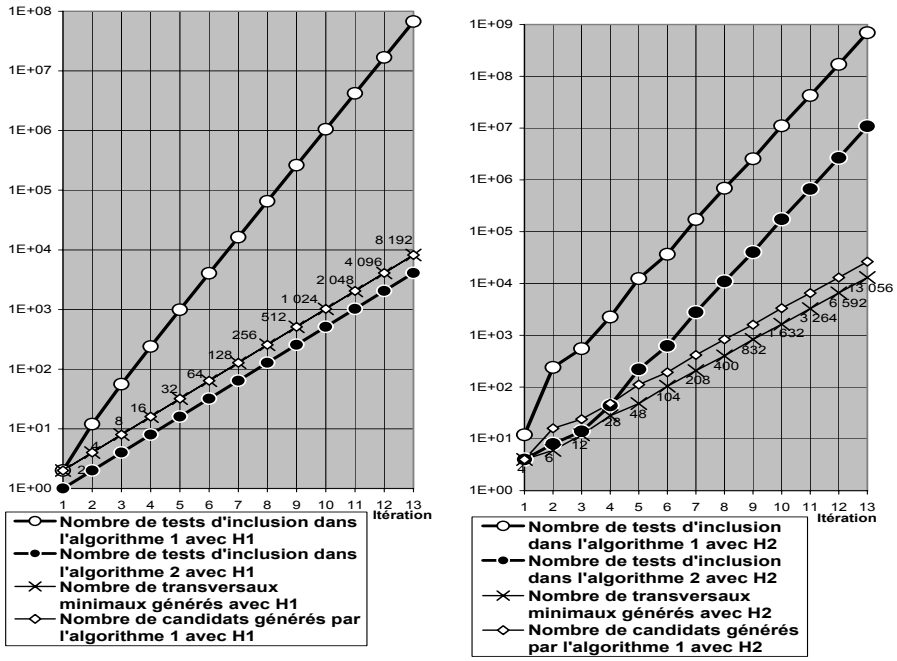

Figure 8. *Evolution des nombres de tests d'inclusion pendant l'exécution des algorithmes 1 et 2 sur H_1 et H_2*

A l'itération i	Nb. max de générations de transversaux	Nb. max de tests d'inclusion et de calculs d'intersection
Algorithme 1	$x_{i-1} * \|e_i\|$	$(x_{i-1} * \|e_i\|) * (x_{i-1} * \|e_i\| - 1)$
Algorithme 2	$\frac{x_{i-1}}{2} * \|e_i\|$	$x_{i-1} + \frac{x_{i-1}^2}{4}$

Figure 9. *Valeurs maximales à chaque itération, pour les algorithmes 1 et 2 dans, respectivement, les pires cas H_1 et H_2. x_{i-1} est le nombre de transversaux minimaux générés à l'itération $i-1$ et e_i est l'arête examinée à l'itération i ($|e_i|$ est sa cardinalité)*

nomial) $k^{O(log(k))}$, où k est la taille combinée de l'entrée et de la sortie. C'est, à notre connaissance, la meilleure borne théorique connue pour ce problème. D'autres travaux importants sur la complexité de ce problème sont présents dans [EIT 95, EIT 02b] : dans ces travaux, des classes d'hypergraphes sont identifiées pour lesquelles le problème est polynomial en fonction des tailles d'entrée et de sortie.

D'un point de vue pratique, plusieurs approches algorithmiques existent. Il y a d'abord celle que nous avons choisie qui se base sur l'algorithme 1 extrait de [EIT 95] et [MAN 94], lui-même fondé sur une propriété donnée dans [BER 89]. Notre algorithme 2 apparaît comme une amélioration de cet algorithme de base.

Une autre amélioration est proposée dans [KAV 99] et prend la forme, à l'instar du théorème 3.1, d'une condition nécessaire et suffisante de minimalité pour un transversal formé d'un non-persistant et d'un sommet de l'arête à ajouter. Une étude comparative avec le théorème 3.1 est en cours de réalisation. Les premiers résultats concernant notamment le cas très mauvais H_2 vu précédemment montrent que cette condition serait un peu plus performante que celle du théorème 3.1 car H_2 est un petit hypergraphe possédant beaucoup de transversaux. Cependant, dans des cas moins particuliers, il semble que la vérification de la condition puisse être assez coûteuse. Concernant l'implémentation, cette condition présente l'avantage de permettre la découverte des transversaux en profondeur et non plus en largeur : on n'a pas besoin de calculer tous les transversaux minimaux d'un hypergraphe, pour ensuite pouvoir calculer les transversaux minimaux du même hypergraphe avec une arête supplémentaire. Cela permet de réduire l'espace mémoire utilisé. De plus, les transversaux minimaux ne sont plus donnés en fin de processus, mais égrénés tout au long du calcul, permettant ainsi d'avoir les premières solutions très rapidement.

Deux autres approches ont été proposées. Dans [LOP 00], une stratégie de calcul par niveaux dans le treillis des parties est utilisée. Cette technique apparaît comme assez efficace si la cardinalité moyenne des transversaux minimaux est réduite. Dans les autres cas, elle est très coûteuse en temps de calcul. Dans [WYS 01], c'est une énumération des sommets, basée sur l'ordre lectique, qui permet de parcourir en profondeur le treillis des parties de l'ensemble des sommets de l'hypergraphe et ainsi de découvrir les transversaux minimaux. Par rapport à l'approche précédente, les résultats en pratique sont meilleurs car le parcours en profondeur permet une utilisation plus réduite de la mémoire.

3.5. *Séparation et évaluation (Branch and Bound)*

Nous présentons maintenant comment nous avons adapté l'algorithme de calcul des transversaux minimaux au problème de calcul des transversaux minimaux de coût minimal pour l'hypergraphe $\mathcal{H}_{\mathcal{T}Q}$ construit à partir d'une ontologie $\mathcal{T}$ de services et d'une requête Q. L'algorithme 3 *computeBCov*, résultat de cette adaptation, est donné à la fin de cette section. Son implémentation et les tests qu'il a permis de réaliser sont étudiés aux sections 4 et 5.

En partant de l'algorithme 1 ou 2, l'approche naïve, pour calculer les transversaux minimaux de coût minimal, est de calculer tous les transversaux minimaux, puis de les trier afin de ne conserver que ceux qui ont un coût minimal. Au lieu de cela, nous utilisons la structure itérative de ces algorithmes pour, à chaque itération, supprimer les transversaux qui sont minimaux pour cette itération mais dont on est sûr qu'ils ne généreront aucun transversal minimal pour l'hypergraphe entier. Notons ici que le coût d'un ensemble de sommets est donné par la taille du *miss* de la conjonction de services correspondante (voir [HAC 02a]).

Pour ce faire, l'idée principale de *computeBCov* est d'utiliser une technique d'optimisation combinatoire nommée « séparation et évaluation » (ou Branch and Bound, « BnB », en anglais), pour élaguer des branches de l'arbre de recherche et ainsi se diriger plus rapidement vers les solutions. Au début de *computeBCov* (ligne 3), une heuristique simple est utilisée pour calculer le coût d'un transversal (pas forcément minimal) de l'hypergraphe entier. Ce coût est *a priori* assez faible : il est obtenu en ajoutant, pour chaque arête de l'hypergraphe, le coût du sommet de plus petit coût. Le coût total obtenu par l'heuristique est stocké dans la variable $CostEval$. Comme nous avons, pour tout ensemble de sommets $X = \{V_{S_i}\}$:

$$cout(X) \leq \sum_{V_{S_i} \in X} cout(\{V_{S_i}\})$$

alors la valeur stockée dans $CostEval$ est une borne supérieure du coût d'un transversal minimal de l'hypergraphe entier. Comme l'ajout d'un sommet à un ensemble de sommets fait augmenter son coût (au mieux de 0), on est sûr que tous les transversaux minimaux intermédiaires (*i.e.* les transversaux minimaux de l'hypergraphe privé d'au moins une arête), possédant un coût strictement supérieur au coût de $CostEval$, ne généreront que des transversaux minimaux de l'hypergraphe entier ayant un coût strictement supérieur à $CostEval$. Comme le coût stocké dans $CostEval$ est celui d'un transversal de l'hypergraphe entier, on peut supprimer ces transversaux minimaux intermédiaires tout de suite (juste après leur génération). Ainsi, à chaque itération, après l'étape de génération des transversaux minimaux (ligne 5), nous éliminons ceux qui ont un coût strictement supérieur à $CostEval$ (lignes 8 et 9). Ensuite, à partir de chaque transversal minimal restant, nous calculons dans une nouvelle évaluation du coût d'un transversal (pas forcément minimal) de l'hypergraphe entier construit à partir du transversal examiné (ligne 11). Si l'une de ces nouvelles évaluations est strictement plus petite que $CostEval$ alors $CostEval$ prend sa valeur (lignes 12 et 13).

Pour calculer une nouvelle évaluation du coût d'un transversal de l'hypergraphe entier construit à partir d'un transversal minimal intermédiaire (ligne 11), nous avons deux politiques possibles dans le Branch and Bound. La politique 1 (ou BnB1) est l'utilisation du principe de l'heuristique qui calcule $CostEval$ au début de l'algorithme, *i.e.* qu'au coût du transversal intermédiaire (stocké dans $RealCost$), on ajoute le coût du sommet de plus petit coût pour chaque arête qui n'a pas été encore examinée. La politique 2 (ou BnB2) consiste à ajouter, au coût du transversal intermédiaire (stocké dans $RealCost$), le coût du sommet de plus petit coût pour chaque arête qui n'a pas été encore examinée et qui a une intersection vide avec le transversal intermédiaire.

C'est la politique 2 qui est utilisée dans l'algorithme 3. En effet, elle apparaît *a priori* comme plus performante puisque ses évaluations sont toujours au moins aussi bonnes que celles de la politique 1, et donc le BnB converge d'autant plus rapidement vers les solutions. Cependant, elle nécessite quelques calculs d'intersection supplémentaires. Comme il est difficile de se faire une idée sur le gain relatif dû à une meilleure évaluation par rapport au coût supplémentaire dû à des calculs d'intersections, nous avons implémenté les deux politiques (voir la section 4). En pratique, on a montré (voir le chapitre 5) que la politique 2 est meilleure que la politique 1.

A la fin de l'algorithme, chaque transversal minimal de l'hypergraphe entier est traduit en la solution correspondante du problème de recherche des meilleures combinaisons de services pour une requête étant donné une ontologie.

Algorithme 3 *computeBCov pour le calcul des meilleures couvertures d'un concept en utilisant une ontologie, pour le langage* $\mathcal{FL}_0$.

Entrée(s) : Une ontologie $\mathcal{T}$ de services et une requête Q exprimées dans la logique de description $\mathcal{FL}_0$.

Sortie(s) : L'ensemble des meilleures combinaisons de services répondant à Q étant donné l'ontologie $\mathcal{T}$.

1: Construction de l'hypergraphe $\mathcal{H}_{\mathcal{T}Q} = (\Sigma, \Gamma)$
2: $Tr \leftarrow \emptyset$.
3: $CostEval \leftarrow \sum_{e \in \Gamma} cout$ du sommet V_{S_i} de plus petit $cout$ dans e ;
4: **pour chaque** arête $e \in \Gamma$ **faire**
5: $Tr \leftarrow \{$transversaux minimaux générés selon l'algorithme 2$\}$
6: **pour chaque** transversal minimal $X \in Tr$ **faire**
7: $RealCost \leftarrow cout(X)$;
8: **si** $RealCost > CostEval$ **alors**
9: $Tr \leftarrow Tr \setminus X$;
10: **sinon si** $RealCost < CostEval$ **alors**
11: $Eval \leftarrow RealCost + \sum_{f \in \Gamma \mid f \cap X = \emptyset} cout$ du sommet V_{S_i} de plus petit $cout$ dans f ;
12: **si** $Eval < CostEval$ **alors**
13: $CostEval \leftarrow Eval$;
14: **fin si**
15: **fin si**
16: **fin pour**
17: **fin pour**
18: **pour chaque** $X \in Tr$ tel que $cout(X) = CostEval$ **faire**
19: Retourner le concept E_X.
20: **fin pour**

En résumé, l'algorithme 3 est structuré ainsi :

– ligne 1 : création de l'hypergraphe,

– ligne 2 : initialisation de l'ensemble des transversaux minimaux de coût minimal à l'ensemble vide,

– ligne 3 : on calcule une première évaluation du coût d'un transversal par l'heuristique vue précédemment,

– lignes 4 à 17 : on itère sur les arêtes de l'hypergraphe,

- ligne 5 : on génère les transversaux minimaux selon l'algorithme 2,

- lignes 6 à 9 : parmi les transversaux miminaux obtenus, on ne conserve que ceux qui ont un coût inférieur à l'évaluation courante,

- lignes 10 à 15 : on met éventuellement à jour l'évaluation courante,

– lignes 18 à 20 : parmi les solutions obtenues, on ne conserve que celles de coût minimal (cas où l'évaluation a été modifiée au cours de la dernière itération).

4. Le prototype D^2CP

Dans cette partie, nous présentons le prototype Java dans lequel est implémenté *computeBCov*. Nous l'avons appelé D^2CP pour « Dynamic Discovery of Concepts Prototype », ou prototype de découverte dynamique de concepts.

Le but de ce système est d'être une plate-forme de tests de l'algorithme *computeBCov* de découverte dynamique de services web. Pour ceci, D^2CP permet de découvrir dynamiquement les meilleures combinaisons de services web étant donné une requête Q et une ontologie $\mathcal{T}$ grâce aux 6 variantes possibles de *computeBCov*, de générer aléatoirement des ontologies de services web et des requêtes associées sous la forme de fichiers XML, et de visualiser sous différentes formes les résultats de la découverte dynamique et les mesures de performances. La figure 10 résume ces fonctionnalités et la figure 11 montre l'interface graphique utilisateur de D^2CP.

D^2CP permet à l'utilisateur d'exécuter 6 variantes de *computeBCov* pour effectuer la découverte dynamique de services web. En effet, l'utilisateur peut choisir d'utiliser ou non le théorème des persistants pour générer les transversaux minimaux (c'est-à-dire utiliser soit l'algorithme 2 soit l'algorithme 1), d'utiliser ou non le BnB pour trouver les transversaux de coût minimal et d'utiliser la politique 1 (BnB1) ou la politique 2 (BnB2) s'il utilise le BnB. Les six variantes de *computeBCov* sont résumées à la figure 12 où elles sont ordonnées selon leur efficacité.

Le module de génération d'ontologies de D^2CP permet de générer des ontologies $\mathcal{FL}_0$ sous la forme de fichiers XML, à partir d'un fichier DTD qui décrit la structure de l'ontologie à générer. Les ontologies générées contiennent une description de concept particulière (la requête), un ensemble de descriptions de concepts (les services) et un autre ensemble de descriptions de concepts (les autres concepts définis) qui sont utilisés dans la description des services. La génération est divisée en deux étapes : à partir du fichier DTD fourni par défaut ou modifié par l'utilisateur, et à partir d'autres paramètres entrés *via* son interface graphique, D^2CP va générer un squelette d'ontologie grâce au générateur IBM XML Generator v1r8[5], puis, il génère les noms des éléments XML afin d'obtenir une ontologie $\mathcal{FL}_0$ sans cycle pouvant être traitée par *computeBCov*. La figure 13 représente graphiquement une partie d'une ontologie XML générée par D^2CP. Les paramètres de génération d'ontologie ajustables par l'utilisateur sont donnés à la figure 14.

L'utilisateur peut choisir le type d'affichage (voir la figure 10) qu'il souhaite parmi un affichage textuel simple des meilleures combinaisons de services web, une trace arborescente de chaque exécution, didactique et utile pour vérifier le bon déroulement de chaque variante exécutée, et un affichage des statistiques d'exécution qui sont des mesures de chaque étape de *computeBCov* destinées à identifier les performances respectives de chaque variante. Ces statistiques peuvent être présentées sous la forme d'un histogramme ou d'un tableau de valeurs, et sont par ailleurs stockées dans un fichier HTML afin de servir d'entrée à un tableur.

---

5. Voir http ://www.alphaworks.ibm.com/tech/xmlgenerator.

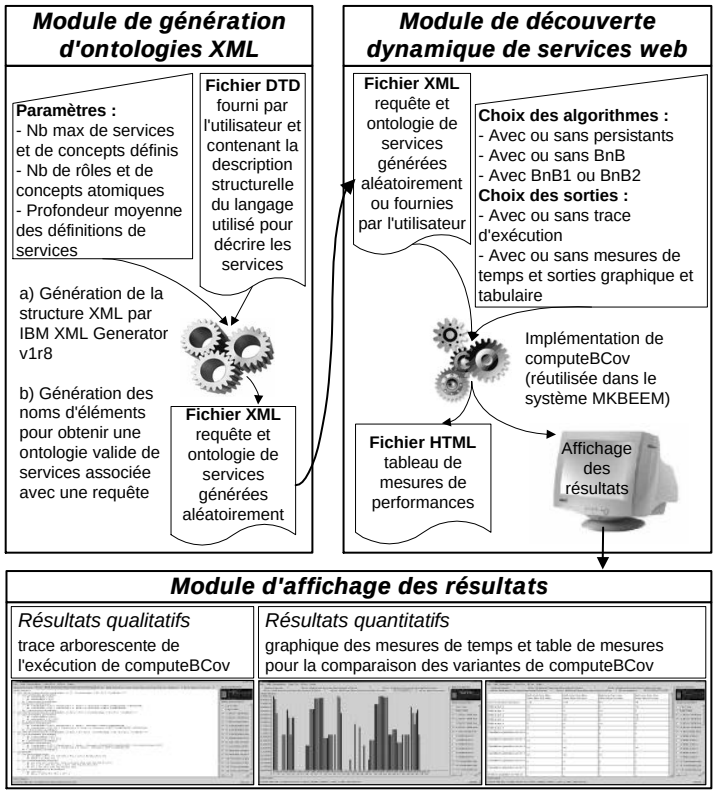

Figure 10. *Vue d'ensemble des fonctionnalités de D^2CP*

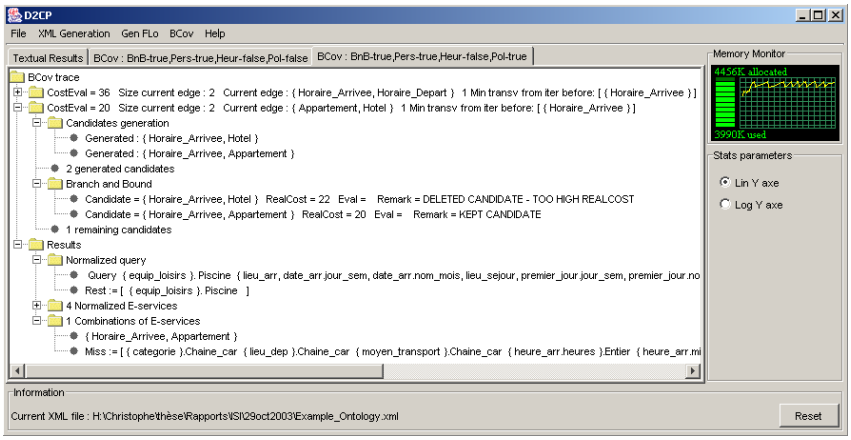

Figure 11. *L'interface graphique utilisateur de D^2CP : on peut voir la trace d'exécution de computeBCov sur l'exemple de la section 2*

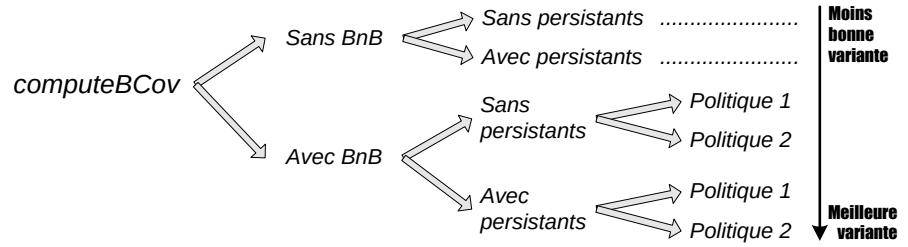

Figure 12. *Les 6 variantes de computeBCov ordonnées selon leur efficacité*

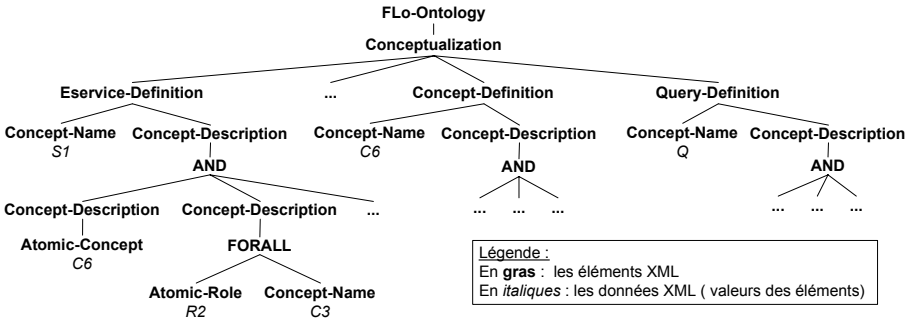

Figure 13. *Représentation graphique d'un fichier XML généré par D^2CP et contenant une ontologie $\mathcal{FL}_0$*

Nature du paramètre	Comment le modifie-t-on ?
Le nombre maximal de définitions de concepts (services et autres).	Via l'interface de D^2CP qui modifie ensuite un paramètre du générateur XML d'IBM.
La proportion de définitions de services par rapport aux autres définitions de concepts.	Dans le fichier DTD en ajoutant ou otant des éléments *Concept-Definition* comme fils de l'élément *Conceptualization*.
Le nombre maximal de clauses dans les descriptions de concepts (i.e. le nombre maximal d'éléments fils de l'élément *AND* dans le fichier XML).	Dans le fichier DTD, en ajoutant ou en otant des éléments *Concept-Description* comme fils de l'élément *AND*.
La proportion d'éléments *Atomic-Concept* par rapports aux éléments *FORALL* dans chaque description de concept.	Dans le fichier DTD en ajoutant ou otant des éléments *FORALL* comme fils de l'élément *Concept-Description*.
Le nombre total de concepts atomiques utilisés dans l'ontologie.	Via l'interface de D^2CP.
Le nombre total de rôles atomiques utilisés dans l'ontologie.	Via l'interface de D^2CP.
La proportion de concepts définis (non services) réutilisés pour définir d'autres concepts (services ou non services).	Via l'interface de D^2CP.

Figure 14. *Paramètres réglables pour la génération d'ontologies XML dans D^2CP*

5. Expérimentations

Dans cette partie, nous présentons les expérimentations qui ont été menées avec D^2CP et qui ont permis de valider la découverte dynamique de services web ainsi que de déterminer les variantes les plus efficaces de *computeBCov*. Les tests présentés dans cette partie, sauf en ce qui concerne la validation MKBEEM, ont été effectués sur une machine de type PC, avec un processeur Intel Pentium 3 à 500 MHz et 320 Mo de RAM.

5.1. *Validation qualitative de D^2CP au sein du projet MKBEEM*

Les classes Java qui implémentent les différentes variantes de *computeBCov* et qui constituent le coeur de D^2CP ont été utilisées dans le cadre du projet MKBEEM[6], un médiateur pour le e-commerce dont le but est d'être le support de marchés électroniques offrant des services multilingues et intelligents, basés sur des connaissances regroupées en ontologies, dans le domaine du tourisme et de la vente par correspondance. L'exemple de la section 2 est un extrait simplifié des scenarii rendus possibles par le prototype MKBEEM. Ces scenarii, de type vente en ligne et services web de voyage/tourisme, ont permis de valider le système, et entre autres la découverte dynamique de services, à une échelle européenne (France et Finlande) et en trois langues (finlandais, anglais et français).

Dans les premières expériences menées avec le prototype MKBEEM, nous avons modélisé de petites ontologies (environ 500 concepts définis et 50 services web) sur le domaine du tourisme notamment. Le traitement des requêtes utilisateurs s'est avéré qualitativement satisfaisant, les combinaisons de services web découverts étaient pertinentes, et quantitativement intéressant, le temps total dédié à la découverte n'a jamais dépassé quelques secondes. Les tests n'ont pas encore été menés sur des ontologies de tailles plus réalistes pour des raisons de temps.

Après ces tests qualitatifs, nous avons poursuivi les expérimentations de *computeBCov* de deux façons. D'abord, nous avons testé les algorithmes 1 et 2 sur les pires cas théoriques étudiés à la section 3.3, afin d'évaluer l'efficacité au pire de la génération des transversaux minimaux, étape principale de *computeBCov*. Puis, nous avons généré des ontologies aléatoires de plus grandes tailles pour le passage à l'échelle de *computeBCov*. Les résultats sont présentés dans les sections suivantes.

5.2. *Pires cas dans la génération des transversaux*

Nous avons d'abord exécuté les algorithmes 1 et 2 sur H_1 et H_2 dans D^2CP. Les résultats de ces exécutions sont donnés à la figure 15. La figure 15 résume les

6. MKBEEM est l'acronyme de Multilingual Knowledge Based European Electronic Marketplace (IST-1999-10589, 1er Fév. 2000 - 1er Déc. 2002). Voir http ://www.mkbeem.com

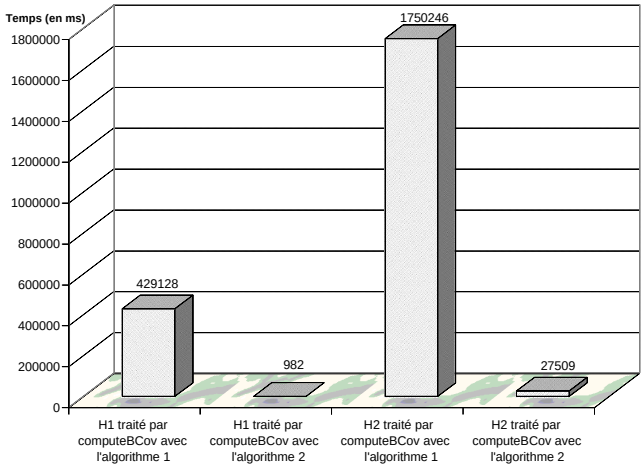

Figure 15. *Temps total d'exécution de computeBCov pour H_1 et H_2 par D^2CP*

temps totaux d'exécution des variantes de *computeBCov* utilisant l'algorithme 1 ou l'algorithme 2 sur H_1 et H_2. Ces temps confirment les évaluations théoriques des nombres de tests d'inclusion à chaque itération données à la figure 8 : l'algorithme 2 est meilleur pour générer les transversaux minimaux d'un hypergraphe. Puis nous avons construit, sur le modèle de H_2, des hypergraphes de tailles croissantes pour tester l'algorithme 2 sur des instances de plus en plus grandes de pires cas. La figure 16 donne les temps globaux d'exécution de *computeBCov* utilisant l'algorithme 2, sur ces instances de tailles croissantes. On obtient ainsi une borne supérieure pour les temps de calculs de *computeBCov* pour des hypergraphes de taille équivalente ou ayant un nombre équivalent de transversaux minimaux. Ainsi, on voit que pour un très mauvais cas (grand nombre de solutions et maximisation du nombre d'opérations élémentaires nécessaires), il faut environ une seconde à la meilleure variante de *computeBCov* pour résoudre une instance du problème possédant jusqu'à 3264 solutions, et il faut entre 1 et 20 secondes à la même variante pour résoudre une instance possédant jusqu'à 13056 solutions.

Ces conclusions peuvent servir de repère pour se faire une idée sur la pertinence des résultats qui seront donnés dans la section 5.3. Il y a cependant quelques nuances à apporter :

– On a identifié dans les cas de H_1 et H_2 les meilleures combinaisons de services aux transversaux minimaux. Ceci est dû au fait que l'on a construit ces hypergraphes sans se préoccuper des ontologies qui auraient dû être à leur origine. Ainsi, on teste ici plus les algorithmes de calculs des transversaux minimaux que *computeBCov* : le coût du Branch and Bound n'est pas évalué, et pourtant il n'est pas négligeable, comme on le verra par la suite. Cette démarche se justifie par le fait que c'est bien la recherche des transversaux minimaux qui donne sa complexité au problème de base. En effet,

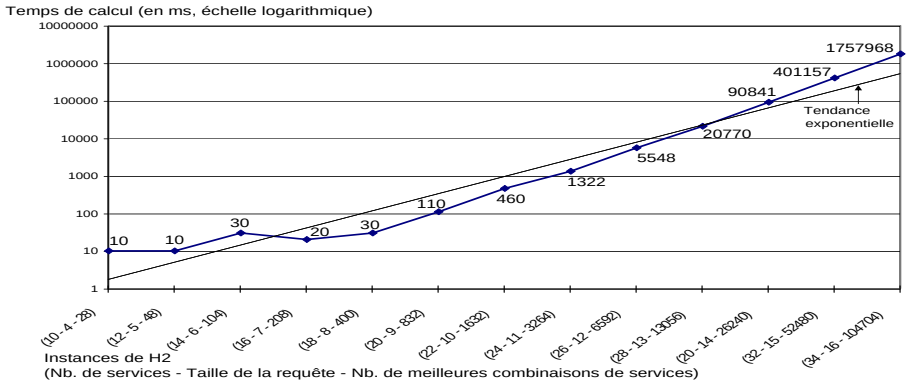

Figure 16. *Temps total d'exécution de l'algorithme 2 appliqué à différentes instances de H_2 (i.e. à des hypergraphes ayant la même structure et le même comportement que H_2 dans l'algorithme 2 mais avec des tailles variables)*

si le Branch and Bound possède un coût non négligeable en lui-même, il est là pour améliorer la rapidité de l'obtention des solutions et il implique donc une baisse du coût total de l'algorithme.

– Les exemples de H_1, H_2 et des différentes instances de H_2 ont comme caractéristique d'être de petits exemples (moins de 40 services), avec un grand nombre de solutions (jusqu'à 100 000, voir la figure 16). Or l'expérience de MKBEEM nous a montré que des exemples plus réalistes seraient plutôt de grands exemples (au moins 500 services et 2000 concepts définis) avec un petit nombre de solutions (moins de 10). On peut donc s'interroger sur la pertinence de ces exemples. Ce qui justife cette démarche est le fait que l'on veut étudier les pires cas possibles pour *computeBCov*. Comme c'est la recherche des transversaux minimaux qui donne sa complexité au problème, il fallait donc bien étudier ces cas. Ainsi, on travaille sur des cas dont on sait qu'ils ne sont pas réalistes, mais dont on sait aussi, et surtout, qu'ils sont *a priori* bien pires que des cas réalistes. De plus, ils nous permettent de voir comment se comporte *computeBCov* dans la manipulation d'un grand nombre de solutions, ce qui peut tout à fait arriver *au cours de* l'exécution de *computeBCov* pour une instance du problème avec très peu de solutions au final.

Ainsi, il reste à tester *computeBCov* sur des exemples dans lesquels le Branch and Bound a un impact (son propre coût et le gain qu'il induit) et sur des exemples plus réalistes, en tout cas de plus grandes tailles avec un nombre réduit de solutions. C'est ce qui est fait dans la section suivante.

Caractéristique	Ontologie repère	Cas 1	Cas 2	Cas 3
Nombre de concepts définis	2500	365	1334	3405
Nombre de services web	500	366	660	570
Nombre de clauses dans la requête	15	6	33	12
Nombre moyen de services web pour une clause (i.e. de services web dans lesquels se retrouve une clause de la requête)	20	10,83	20,84	30,75
Nombre de solutions	moins de 10	1	2	1
Nombre de concepts atomiques	150	13	30	12
Nombre de rôles atomiques	100	13	30	12
Proportion de concepts définis réutilisés dans une définition de concept par rapport aux concepts atomiques	30 %	30 %	20 %	33 %

Figure 17. *Caractéristiques de l'ontologie repère et des trois cas générés avec* D^2CP

Ontologie	Description
Cas 1	Assez différente de l'ontologie repère de par sa petite taille, petite requête, cas très défavorable pour *computeBCov*.
Cas 2	Proche de l'ontologie repère mais de petite taille, requête de taille normale, cas défavorable pour *computeBCov*.
Cas 3	Très proche de l'ontologie repère, petite requête, cas très défavorable pour *computeBCov*.

Figure 18. *Résumé des trois ontologies générées avec* D^2CP

5.3. *Ontologies synthétiques*

Parmi toutes les ontologies que nous avons générées grâce à D^2CP, nous en avons selectionné trois qui sont représentatives de tous les cas de figure que l'on a pu observer. Après avoir présenté ces trois cas, nous discutons des temps d'exécution de chaque variante sur chacun des cas.

Les caractéristiques de ces trois cas sont résumées à la figure 17. Pour les étudier, nous considérons une « ontologie repère » qui représente une ontologie possédant des caractéristiques qui nous semblent se rapprocher d'une ontologie réaliste[7]. La figure 18 positionne chaque cas par rapport à l'ontologie repère. Pour établir si le cas est favorable ou non par rapport à *computeBCov*, on raisonne sur les nombres de concepts et rôles atomiques : plus ces valeurs sont petites, plus ces concepts et rôles atomiques auront été réutilisés durant la génération aléatoire. Ceci implique une combinatoire plus grande, et donc une découverte dynamique plus lente *a priori*.

Les temps globaux d'exécution des 6 variantes de *computeBCov* dans D^2CP sont donnés à la figure 19. Celle-ci montre que, pour les cas 1 et 3, il existe au moins une variante qui effectue la découverte dynamique en moins de deux secondes, et en 30 secondes pour le cas 2. Ces tests montrent aussi qu'il y a une grande différence de performance suivant la variante utilisée. Pour résumer, nous pouvons dire d'après la figure 19 que :

7. Ces valeurs prennent en compte l'expérience acquise avec le système MKBEEM. Elles restent arbitraires et sujettes à ajustements.

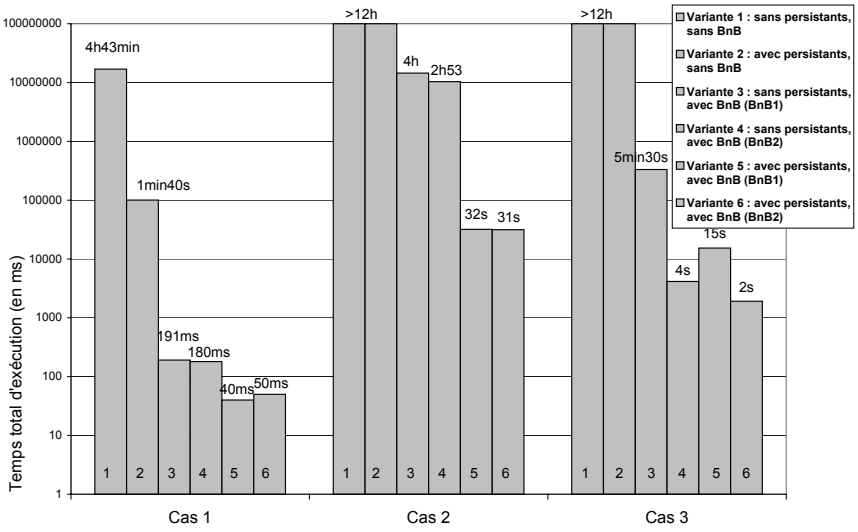

Figure 19. *Temps global d'exécution, en millisecondes, de chaque variante de computeBCov pour les 3 cas d'étude dans* D^2CP

– l'optimisation par le Branch and Bound est très efficace : pour le cas 1, même sans les persistants, le BnB permet d'obtenir une solution en temps réel (moins d'une seconde), et pour les cas 2 et 3, le BnB permet d'envisager l'obtention d'une solution dans un temps limité sans les persistants, voire presque en temps réel s'il est associé aux persistants. Sans le BnB, ces cas sont intraitables (plus de 12 heures). L'effet espéré du Branch and Bound, qui est de limiter l'explosion combinatoire du problème, est donc bien présent. Cet effet est encore accentué par l'usage de BnB2 au lieu de BnB1 : comme BnB2 permet une meilleure évaluation d'une solution possible à chaque itération, le BnB implémenté avec BnB2 élague plus de branches dans l'arbre des combinaisons ;

– les persistants sont aussi très intéressants, mais uniquement associés avec le BnB. Ils n'ont pas d'effet sur le nombre de combinaisons explorées, et donc ne limitent pas l'explosion combinatoire, mais ils accélèrent très sensiblement la génération des combinaisons à explorer. Dans le cas 2, associés avec le BnB, ils permettent l'obtention des solutions en un temps raisonnable (environ 30 secondes). Dans le cas 3, ils permettent l'obtention des solutions presque en temps réel (quelques secondes).

Ainsi, la variante la plus performante de *computeBCov* est celle qui associe le BnB implémenté avec la méthode BnB2 et les persistants. C'est cette variante qui a été utilisée pour valider la découverte de services dans MKBEEM. Les cas étudiés étant de tailles respectables, proches de l'ontologie repère et de nature assez défavorable pour *computeBCov*, on peut être optimiste quant aux performances de D^2CP sur des ontologies réelles qui restent à construire.

6. Travaux associés

Dans cette section, nous reprenons les approches existantes de découverte étudiées à la section 2.2, et nous comparons leurs implémentations avec *computeBCov* et D^2CP. Les critères de comparaison sont la nature de l'algorithme de découverte (description générale du fonctionnement de l'algorithme), le système dans lequel l'algorithme est implanté et les résultats d'expérimentations. Les résultats précis de cette comparaison sont donnés à la figure 20.

La conclusion générale de cette comparaison est la suivante : étant donné le fait que l'espace de recherche que l'on parcourt est exponentiel en fonction du nombre de services (c'est l'ensemble des parties de l'ensemble des services), les performances de *computeBCov* sont *a priori* moins bonnes que dans les autres algorithmes où l'espace de recherche est linéaire en fonction du nombre de services. Cela se vérifie ainsi avec l'algorithme de [LI 03]. Cependant, les résultats de nos expérimentations sont très encourageants puisque de l'ordre de la dizaine de secondes, voire de la seconde, pour des tailles déjà assez importantes (570 services et 3400 concepts).

7. Conclusion

Dans cet article, nous avons présenté l'algorithme *computeBCov* qui permet la découverte dynamique de combinaisons de services web, étant donné une ontologie de services web et une requête stockées dans un fichier XML, et le système D^2CP qui l'implémente. Qualitativement, D^2CP a été validé au sein du projet européen MKBEEM. Quantitativement, les résultats des pires cas et des cas générés par D^2CP sont très encourageants et de bon augure pour le passage à l'échelle dans des conditions réalistes.

Actuellement, nous travaillons sur l'extension du problème à la LD $\mathcal{ALN}$ qui permet notamment d'exprimer l'inconsistance entre concept, ce qui augmente l'expressivité par rapport aux langages à subsomption structurelle. Parallèlement, nous construisons une nouvelle interface permettant par exemple à un utilisateur de définir, modifier ou supprimer facilement de nouveaux services et requêtes, et ainsi d'en voir les effets sur les résultats de *computeBCov*. Le but est aussi de permettre la construction plus conviviale d'ontologies réalistes en taille et en contenu, dans un ou plusieurs domaines, pour tester les performances de D^2CP dans des conditions réelles. Enfin, nous étudions comment relier D^2CP à un registre UDDI et à des services web décrits en DAML-S.

Le prototype D^2CP est à notre connaissance un des premiers systèmes à proposer une implémentation aussi poussée d'un processus de découverte dynamique de services web, et le seul qui permet la découverte de combinaisons de services web. C'est ainsi un des premiers systèmes qui prouve concrètement le fort potentiel de l'approche web sémantique dans le traitement des services web.

	Nature de l'algorithme de découverte
computeBCov	Recherche des transversaux minimaux de coût minimal dans un hypergraphe. Espace de recherche exponentiel en fonction du nombre de services (toutes les combinaisons possibles de services).
[GON 01, TRA 02]	Pour chaque service, on cherche son degré de correspondance avec la requête. Espace de recherche linéaire en fonction du nombre de services.
[LI 03]	
[PAO 02]	
Conclusion	*computeBCov* explore plus de solutions pour une plus grande variété et pertinence dans les résultats.
	Système dans lequel est implanté l'algorithme
computeBCov	D^2CP, dédié à la génération aléatoire d'ontologies et aux expérimentations de *computeBCov*. D^2CP est écrit en Java.
[GON 01, TRA 02]	Prototype de découverte basé sur le système Fact, un des principaux systèmes implémentant les raisonnement de base des LD (subsomption et satisfiabilité pour la LD $\mathcal{SHIQ}$). Fact est écrit en Lisp.
[LI 03]	Un système multiagent, basé sur la plateforme Jade et sur le système Racer, un autre système reconnu implémentant les raisonnements de base pour la LD $\mathcal{SHIQ}$. Racer est aussi écrit en Lisp. Ce système multiagent est dédié à l'étude de la description des services à partir d'ontologies et à l'étude de la découverte de services exprimés avec les LD.
[PAO 02]	Un moteur de découverte DAML-S utilisé avec un registre UDDI pour apporter une fonctionnalité de découverte de services basée sur un critère sémantique. Ce moteur est basé sur un moteur d'inférences DAML+OIL (DAML+OIL est un langage issu de la LD $\mathcal{SHIQ}$).
Conclusion	D^2CP est bien un des premiers systèmes de découverte de services web dans le contexte du web sémantique : comme les trois autres systèmes, il utilise les logiques de description pour fonder la découverte sur un critère sémantique.
	Résultats d'expérimentations
computeBCov	Etude qualitative au sein du projet MKBEEM. Etude quantitative du comportement au pire et du passage à l'échelle sur des ontologies générées aléatoirement. Résultats de l'ordre de la dizaine de secondes, voire de la seconde, dans des cas de taille assez importante.
[GON 01, TRA 02]	Pas à notre connaissance.
[LI 03]	Etude quantitative sur des ontologies générées aléatoirement contenant de 100 à 1500 services. Résultats de l'ordre de quelques dizaines de millisecondes.
[PAO 02]	Pas à notre connaissance.
Conclusion	Les performances de l'approche de [LI 03] ne sont pas surprenantes : l'espace de recherche étant linéaire en fonction du nombre de services (et non exponentiel), la découverte s'effectue bien plus rapidement. La différence avec *computeBCov*, difficile à évaluer sans données supplémentaires, paraît se situer au niveau d'un facteur 100, ce qui n'est pas mauvais au vu des objectifs et des critères des deux découvertes.

Figure 20. *Comparaison avec les systèmes existants*

8. Bibliographie

[ANK 02] ANKOLEKAR A., ET AL, « DAML-S : Web Service Description for the Semantic Web », *Proceedings of the 1st Int'l Semantic Web Conf. (ISWC 02)*, 2002.

[BAA 00a] BAADER F., KÜSTERS R., MOLITOR R., « Rewriting Concepts Using Terminologies », *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning (KR2000)*, San Francisco, CA, 2000, p. 297-308.

[BAA 00b] BAADER F., KÜSTERS R., MOLITOR R., « Rewriting Concepts Using Terminologies – Revisited », Report n° 00-04, 2000, LTCS, RWTH Aachen, Germany.

[BAA 03] BAADER F., CALVANESE D., MCGUINNESS D., NARDI D., PATEL-SCHNEIDER P., Eds., *The Description Logic Handbook. Theory, Implementation and Applications*, Cambridge University Press, January 2003.

[BEN 03] BENATALLAH B., HACID M.-S., REY C., TOUMANI F., « Request Rewriting-Based Web Service Discovery », *Proceedings of the Second International Semantic Web Conference (ISWC)*, vol. 2870 de *LNCS*, Sanibel Island, FL, USA, October 2003, Springer, p. 242-257.

[BER 89] BERGE C., *Hypergraphs*, vol. 45 de *North Holland Mathematical Library*, Elsevier Science Publishers B.V. (North-Holland), 1989.

[BER 02] BERNSTEIN A., KLEIN M., « Discovering Services : Towards High-Precision Service Retrieval. », *Proceedings of the Web Services, E-Business, and the Semantic Web, CAiSE 2002 International Workshop, WES 2002*, vol. 2512 de *Lecture Notes in Computer Science*, Toronto, Canada, May 2002, Springer, p. 260-276.

[CAS 01a] CASATI F., SHAN M.-C., « Dynamic and adaptive composition of e-services », *Information Systems*, vol. 26, n° 3, 2001, p. 143-163.

[CAS 01b] CASATI F., SHAN M.-C., « Models and Languages for Describing and Discovering E-Services », *Proceedings of SIGMOD 2001*, Santa Barbara, USA, May 2001.

[CAS 01c] CASATI F., SHAN M.-C., GEORGAKOPOULOS D., Eds., *The VLDB Journal : Special Issue on E-Services*, 2001.

[COR 03] CORCHO O., GOMEZ-PEREZ A., LÉGER A., REY C., TOUMANI F., « An ontology-based mediation architecture for E-commerce applications », *Proceedings of Intelligent Information Systems 2003*, Zakopane, Poland, June 2003.

[EIT 95] EITER T., GOTTLOB G., « Identifying the Minimal Transversals of a hypergraph and Related Problems », *SIAM Journal on Computing*, vol. 24, n° 6, 1995, p. 1278-1304.

[EIT 02a] EITER T., GOTTLOB G., « Hypergraph Transversal Computation and Related Problems in Logic and AI », FLESCA S., GRECO S., LEONE N., IANNI G., Eds., *Proceeding of the Logics in Artificial Intelligence, European Conference, JELIA, Cosenza, Italy, September, 23-26*, vol. 2424 de *Lecture Notes in Computer Science*, Springer, 2002.

[EIT 02b] EITER T., GOTTLOB G., MAKINO K., « New Results on Monotone Dualization and Generating Hypergraph Transversals », *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC-02)*, New York, May 2002, ACM Press, p. 14-22.

[FEN 02] FENSEL D., BUSSLER C., MAEDCHE A., « Semantic Web Enabled Web Services », *International Semantic Web Conference*, Sardinia, Italy, June 2002, p. 1-2.

[GON 01] GONZÁLEZ-CASTILLO J., TRASTOUR D., BARTOLINI C., « Description Logics for Matchmaking of Services », *Proc. of the KI-2001 Workshop*, vol. 44, Vienna, Austria, September 2001.

[HAC 02a] HACID M.-S., REY C., TOUMANI F., « Dynamic Discovery of E-services (A Description Logics Based Approach). », Report, 2002, LIMOS, Clemont-Ferrand, France, see http ://lisi.insa-lyon.fr/∼mshacid/eServices.pdf.

[HAC 02b] HACID M., LÉGER A., REY C., TOUMANI. F., « Computing concept covers : a preliminary report. », *Proceedings of the International Workshop on Description Logics (DL02)*, Toulouse, France, April 2002.

[HAC 02c] HACID M., LÉGER A., REY C., TOUMANI F., « Dynamic discovery of e-services in a Knowledge Representation and Reasoning context. », *Proceedings of the 18th French conference on advanced databases (BDA)*, Paris, France, October 2002.

[HOR 02a] HORROCKS I., P.F.PATEL-SCHNEIDER, VAN HARMELEN F. ., « Reviewing the Design of DAML+OIL : An Ontology Language for the Semantic Web », *Proc. of the 18th Nat. Conf. on Artificial Intelligence (AAAI 2002)*, 2002.

[HOR 02b] HORROCKS I., « DAML+OIL : a Reason-able web ontology language », *Proceedings of the Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology*, vol. 2287 de *Lecture Notes in Computer Science*, Prague, Czech Republic, March 2002, Springer.

[KAV 99] KAVVADIAS D. J., STAVROPOULOS E. C., « Evaluation of an Algorithm for the Transversal Hypergraph Problem », *Lecture Notes in Computer Science*, vol. 1668, 1999, p. 72-85.

[LI 03] LI L., HORROCKS I., « A software framework for matchmaking based on semantic web technology », *Proceedings of the Twelfth International World Wide Web Conference (WWW'2003)*, 2003.

[LOP 00] LOPES S., PETIT J.-M., LAKHAL L., « Efficient Discovery of Functional Dependencies and Armstrong Relations », ZANIOLO C., LOCKEMANN P. C., SCHOLL M. H., GRUST T., Eds., *Proceedings of the 7th International Conference on Extending Database Technology (EDBT 2000)*, vol. 1777 de *Lecture Notes in Computer Science*, Konstanz, Germany, 2000, Springer, p. 350-364.

[LUT 02] LUTZ C., SATTLER U., « A Proposal for Describing Services with DLs », *Proceedings of the 2002 International Workshop on Description Logics*, 2002.

[MAN 94] MANNILA H., RäIHä K.-J., *The Design of Relational Databases*, Addison-Wesley, Wokingham, England, 1994.

[MIC 96] MICHAEL L. FREDMAN L. K., « On the Complexity of Dualization of Monotone Disjunctive Normal Forms. », *Journal of Algorithms*, vol. 21, n° 3, 1996, p. 618-628.

[PAO 02] PAOLUCCI M., KAWAMURA T., PAYNE T., SYCARA K., « Semantic Matching of Web Services Capabilities. », *Proc. of the Int. Semantic Web Conference*, Sardinia, Italy, June 2002, p. 333-347.

[REY 03] REY C., TOUMANI F., HACID M.-S., LÉGER A., « An algorithm and a prototype for the dynamic discovery of e-services », Report, 2003, LIMOS, Clermont-Ferrand, France, see http ://www.isima.fr/∼rey/.

[SYC 02] SYCARA K., WIDOFF S., KLUSCH M., , LU J., « LARKS : Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace », *Autonomous Agents and Multi-Agent Systems*, vol. 5, 2002, p. 173-203.

[TEE 94] TEEGE G., « Making the Difference : A Subtraction Operation for Description Logics. », *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, May 1994, p. 540-550.

[TRA 02] TRASTOUR D., BARTOLINI C., PREIST C., « Semantic Web Services : Semantic web support for the business-to-business e-commerce lifecycle », *Proceedings of the eleventh international conference on World Wide Web*, ACM Press, 2002, p. 89-98.

[WEI 01] WEIKUM G., Ed., *Data Engineering Bulletin : Special Issue on Infrastructure for Advanced E-Services*, 2001.

[WYS 01] WYSS C., GIANNELLA C., ROBERTSON E., « FastFDs : A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances », KAMBAYASHI Y., WINIWARTER W., ARIKAWA M., Eds., *Proceedings of the Data Warehousing and Knowledge Discovery, Third International Conference, DaWaK 2001*, vol. 2114 de *Lecture Notes in Computer Science*, Munich, Germany, September 2001, Springer.

Annexe : démonstration détaillée du théorème 3.1

a. : Immédiat.
b. : Nous rappelons que (1) $X_i \cap e = \emptyset$, (2) $X_i \in Tr(\mathcal{H})$, (3) $\mathcal{H}' = \mathcal{H} \cup e$ et (4) $c_j \in e$.
Soit $(*) = X_i \cup \{c_j\}$ est un transversal non minimal de $\mathcal{H}'$.
Soit $(**) = \exists X_k \in Tr(\mathcal{H}) \mid X_k \cap e = \{c_j\}$ et $X_k \setminus \{c_j\} \subset X_i$.

$(*) \overset{(1,2,3)}{\Leftrightarrow}$ $\exists Y \mid$
$Y \in Tr(\mathcal{H}')$ et
$Y \subset X_i \cup \{c_j\}$

$\Leftrightarrow$ $\exists Y \mid$
$\forall e' \in \mathcal{H}',\ Y \cap e' \neq \emptyset$ et
$\forall c_y \in Y,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}')$ et
$c_j \in Y$ et
$Y \setminus \{c_j\} \subset X_i$

$\overset{(2,3)}{\Leftrightarrow}$ $\exists Y \mid$
$\forall e' \in \mathcal{H}',\ Y \cap e' \neq \emptyset$ et
$Y \setminus \{c_j\} \notin tr(\mathcal{H})$ et
$\forall c_y \in Y,\ c_y \neq c_j,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}')$ et
$c_j \in Y$ et
$Y \setminus \{c_j\} \subset X_i$

$\overset{(3,4)}{\Leftrightarrow}$ $\exists Y \mid$
$\forall e' \in \mathcal{H},\ Y \cap e' \neq \emptyset$ et
$Y \setminus \{c_j\} \notin tr(\mathcal{H})$ et
$\forall c_y \in Y,\ c_y \neq c_j,\ Y \setminus \{c_y\} \notin tr(\mathcal{H})$ et
$c_j \in Y$ et
$Y \setminus \{c_j\} \subset X_i$

$\overset{(1,4)}{\Leftrightarrow}$ $\exists Y \mid$
$Y \in Tr(\mathcal{H})$ et
$Y \cap e = c_j$ et
$Y \setminus \{c_j\} \subset X_i$

$\overset{(1,2,3,4)}{\Leftrightarrow}$ $\exists Y \mid$
$Y \in Tr(\mathcal{H}')$ et
$c_j \in Y$ et
$Y \setminus \{c_j\} \subset X_i$

$\Leftrightarrow$ $\exists Y \mid$
$\forall e' \in \mathcal{H}',\ Y \cap e' \neq \emptyset$ et
$Y \setminus \{c_j\} \notin tr(\mathcal{H}')$ et
$\forall c_y \in Y,\ c_y \neq c_j,\ Y \setminus \{c_y\} \notin tr(\mathcal{H}')$ et
$c_j \in Y$ et
$Y \setminus \{c_j\} \subset X_i$

$\overset{(3,4)}{\Leftrightarrow}$ $\exists Y \mid$
$\forall e' \in \mathcal{H}',\ Y \cap e' \neq \emptyset$ et
$Y \setminus \{c_j\} \notin tr(\mathcal{H})$ et
$\forall c_y \in Y,\ c_y \neq c_j,\ Y \setminus \{c_y\} \notin tr(\mathcal{H})$ et
$c_j \in Y$ et
$Y \setminus \{c_j\} \subset X_i$

$\Leftrightarrow$ $\exists Y \mid$
$Y \in Tr(\mathcal{H})$ et
$c_j \in Y$ et
$Y \setminus \{c_j\} \subset X_i$

$\Leftrightarrow$ $(**)$