Traitement coopératif des requêtes RDF dans le contexte des bases de connaissances incertaines

Ibrahim Dellal, Stéphane Jean, Allel Hadjali, Brice Chardin, Mickaël Baron

LIAS / ISAE-ENSMA / Université de Poitiers
1 Avenue Clément Ader, 86960 Futuroscope Cedex, France
prénom.nom@ensma.fr

DOI : 30.3166/DN.21.1-2.9-35

RÉSUMÉ. *De nombreuses larges bases de connaissances (BC) incertaines sont disponibles sur le web dans lesquelles les faits représentés sont associés à un degré de confiance α. Les utilisateurs de ces BC n'ayant, en général, qu'une connaissance partielle de leurs contenus, certaines de leurs requêtes peuvent échouer, c'est-à-dire qu'elles ne retournent aucun résultat. Pour éviter de renvoyer un ensemble vide de réponses, qui est souvent un résultat frustrant pour les utilisateurs, nous proposons, d'une part, d'expliquer l'échec de la requête (pour un ou plusieurs degrés de confiance α) en fournissant à l'utilisateur un ensemble de αMFS (α*Minimal Failing Subqueries*) et, d'autre part, d'indiquer à l'utilisateur des requêtes alternatives, appelées αXSS (α*MaXimal Succeeding Subqueries*), qui réussissent (c'est-à-dire, qui ont des résultats) et sont aussi proches que possible de la requête initiale. Pour calculer ces αMFS et αXSS pour plusieurs degrés α, nous proposons trois approches algorithmiques. Les expérimentations menées sur le benchmark WatDiv montrent l'intérêt de ces approches en comparaison avec une approche naïve.*

ABSTRACT. *Several large uncertain Knowledge Bases (KBs) are available on the web where facts are associated with a certainty degree α. Usually, users only partially understand the content of the KBs, and may submit failing queries, i.e. queries that return no result for the desired certainty. To prevent this frustrating situation, instead of returning an empty set of answers, our approach explains the reasons of the failure (for a single degree α and for several degrees) with a set of (αMFSs (αMinimal Failing Subqueries), and computes alternative relaxed queries, called αXSSs (αMaXimal Succeeding Subqueries), that are as close as possible to the initial failing query. We propose three algorithmic approaches to compute αMFSs and αXSSs. The conducted experiments on the WatDiv benchmark show the relevance of our approaches in comparison with a baseline method.*

MOTS-CLÉS : *BC incertaines, Requêtes SPARQL, Réponse vide.*

KEYWORDS: *Uncertain KB, SPARQL queries, Empty answers.*

1. Introduction

Une *base de connaissances* (BC) est un ensemble d'entités (nommées) et de faits sur ces entités. Les techniques récentes d'extraction d'information ont conduit à la construction de larges BC à partir des données du web comme, par exemple, DBpedia (Lehmann *et al.*, 2015), YAGO (Hoffart *et al.*, 2013), Wikidata (Vrandečić, 2012) et NELL (Carlson *et al.*, 2010) construites dans le cadre de projets académiques. Des BC ont également été conçues dans le cadre de projets commerciaux telles que celles définies par Google (Dong *et al.*, 2014)) et Walmart (Deshpande *et al.*, 2013). Ces BC contiennent des milliards de faits représentés sous forme de triplets RDF *(sujet, prédicat, objet)* et peuvent être interrogées au moyen du langage SPARQL (Prud'hommeaux, Seaborne, 2008). Comme ces BC ont été construites à partir de sources externes et notamment du web, leurs faits peuvent être *incertains* (c'est-à-dire potentiellement faux et incohérents). Pour prendre en compte cette incertitude, des extensions de RDF et de SPARQL ont été proposées afin de supporter des données pondérées par une confiance (Hartig, 2009 ; Tomaszuk *et al.*, 2013). Ainsi un degré de confiance explicite est associé aux faits de la BC et aux résultats des requêtes SPARQL.

Lors de l'interrogation des BC incertaines, les utilisateurs s'attendent à obtenir des résultats de qualité, c'est-à-dire des résultats possédant un degré de confiance supérieur à un seuil donné α. Cependant, comme ces utilisateurs connaissent rarement la structure et le contenu de la BC cible, ils peuvent formuler des requêtes trop restrictives et ainsi être confrontés au problème de réponses insatisfaisantes, c'est-à-dire qu'ils n'obtiennent aucun résultat ou qu'ils n'obtiennent que des résultats avec un degré de confiance inférieur au seuil donné. Une étude réalisée par Saleem et al. sur *les points d'entrée SPARQL* montre que 10 % des requêtes soumises à DBpedia entre mai et juillet 2010 retournaient des résultats vides (Saleem *et al.*, 2015). Au lieu de retourner à l'utilisateur un ensemble vide comme réponse à sa requête, un système coopératif et intelligent peut l'aider à comprendre les raisons de cet échec. Une des approches d'explication de l'échec se base sur l'identification des sous-parties de la requête qui échouent ou qui réussissent (Fokou *et al.*, 2017). Deux catégories de sous requêtes sont ainsi identifiées et représentées par deux ensembles : les sous requêtes minimales échouant (MFS pour Minimal Failing Subqueries) et les sous requêtes maximales réussissant (XSS pour MaXimal Succeding Subqueries). Les MFS représentent les causes d'échec de la requête et peuvent ainsi permettre à l'utilisateur de connaître les parties de sa requête qui font qu'il n'obtient pas de résultat. Les XSS sont des requêtes *relaxées* comportant un nombre maximal d'éléments de la requête initiale et fournissant un résultat non vide de résultats. L'utilisateur peut ainsi choisir d'exécuter l'une de ses requêtes pour obtenir une réponse non vide.

Dans cet article, nous nous intéressons à la généralisation de la notion de MFS et de XSS dans le contexte des BC incertaines. Nous appelons αMFS et αXSS les

causes d'échec et les sous-requêtes relaxées maximales d'une requête recherchant des résultats avec un degré de confiance supérieur ou égal à un seuil α. Nous montrons d'abord à quelles conditions l'approche de calcul des MFS et des XSS proposée par Fokou et al. (Fokou *et al.*, 2017) peut être directement adaptée pour les αMFS et αXSS. Obtenir les αMFS et αXSS, pour le seuil de confiance défini dans sa requête, permet à l'utilisateur de savoir pourquoi sa requête échoue pour ce degré de confiance. Cependant, ceci ne permet pas de savoir ce qui se passerait si l'utilisateur choisissait de diminuer le degré de confiance attendu. Aussi, nous étudions ensuite la possibilité de suggérer à l'utilisateur des requêtes relaxées avec des seuils de confiance inférieurs au seuil α donné. Ce type de relaxation nécessite le calcul des αMFS et αXSS pour différents seuils α. Une solution naïve consisterait à utiliser l'algorithme de calcul des αMFS et αXSS pour chaque seuil α. Cependant, nous avons identifié que les αMFS et αXSS pour un α donné peuvent permettre de déduire des αMFS et αXSS pour un seuil supérieur ou inférieur à α. Ainsi, en fonction de l'ordre dans lequel les valeurs α sont considérées, trois approches sont proposées : *ascendante*, *descendante* et *hybride*. Pour montrer l'intérêt de ces approches en comparaison avec l'approche naïve, nous effectuons une série d'expérimentations sur le Benchmark WatDiv (Aluç *et al.*, 2014) avec le QuadStore Jena TDB.

Cet article est une extension de notre travail publié dans la conférence INFORSID (Dellal *et al.*, 2017). Nous avons développé, révisé et amélioré ce que nous avons présenté précédemment. En particulier, cet article contient les contributions originales suivantes : (1) la proposition d'une troisième approche de calcul des αMFS et αXSS pour différents seuils α, nommée *hybride*, (2) la définition de l'ensemble des preuves sur lesquelles se basent nos approches et (3) l'analyse détaillée des travaux existants sur le problème des réponses vides dans le contexte des requêtes SPARQL et relationnelles.

La suite de cet article est organisée comme suit. La section 2 fournit quelques notions de base et formalise le problème abordé. La section 3 motive notre contribution avec un exemple de requête sur une BC incertaine. La section 4 définit les conditions dans lesquelles nos précédents travaux peuvent être directement adaptés pour trouver les αMFS et αXSS d'une requête RDF incertaine défaillante. La section 5 présente les trois approches proposées (*ascendante*, *descendante* et *hybride*) pour le calcul des αMFS et les αXSS pour différents seuils α. La section 6 décrit la mise en œuvre et l'évaluation expérimentale réalisée. La section 7 propose une revue des travaux connexes. Enfin, nous concluons et fournissons quelques perspectives dans la section 8.

2. Préliminaires et problématique

Dans toute la suite de l'article, nous adoptons les notations de RDF et SPARQL utilisées par Pérez et al. (Pérez *et al.*, 2009), et le modèle de confiance proposé par Hartig (Hartig, 2009). Nous définissons ces notations dans ce qui suit.

2.1. Modèle de données

Un *triplet RDF* est un triplet (sujet, prédicat, objet) $\in (U \cup B) \times U \times (U \cup B \cup L)$ où U est un ensemble d'URI, B est un ensemble de ressources anonymes et L est un ensemble de littéraux. Nous notons T l'union $U \cup B \cup L$. Une *base de données RDF* contient un ensemble de triplets RDF (noté T_{RDF}). Chaque triplet RDF est associé à un *degré de confiance* (ou un *degré de certitude*) représentant la fiabilité du fait représenté. Ce degré est associé au triplet par une fonction $tv : T_{RDF} \rightarrow [0,1]$.

2.2. Requêtes RDF

Un *patron de triplet RDF* t est un triplet (sujet, prédicat, objet) $\in (U \cup V) \times (U \cup V) \times (U \cup V \cup L)$, où V est un ensemble de variables disjointes de U, B et L. Nous notons $var(t) \subseteq V$ l'ensemble des variables de t. Une *requête RDF* est vue comme une conjonction de patrons de triplets : $Q = t_1 \wedge \cdots \wedge t_n$. Cette définition correspond aux requêtes *Basic Graph Pattern* de SPARQL. Le nombre de patrons de triplets d'une requête Q est noté $|Q|$ et les variables de cette dernière sont notées $var(Q) = \bigcup var(t_i)$.

2.3. Évaluation d'une requête RDF

Pour définir comment une requête RDF est évaluée, nous avons besoin de la notion de *mapping* introduite par Pérez et al. (Pérez *et al.*, 2009). Un *mapping* est une fonction partielle $\mu : V \rightarrow T$. Pour un patron de triplets t, nous notons $\mu(t)$ le triplet obtenu en remplaçant les variables de t, $var(t)$, par leurs mappings $\mu(var(t))$. Le domaine de μ, $dom(\mu)$, est un sous-ensemble de V où μ est défini. Deux mappings μ_1 et μ_2 sont *compatibles* lorsque pour tout $x \in dom(\mu_1) \cap dom(\mu_2)$, on a $\mu_1(x) = \mu_2(x)$, c'est-à-dire lorsque $\mu_1 \cup \mu_2$ est aussi un mapping. Soit Ω_1 et Ω_2 des ensembles de mappings, on définit *la jointure* de Ω_1 et Ω_2 par : $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \; sont \; des \; mappings \; compatibles\}$. Soit D une base de données RDF, t un patron de triplet. L'évaluation du patron de triplet t dans D notée $[[t]]_D$ est définie par : $[[t]]_D = \{\mu \mid dom(\mu) = var(t) \wedge \mu(t) \in D\}$. Soit $Q = t_1 \wedge \cdots \wedge t_n$ une requête, l'évaluation de Q sur D est définie par : $[[Q]]_D = [[t_1]]_D \bowtie \cdots \bowtie [[t_n]]_D$. Cette évaluation peut être effectuée sous différents régimes d'implication (*entailment regimes*) tels que définis dans la spécification $SPARQL$ (par exemple, le régime d'implication simple ou RDF). Dans cet article, les exemples et les expérimentations sont basés sur le régime d'implication simple même si n'importe quel autre régime d'implication pourrait être utilisé.

Soit μ une solution de la requête $Q = t_1 \wedge \cdots \wedge t_n$ et $aggreg$ une fonction d'agrégation (par exemple, l'opérateur minimum), le degré de confiance de μ est défini par $tv(\mu, Q) = aggreg(tv(\mu(t_1)), \cdots, tv(\mu(t_n)))$. Une requête RDF peut être associée à un seuil de confiance α afin de ne retourner que les résultats dont le degré de confiance est supérieur ou égal à α. L'évaluation d'une telle requête est définie par : $[[Q]]_D^{\alpha} = \{\mu \in [[Q]]_D \mid tv(\mu) \geq \alpha\}$.

2.4. Notions de αMFS et αXSS

Soit une requête $Q = t_1 \wedge \cdots \wedge t_n$, une requête $Q' = t_i \wedge \cdots \wedge t_j$ est une *sous requête* de Q, i.e. $Q' \subseteq Q$, ssi $\{i, \cdots, j\} \subseteq \{1, \cdots, n\}$. Si $\{i, \cdots, j\} \subset \{1, \cdots, n\}$, Q' est dite une *sous requête propre* de Q ($Q' \subset Q$).

DÉFINITION 1. — *Une requête minimale échouant MFS d'une requête Q est définie comme suit : $[[MFS]]_D = \emptyset \ \wedge \nexists\, Q' \subset MFS$ tel que $[[Q']]_D = \emptyset$. Par extension, une α requête minimale échouant (αMFS) MFS d'une requête Q, pour un α donné, est définie par : $[[MFS]]^{\alpha}_D = \emptyset \ \wedge \nexists\, Q' \subset MFS$ tel que $[[Q']]^{\alpha}_D = \emptyset$. L'ensemble de toutes les αMFS de Q est noté par mfs$^{\alpha}(Q)$.*

DÉFINITION 2. — *Une requêtes maximale réussissant XSS de Q est définie comme suit : $[[XSS]]_D \neq \emptyset \ \wedge \nexists\, Q'$ tel que $XSS \subset Q' \wedge [[Q']]_D \neq \emptyset$. Par extension, une α requêtes maximale réussissant (αXSS) XSS de Q, pour un α donné, est définie par : $[[XSS]]^{\alpha}_D \neq \emptyset \ \wedge \nexists\, Q'$ tel que $XSS \subset Q' \wedge [[Q']]^{\alpha}_D \neq \emptyset$. L'ensemble de toutes les αXSS de Q est noté par xss$^{\alpha}(Q)$.*

2.5. Description du problème

En entrée du problème, nous considérons que nous avons une requête RDF Q qui échoue pour un seuil de confiance α sur une BC incertaine. Nous nous intéressons d'abord au calcul des αMFS et αXSS pour le seuil α. Puis, nous considérons le problème du calcul des mfs$^{\alpha_i}(Q)$ et des xss$^{\alpha_i}(Q)$ pour un ensemble de seuils $\{\alpha_1, \cdots, \alpha_n\}$, où $\alpha \in \{\alpha_1, \cdots, \alpha_n\}$.

3. Un exemple motivant

Considérons la requête suivante qui recherche les livres (Book) édités par Springer, écrits par Smith et dont le nombre de pages est renseigné. Nous désignons cette requête par $Q = t_1 \wedge t_2 \wedge t_3 \wedge t_4$ (ou $t_1t_2t_3t_4$ pour simplifier). Nous considérons également que cette requête interroge la BC incertaine D présentée dans le tableau 1

```
SELECT ?b ?p WHERE {
?b authors "Smith".          (t1)
?b editor "Springer" .       (t2)
?b type Book .               (t3)
?b nbPages ?p }              (t4)
```

Les αMFS et αXSS pour différents seuils de cette requête sont présentées dans la figure 1 sous forme de treillis de sous-requêtes de Q.

Supposons que l'utilisateur souhaite avoir des résultats avec un degré de confiance d'au moins 0,8. Dans notre exemple, cette requête échoue. Cependant, grâce aux 0,8 αMFS (t_1 et t_2), nous pouvons fournir les explications suivantes à l'utilisateur. Pour un degré de confiance de 0,8 :

Tableau 1. Une base de données RDF incertaine D

BC incertaine			
sujet	**prédicat**	**object**	**tv**
b_1	author	Victor	0,1
b_1	editor	Easy	0,3
b_1	type	Book	0,9
b_1	nbPages	90	0,9
b_2	editor	Springer	0,7
b_2	type	Book	0,3
b_2	nbPages	90	0,7
b_3	type	Book	0,7
b_3	nbPages	88	0,7
b_4	authors	90	0,5
b_4	type	Book	0,1
b_5	authors	Smith	0,5
b_5	nbPages	90	0,3
b_6	authors	Smith	0,3
b_6	editor	Springer	0,3
b_6	type	Book	0,3

– il n'existe aucun article écrit par Smith (αMFS t_1) ;

– il n'existe aucun article édité par Springer (αMFS t_2).

La requête Q a une seule 0,8 αXSS (t_3t_4) qui indique une requête alternative proposant des résultats :

– il existe des livres avec leurs nombres de pages (αXSS t_3t_4).

Si nous calculons les αMFS et αXSS pour des degrés de confiance inférieurs (par exemple, 0,2, 0,4 et 0,6), nous pouvons améliorer considérablement l'information retournée à l'utilisateur. Par exemple, les αXSS suivantes peuvent-être utilisées comme requêtes alternatives, à condition que l'utilisateur accepte de diminuer son seuil de confiance. Avec un degré de :

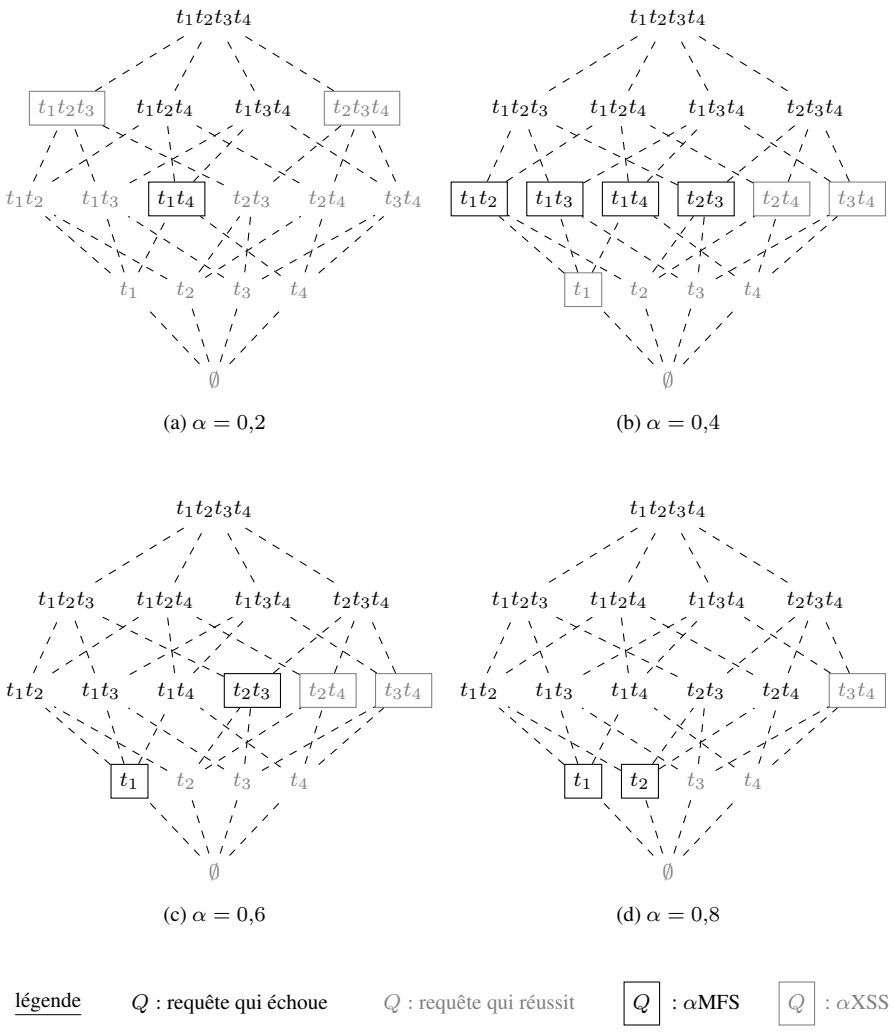
– 0,6, il existe des articles édités par Springer avec leurs nombres de pages ;

– 0,4, il existe des articles écrits par Smith ;

– 0,2, il existe des livres édités par Springer et écrits par Smith ;

– 0,2, il existe des livres édités par Springer avec leurs nombres de pages.

Ce retour peut aider l'utilisateur à reformuler sa requête, à ajuster ses attentes ou à mieux appréhender le contenu de la BC incertaines.



(a) $\alpha = 0{,}2$ (b) $\alpha = 0{,}4$

(c) $\alpha = 0{,}6$ (d) $\alpha = 0{,}8$

Figure 1. Treillis de sous-requêtes de Q pour différents α

4. Calcul des αMFS et αXSS pour un seuil α

Dans cette section, nous proposons dans un premier temps une adaptation de l'algorithme *Lattice-Based Approach* (LBA) proposé par Fokou et al. (Fokou *et al.*, 2017) pour calculer les αMFS et αXSS d'une requête pour un α donné. Notre adaptation possède la même complexité algorithmique que l'algorithme original, à savoir $\mathcal{O}(\sqrt{n} * 2^n)$ dans le pire cas (Fokou *et al.*, 2017).

4.1. L'approche αLBA

Nous présentons ici le fonctionnement de notre algorithmes αLBA comme une adaptation directe de LBA dans le contexte des BC incertaines. Nous verrons par la suite les conditions nécessaires pour que cet algorithme puisse être utilisé.

αLBA explore le treillis des sous-requêtes d'une requête Q en suivant trois étapes : (1) trouver une αMFS de Q, (2) calculer les αXSS potentielles, c'est-à-dire les requêtes maximales qui n'incluent pas la αMFS trouvée précédemment et (3) exécuter les αXSS potentielles ; si elles ont des résultats, ce sont des αXSS sinon, les étapes précédentes sont faites sur les αXSS potentielles qui échouent. Ces trois étapes sont détaillées dans ce qui suit.

4.1.1. Calcul d'une αMFS Q^* de Q

L'algorithme 1 permet de trouver une αMFS de Q. Cet algorithme supprime d'une manière itérative chaque patron de triplet t_i de Q, ce qui conduit à évaluer des sous-requêtes propres Q' de Q. Si Q' échoue pour α, alors Q' contient une αMFS. Inversement, si Q' réussit, alors chaque αMFS de Q contient t_i. La preuve de cette propriété repose sur le fait qu'une requête qui réussit ne peut pas contenir une requête qui échoue (Fokou *et al.*, 2017).

Algorithme 1 : Découverte d'une αMFS pour une requête RDF Q qui échoue

***TrouverUneαMFS*(Q, D, α)**

entrées : Une requête qui échoue $Q = t_1 \wedge ... \wedge t_n$;
une base de données RDF D ;
un seuil α

sorties : Une αMFS de Q notée par Q^*

1 $Q^* \leftarrow \emptyset$;
2 $Q' \leftarrow Q$;
3 **foreach** *patron de triplet* $t_i \in Q$ **do**
4 $\quad Q' \leftarrow Q' - t_i$;
5 $\quad$ **if** $[[Q' \wedge Q^*]]_D^\alpha \neq \emptyset$ **then**
6 $\quad\quad Q^* \leftarrow Q^* \wedge t_i$;
7 **return** Q^*;

Pour illustrer cet algorithme, nous considérons l'exemple de la section 3 avec $\alpha = 0{,}2$ (voir figure 1(a)). En partant de la requête initiale $Q = t_1t_2t_3t_4$, l'algorithme supprime le patron de triplet t_1 et obtient la sous-requête $t_2t_3t_4$. Étant donné que cette requête réussit, t_1 fait partie de la αMFS Q^* recherchée. L'algorithme supprime t_2 et exécute la requête $t_1t_3t_4$. Cette dernière échoue et ainsi, l'algorithme recherche Q^* dans $t_1t_3t_4$. t_3 est supprimé menant à la sous-requête t_1t_4 qui échoue. Ainsi, elle contient Q^*. Enfin, t_4 est supprimé et la sous-requête t_1 réussit. Donc, t_4 est un élément de Q^*. L'algorithme s'arrête et retourne le résultat $Q^* = t_1t_4$.

4.1.2. Calcul des αXSS potentielles

Les αXSS potentielles sont les requêtes maximales qui ne sont pas des super-requêtes de la αMFS trouvée précédemment. L'ensemble des *αXSS potentielles* est noté *pxss*(Q, Q^*). Cet ensemble peut être calculé comme suit :

$$\textit{pxss}(Q,Q^*) = \begin{cases} \emptyset, & \text{si } |Q| = 1. \\ \{Q - t_i \mid t_i \in Q^*\}, & \text{sinon.} \end{cases}$$

Dans notre exemple précédent, *pxss*$(Q, Q^*) = \{t_2t_3t_4, t_1t_2t_3\}$.

Cette deuxième étape est basée sur le fait que toutes les super-requêtes de $Q*$ (c'est-à-dire les requêtes incluant la αMFS Q^* trouvée dans l'étape précédente) renvoient un ensemble vide de réponses et peuvent donc être retirées de l'espace de recherche. Cette propriété est toujours vraie dans le contexte des BC classiques mais, pour les BC incertaines, il est nécessaire qu'une requête qui réussit ne contienne pas une requête qui échoue pour le α donné. Ce pré-requis est discuté dans la section 4.2

4.1.3. Exécution des αXSS potentielles.

Si une sous-requête trouvée lors de l'étape 2 réussit, alors il s'agit d'une αXSS. Sinon, nous appliquons les deux étapes précédentes sur cette sous-requête pour trouver une nouvelle αMFS ainsi que les αXSS potentielles associées. Ceci est illustré par l'algorithme 2. Il est important de noter que cet algorithme évite de redécouvrir une αMFS plusieurs fois (lignes 13-15). Dans notre exemple illustratif, les deux potentielles αXSS $t_2t_3t_4$ et $t_1t_2t_3$ réussissent, ce sont donc les αXSS de Q.

4.2. Propriétés requises de la fonction d'agrégation

Comme nous l'avons vu dans la section précédente, l'algorithme αLBA repose sur le fait qu'une requête qui réussit ne puisse pas contenir une requête qui échoue. Cette propriété est toujours vraie dans le cas des requêtes RDF ne prenant pas en compte les degrés de confiance. Par contre, si la requête est exprimée sur une BC incertaine avec un seuil de confiance minimum spécifiée, cette propriété n'est pas toujours vraie et dépend de la fonction d'agrégation ($aggreg$) choisie pour calculer le degré de confiance d'un résultat. Ceci est illustré dans la figure 2, où nous présentons les résultats d'une requête Q et de sa sous-requête Q' sur une base de données RDF incertaines pour $\alpha = 0{,}4$. Pour les fonctions d'agrégation max et avg, la requête Q réussit alors que sa sous-requête Q' échoue. Ainsi, l'algorithme αLBA ne peut pas être utilisé avec ces fonctions d'agrégations. Comme démontré ci-dessous, cet algorithme peut être utilisé si et seulement si la fonction d'agrégation $aggreg$, utilisée pour attribuer une valeur de confiance aux résultats de la requête, est décroissante par rapport à l'inclusion ensembliste.

Algorithme 2 : Trouver les αMFS et αXSS d'une requête Q

α**LBA(**Q**,** D**,** α**)**
entrées : Une requête qui échoue $Q = t_1 \wedge ... \wedge t_n$;
une base de données RDF D ;
un seuil α
sorties : les αMFS et αXSS de Q
1 $Q^* \leftarrow$TrouverUneαMFS(Q,D,α);
2 *pxss* $\leftarrow$ *pxss*(Q, Q^*);
3 *mfs*$^\alpha(Q) \leftarrow \{Q^*\}$;
4 *xss*$^\alpha(Q) \leftarrow \emptyset$;
5 **while** *pxss* $\neq \emptyset$ **do**
6 $Q' \leftarrow$ *pxss*.*element*(); // choisir un élément de *pxss*
7 **if** $[[Q']]^\alpha_D \neq \emptyset$ **then** // Q' est une αXSS
8 *xss*$^\alpha(Q) \leftarrow$ *xss*$^\alpha(Q) \cup \{Q'\}$;
9 *pxss* $\leftarrow$ *pxss* $- \{Q'\}$;
10 **else** // Q' contient une αMFS
11 $Q^{**} \leftarrow$TrouverUneαMFS(Q',D,α);
12 *mfs*$^\alpha(Q) \leftarrow$ *mfs*$^\alpha(Q) \cup \{Q^{**}\}$;
13 **foreach** $Q'' \in$ *pxss tel que* $Q^{**} \subseteq Q''$ **do**
14 *pxss* $\leftarrow$ *pxss* $- \{Q''\}$;
15 *pxss* $\leftarrow$ *pxss* $\cup \{Q_j \in$ *pxss*$(Q'', Q^{**}) \mid \nexists Q_k \in$ *pxss* $\cup$ *xss*$^\alpha(Q)$ *tel que* $Q_j \subseteq Q_k\}$;
16 **return** $\{$*mfs*$^\alpha(Q)$, *xss*$^\alpha(Q)\}$;

DÉFINITION 3. — *Soit* $aggreg : [0,1]^n \rightarrow [0,1]$ *une fonction d'agrégation,* $aggreg$ *est décroissante par rapport à l'inclusion ensembliste* [1] *si pour tout ensemble A et B* $\in [0,1]^n$, $A \subseteq B \Rightarrow aggreg(A) \geq aggreg(B)$.

Les fonctions $minimum$ et $produit$ sur l'intervalle [0,1] sont deux exemples de fonctions d'agrégation décroissantes.

PROPOSITION 4. — *Soit* $aggreg$ *une fonction décroissante. Si une sous-requête propre* Q' *de* Q *échoue pour un* α *donné (utilisant la fonction* $aggreg$*) alors* Q *échoue également pour* α.

PREUVE 5. — On considère $Q = t_1 \wedge \cdots \wedge t_n$ et sa sous-requête propre $Q' = t_i \wedge \cdots \wedge t_j$ ($\{i, \cdots, j\} \subset \{1, \cdots, n\}$). Supposons que Q' échoue et que Q réussisse : $[[Q']]^\alpha_D = \emptyset$ et $[[Q]]^\alpha_D \neq \emptyset$. Donc, $\exists \mu \in [[Q]]^\alpha_D$. Étant donné que $[[Q]]^\alpha_D \subseteq [[Q]]_D$ et $[[Q]]_D \subset [[Q']]_D$, nous avons $\mu_{|var(Q')} \in [[Q']]_D$ où $\mu_{|var(Q')}$ est la restriction de la fonction μ aux variables de Q'. Par définition, $tv(\mu, Q) = aggreg(tv(\mu(t_1)), \cdots, t$-

1. Pour simplifier, cette définition est limitée aux ensembles mais pourrait être étendue aux multiensembles (multisets)

BC incertaine			
sujet	**prédicat**	**object**	**tv**
b_1	type	Book	0,3
b_1	nbPages	90	0,3
b_2	type	Book	0,3
b_2	nbPages	90	0,9
b_3	type	Book	0,2
b_3	nbPages	88	0,9
b_4	type	Book	0,1
b_4	nbPages	90	0,6
b_5	type	Website	0,8
b_5	nbPages	90	0,9

(a) Une base de données RDF incertaine D

```
Q :  SELECT ?b WHERE {
     ?b type Book . ?b nbPages 90 }
Q' : SELECT ?b WHERE {
     ?b type Book }
```

(b) La requête Q et sa sous-requête Q'

aggreg	$[[Q']]_D^{0,4}$	$[[Q]]_D^{0,4}$
min	$\emptyset$	$\emptyset$
max	$\emptyset$	$\{b_2, b_4\}$
$\prod$	$\emptyset$	$\emptyset$
avg	$\emptyset$	$\{b_2\}$

(c) Résultats de Q et Q'

Figure 2. Illustration des différentes fonctions d'agrégation

$v(\mu(t_n))) \geq \alpha$ et $tv(\mu_{|var(Q')}, Q') = aggreg(tv(\mu(t_i)), \cdots, tv(\mu(t_j)))$ (en effet, $tv(\mu, Q') = tv(\mu_{|var(Q')}, Q')$). Étant donné que $aggreg$ est décroissante, $aggreg(tv(\mu(t_i)), \cdots, tv(\mu(t_j))) \geq aggreg(tv(\mu(t_1)), \cdots, tv(\mu(t_n))) \geq \alpha$. En conséquence, $tv(\mu_{|var(Q')}, Q') \geq \alpha$ et étant donné que $\mu_{|var(Q')} \in [[Q']]_D$ nous déduisons que $\mu_{|var(Q')} \in [[Q']]_D^{\alpha}$. Cela contredit l'hypothèse que Q' échoue. ■

Grâce à l'algorithme αLBA, l'utilisateur peut être informé des causes d'échec de sa requête et obtenir des requêtes relaxées pour le seuil de confiance spécifiée dans sa requête. Cependant, l'utilisateur pourrait également être enclin à diminuer le seuil de confiance spécifié afin de conserver une requête plus proche de sa requête initiale. Aussi, nous nous intéressons maintenant au problème de l'identification des αMFS et αXSS pour un ensemble de seuils α. En plus des causes d'échec basées sur les patrons de triplet, les retours sur des seuils multiples peuvent permettre à l'utilisateur de réévaluer le seuil de confiance associé à sa requête.

5. Calcul des αMFS et αXSS pour différents seuils α

Afin de trouver les αMFS et αXSS pour un ensemble de $\alpha : \{\alpha_1, \cdots \alpha_n\}$, la solution naïve consiste à exécuter l'algorithme αLBA pour chaque α_i. Cette solution de base est appelée $NLBA$. Dans cette section, nous proposons différentes améliorations de cette approche. L'idée est d'utiliser les αMFS et αXSS trouvées pour un seuil donné afin d'en déduire celles d'un seuil supérieur (ou inférieur). Nous commençons par étudier une approche ascendante (d'un seuil inférieur à un seuil plus élevé).

5.1. Approche ascendante

Dans cette section, nous considérons deux seuils α_i et α_j tels que $\alpha_i < \alpha_j$. Si Q^* est une α_iMFS de Q, alors Q^* échoue également pour α_j. Cependant, cette sous-requête n'est pas nécessairement minimale pour α_j et peut donc ne pas être une α_jMFS. La proposition suivante énonce une condition pour laquelle une α_iMFS est aussi une α_jMFS.

PROPOSITION 6. — *Soit α_i et α_j deux seuils tels que $\alpha_i < \alpha_j$ et Q^* une α_iMFS de Q sur un ensemble de données D. Si $|Q^*| = 1$, alors Q^* est aussi une α_jMFS de Q.*

PREUVE 7. — Si Q^* est une α_iMFS de Q sur un ensemble de données D, alors $[[Q^*]]_D^{\alpha_i} = \emptyset$. Puisque $\alpha_i < \alpha_j$, nous avons aussi $[[Q^*]]_D^{\alpha_j} = \emptyset$. Q^* est minimale ($|Q^*| = 1$) et échoue pour α_j, ainsi Q^* est une α_jMFS de Q. ■

Comme indiqué précédemment, toutes les sous-requêtes propres d'une α_jMFS d'une requête Q doivent réussir. D'après la proposition 6, cette propriété est toujours vraie si la requête contient un seul patron de triplet. Vérifier si une requête ne possède qu'un patron de triplet ne nécessite aucun accès à la BC. Ainsi, nous vérifions d'abord ce cas simple et ajoutons le cas échéant les α_jMFS correspondantes à l'ensemble des *αMFS découvertes*, noté *dmfs*$^{\alpha_j}(Q)$. Dans le cas contraire ($|Q^*| \geq 2$), prouver que Q^* est une α_jMFS nécessite de vérifier la réussite de toutes ses sous-requêtes, ce qui exige l'exécution de toutes celles-ci (soit $|Q^*|$ requêtes) sans garantie de trouver une α_jMFS. En revanche, l'algorithme *TrouverUneαMFS* de αLBA (algorithme 1) requiert lui aussi l'exécution de $|Q^*|$ requêtes mais garantit qu'une αMFS soit trouvée. Ainsi, notre approche privilégie l'algorithme *TrouverUneαMFS* par rapport à l'exécution des sous-requêtes de la α_iMFS, comme nous le verrons dans l'algorithme 3.

Considérons maintenant le cas des αXSS. Une α_iXSS de Q ne réussit pas nécessairement pour le seuil α_j. La proposition suivante montre que si cette α_iXSS réussit, elle est aussi une α_jXSS de Q.

PROPOSITION 8. — *Soit α_i et α_j deux seuils tel que $\alpha_i < \alpha_j$ et Q^* une α_iXSS de Q sur un ensemble de données D. Si $[[Q^*]]_D^{\alpha_j} \neq \emptyset$, alors Q^* est une α_jXSS de Q.*

PREUVE 9. — Si Q^* est une α_iXSS de Q sur un ensemble de données D, alors toutes ses super-requêtes échouent pour α_i (autrement, elle ne serait pas maximale). Comme

$\alpha_i < \alpha_j$, ses super-requêtes échouent également pour α_j. Si $[[Q^*]]_D^{\alpha_j} \neq \emptyset$, Q^* réussit et elle est maximale pour α_j. Ainsi, Q^* est une α_jXSS de Q. ■

Ainsi, tester si une α_iXSS est aussi une α_jXSS requiert l'exécution d'une seule requête. En faisant cela pour toutes les α_iXSS, ceci nous permet de trouver un ensemble de α_jXSS découvertes, noté $dxss^{\alpha_j}(Q)$.

L'algorithme 3 présente notre approche complète pour trouver un ensemble de α_jMFS et de α_jXSS en se basant sur les α_iMFS et α_iXSS. Toutes les α_iMFS qui ont un seul patron de triplet (*requêteElementaire*) sont insérées dans $dmfs^{\alpha_j}(Q)$ (ligne 1). Ensuite, l'algorithme itère sur les α_iMFS possédant au moins deux patrons de triplet (l'ensemble FQ). Il cherche une α_jMFS Q^* dans une requête Q' de FQ avec l'algorithme *TrouverUneαMFS* (ligne 6). Ensuite, il supprime de l'ensemble FQ toutes les requêtes qui échouent et contiennent Q^* car elles ne peuvent pas être minimales. Ce processus s'arrête lorsque toutes les requêtes de FQ sont traitées (elles ont été utilisées pour trouver une α_jMFS ou supprimées car elles contiennent une α_jMFS trouvée). Les α_jXSS sont ensuite identifiées en exécutant chaque α_iXSS et en conservant celles qui réussissent pour le seuil α_j (lignes 10-11).

Algorithme 3 : découvrir des α_jMFS et α_jXSS avec l'approche ascendante

DécouvrirαMFSXSS($mfs^{\alpha_i}(Q)$**,** $xss^{\alpha_i}(Q)$ D**,** α_j**)**
- **entrées :** Les α_iMFS $mfs^{\alpha_i}(Q)$ d'une requête Q pour un seuil α_i ;
  Les α_iXSS $xss^{\alpha_i}(Q)$ d'une requête Q pour un seuil α_i ;
  une base de données RDF D ;
  un seuil $\alpha_j > \alpha_i$
- **sorties :** Un ensemble de α_jMFS de Q noté $dmfs^{\alpha_j}(Q)$;
  Un ensemble de α_jXSS de Q noté $dxss^{\alpha_j}(Q)$;

1 *requêteElementaire* $\leftarrow \{Q_a \in mfs^{\alpha_i}(Q) \mid |Q_a| = 1\}$;
2 $dmfs^{\alpha_j}(Q) \leftarrow$ *requêteElementaire*;
3 $FQ \leftarrow mfs^{\alpha_i}(Q) -$ *requêteElementaire*;
4 **while** $FQ \neq \emptyset$ **do**
5 $Q' \leftarrow fQ.dequeue()$;
6 $Q^* \leftarrow TrouverUne\alpha MFS(Q', D, \alpha_j)$;
7 $dmfs^{\alpha_j}(Q) \leftarrow dmfs^{\alpha_j}(Q) \cup \{Q^*\}$;
8 **foreach** $Q'' \in FQ$ *tel que* $Q^{**} \subseteq Q''$ **do**
9 $FQ \leftarrow FQ - \{Q''\}$;
10 **foreach** $Q^* \in xss^{\alpha_i}(Q)$ *tel que* $[[Q^*]]_D^{\alpha_j} \neq \emptyset$ **do**
11 $dxss^{\alpha_j}(Q) \leftarrow dxss^{\alpha_j}(Q) \cup \{Q^*\}$;
12 **return** $\{dmfs^{\alpha_j}(Q), dxss^{\alpha_j}(Q)\}$;

Après avoir découvert les α_jMFS et α_jXSS, une version optimisée de αLBA est exécutée en prenant en entrée les α_jMFS et α_jXSS découvertes (voir l'algorithme 4). Cet algorithme calcule les potentielles α_jXSS (*pxss*) qui ne contiennent

aucune α_jMFS (lignes 4-8), supprime de cet ensemble les α_jXSS (ligne 8), puis, itère sur l'ensemble *pxss* comme pour la version originale de αLBA (voir l'algorithme 2).

Algorithme 4 : Version améliorée de αLBA

Optimized-αLBA(Q**,** D**,** α**,** *dmfs*$^{\alpha}(Q)$**,** *dxss*$^{\alpha}(Q)$**)**

entrées : Une requête qui échoue Q;
une base de données RDF D;
un seuil α
un ensemble de αMFS de Q noté *dmfs*$^{\alpha}(Q)$;
un ensemble de αXSS de Q noté *dxss*$^{\alpha}(Q)$;

sorties : Les αMFS et αXSS de Q

1 *mfs*$^{\alpha}(Q) \leftarrow$ *dmfs*$^{\alpha}(Q)$;
2 *xss*$^{\alpha}(Q) \leftarrow$ *dxss*$^{\alpha}(Q)$;
3 $Q^* \leftarrow$ *dmfs*$^{\alpha}(Q).dequeue()$;
4 *pxss* $\leftarrow$ *pxss*(Q, Q^*);
5 **foreach** $Q^* \in$ *dmfs*$^{\alpha}(Q)$ **do**
6 **foreach** $Q' \in$ *pxss tel que* $Q^* \subseteq Q'$ **do**
7 *pxss* $\leftarrow$ *pxss* $- \{Q'\}$;
8 *pxss* $\leftarrow$ *pxss* $\cup \{Q_i \in$ *pxss*$(Q', Q^*) \mid \nexists Q_j \in$ *pxss* $\cup$ *xss*$^{\alpha}(Q)$: $Q_i \subseteq Q_j\}$;
9 *pxss* $\leftarrow$ *pxss* $-$ *dxss*$^{\alpha}(Q)$;
10 **while** *pxss* $\neq \emptyset$ **do**
  `// idem que les lignes 5-15 de αLBA`
11 **return** $\{$*mfs*$^{\alpha}(Q)$, *xss*$^{\alpha}(Q)\}$;

Pour illustrer l'approche ascendante, nous considérons à nouveau l'exemple de la figure 1. À partir des 0,6 αMFS et αXSS, nous montrons comment l'algorithme ascendant calcule les 0,8 αMFS et αXSS. Comme t_1 est une 0,6 αMFS et ne contient qu'un seul patron de triplet, cette dernière est une 0,8 αMFS (proposition 6). La seconde 0,6 αMFS est t_2t_3. Comme cette requête échoue nécessairement pour 0,8, nous utilisons l'algorithme $TrouverUne\alpha MFS$ pour identifier la 0,8 αMFS t_2. Les 0,6 αXSS sont ensuite exécutées, et seule t_3t_4 réussit pour 0,8. Ainsi, t_3t_4 est une 0,8 αXSS (proposition 8). Les αMFS et αXSS découvertes, données en entrée de l'algorithme 4, sont respectivement : *dmfs*$^{\alpha}(Q) = \{t_1, t_2\}$ et *dxss*$^{\alpha}(Q) = \{t_3t_4\}$. À partir de ces ensembles, l'algorithme 4 détermine qu'il n'existe pas de αXSS potentielles (lignes 1-8) et ainsi, que toutes les 0,8 αMFS et αXSS ont été trouvées.

Gestion du cache. Dans sa version originale, l'algorithme LBA maintient un cache des requêtes exécutées annotées avec leurs résultats : réussite ou échec (Fokou *et al.*, 2017). Avant d'exécuter une sous-requête, cet algorithme vérifie d'abord s'il s'agit d'une sous-requête (respectivement., super-requête) d'une des requêtes réussies (resp., échouées) du cache. Si c'est le cas, la sous-requête réussit (resp., échoue). Notre approche étend cette idée comme suit. Les requêtes qui réussissent (resp., échouent) sont associées à un seuil correspondant au maximum (resp., minimum) pour lequel la

requête réussit (resp., échoue). Avant d'exécuter une sous-requête pour un α donné, nous vérifions d'abord s'il s'agit d'une sous-requête (resp., super-requête) d'une des requêtes contenues dans le cache et si le seuil associé est supérieur (resp. inférieur) ou égal à α. Si c'est le cas, la requête réussit (resp., échoue). Comme notre approche découvre également des αXSS (resp., αMFS) pour un α donnée, nous ajoutons au cache toutes les super-requêtes (resp., sous-requêtes) de cette αXSS (resp., αMFS) car elle échouent (resp., réussissent) nécessairement pour α.

5.2. Approche descendante

L'approche ascendante nous permet de découvrir quelques αMFS et αXSS (et ainsi, améliorer l'exécution de αLBA pour plusieurs seuils α), pour des valeurs de seuils croissantes. Nous proposons ici une approche descendante qui calcule les αMFS et αXSS en utilisant des valeurs de seuil décroissantes. Grâce à la relation de dualité entre les αMFS et αXSS, les propriétés et algorithmes utilisés pour cette approche sont similaires à ceux de l'approche ascendante. Soit α_i et α_j deux seuils tel que $\alpha_i > \alpha_j$,

– les α_iMFS qui échouent pour α_j sont des α_jMFS ;

– les α_iXSS d'une taille $|Q| - 1$ sont des α_jXSS ;

– les α_iXSS d'une taille $< |Q| - 1$ réussissent aussi pour α_j et contiennent donc une α_jXSS. La α_jXSS est trouvée en utilisant l'algorithme 5 appelé $TrouverUne\alpha XSS$ qui est le dual de l'algorithme 1 $TrouverUne\alpha MFS$.

Algorithme 5 : Découverte d'une αXSS pour une requête RDF Q qui réussit

***TrouverUne*α*XSS*(Q_i, Q, D, α)**
entrées : La requête initiale Q_i ;
une sous-requête qui réussit $Q = t_1 \wedge ... \wedge t_n$;
une base de données RDF D ;
un seuil α
sorties : Une αXSS de Q notée par Q'
1 $Q' \leftarrow Q$;
2 **foreach** *patron de triplet* $t_i \in (Q_i - Q)$ **do**
3 $Q' \leftarrow Q' \wedge t_i$;
4 **if** $[[Q']]^{\alpha}_{D} = \emptyset$ **then**
5 $Q' \leftarrow Q' - t_i$;
6 **return** Q';

Une fois un ensemble de α_jMFS et de α_jXSS découvert, l'algorithme Optimized-αLBA (algorithme 4) est exécuté pour trouver les α_jMFS et α_jXSS restantes.

Pour illustrer l'approche descendante, nous considérons encore une fois l'exemple de la figure 1. À partir des 0,8 αMFS et αXSS, nous montrons comment l'algorithme ascendant calcule les 0,6 αMFS et αXSS. La 0,8 αMFS t_1 échoue pour 0,6, c'est

donc une 0,6 αMFS. La 0,8 αXSS t_3t_4 réussit nécessairement pour 0,6, cette dernière est utilisée comme paramètre de l'algorithme *TrouverUneαXSS* afin de trouver la 0,6 αXSS t_3t_4 . Ainsi, les 0,6 αMFS et αXSS découvertes sont respectivement $dmfs^{\alpha}(Q) = \{t_1\}$ et $dxss^{\alpha}(Q) = \{t_3t_4\}$. Celles-ci sont utilisées comme paramètre de la version optimisée de αLBA (Algorithme 4) qui identifie alors les autres 0,6 αMFS (t_2t_3) et αXSS (t_2t_4).

5.3. Approche hybride

L'idée de l'approche hybride est de combiner les propriétés utilisées par les approches ascendante et descendante pour découvrir le plus grand nombre possible de αMFS et αXSS. Pour une série ordonnée de seuils $\{\alpha_1, \cdots \alpha_n\}$, l'algorithme hybride considère d'abord le seuil le plus bas α_1, suivi du seuil le plus grand α_n. Ensuite, il itère sur la suite $\{\alpha_1, \cdots \alpha_n\}$ en considérant le seuil central α_i, où $i = \lfloor \frac{n+1}{2} \rfloor$. L'algorithme considère alors récursivement les seuils dans (1) le sous-ensemble des seuils inférieurs à α_i : $\{\alpha_1, \cdots \alpha_i\}$ avec son seuil en position centrale, et (2) le sous-ensemble de seuils supérieurs à α_i : $\{\alpha_i, \cdots \alpha_n\}$ avec son seuil en position centrale, jusqu'à ce que les αMFS et αXSS soient calculées pour chaque seuil. Dans notre exemple avec les seuils $\{0{,}2, 0{,}4, 0{,}6, 0{,}8\}$, l'approche hybride considère ces derniers dans l'ordre suivant : $\{0{,}2, 0{,}8, 0{,}4, 0{,}6\}$. Grâce à cet ordre, lors de la recherche des αMFS et αXSS pour les seuils 0,4 et 0,6, l'approche hybride a accès aux αMFS et αXSS de deux seuils inférieur et supérieur et peut aussi bénéficier des propriétés utilisées dans les approches ascendante et descendante pour découvrir des αMFS et αXSS. Nous utilisons la propriété additionnelle suivante pour trouver des αMFS et αXSS supplémentaires.

PROPOSITION 10. — *Soit α_i, α_j et α_k trois seuils tel que $\alpha_i < \alpha_j < \alpha_k$. Si une requête Q^* est à la fois α_iMFS et α_kMFS, alors Q^* est une α_jMFS. De même, si une requête Q^* est à la fois α_iXSS et α_kXSS, alors Q^* est une α_jXSS.*

La preuve de cette propriété est immédiate de part les caractéristiques des αMFS et αXSS maintenues pour les différents seuils. Nous illustrons le fonctionnement de l'algorithme hybride sur l'exemple courant (figure 1), en calculant les 0,6 αMFS et αXSS, connaissant celles pour 0,4 et 0,8. l'approche hybride identifie t_3t_4 comme une αXSS pour 0,4 et 0,8, c'est donc également une 0,6 αXSS (proposition 10). Ensuite, l'algorithme recherche des 0,4 αMFS contenant un seul patron de triplet et des 0,8 αXSS contenant trois patrons de triplet ($|Q|-1$). Comme cet exemple n'en possède pas, l'approche hybride continue en recherchant les 0,8 αMFS qui échouent pour 0,6 (propriété de l'approche descendante). C'est le cas pour t_1, qui est une 0,6 αMFS. De même, il recherche les 0,4 αXSS qui réussissent pour 0,6 (propriété de l'approche ascendante) et trouve la 0,6 αXSS t_2t_4. L'algorithme *TrouverUneαMFS* est ensuite utilisé avec les 0,4 αMFS. Dans cet exemple, l'algorithme est seulement appliqué à t_2t_3 car les autres contiennent t_1 (la 0,6 αMFS). Il trouve que t_2t_3 est une 0,6 αMFS. Inversement, il utilise l'algorithme *TrouverUneαXSS* avec la 0,8 αXSS. Dans cet exemple, toutes les 0,8 αXSS ont déjà été utilisées. Grâce aux propriétés des approches ascendante et des-

cendante, l'approche hybride a découvert toutes les 0,6 αMFS et αXSS sans avoir eu besoin d'exécuter l'algorithme αLBA.

6. Évaluation expérimentale

Afin de montrer l'intérêt des approches présentées précédemment, nous présentons, dans cette section, des expérimentations comparant les performances de nos trois approches avec celle de l'approche de base $NLBA$ (exécution de αLBA pour chacun des N seuils).

6.1. Environnement expérimental

Nous avons développé nos trois algorithmes descendant, ascendant et hybride, avec JAVA 1.8 64 bits. Ces algorithmes prennent en entrées une requête qui échoue et un ensemble de seuils, et renvoient les ensembles de αMFS et αXSS de Q pour chaque seuil. Dans notre implémentation actuelle, ces algorithmes peuvent être exécutés avec Jena TDB. Nous avons choisi Jena TDB car ce Quadstore nous permet de stocker le degré de confiance associé à chaque triplet. Jena TDB fournit un filtre de bas niveau que nous utilisons pour récupérer les résultats satisfaisant le seuil fourni. Notre implémentation est disponible sur https://forge.lias-lab.fr/projects/qars4ukb avec un tutoriel pour reproduire nos expérimentations.

Nos expérimentations ont été menées sur un système Ubuntu Server 16.04 LTS avec un processeur Intel XEON E5-2630 v3 @2.4Ghz et 16GB de RAM. Arbitrairement, nous utilisons la fonction d'agrégation min. Les temps d'exécution sont une moyenne de cinq exécutions consécutives. Pour éviter un effet de démarrage à froid, une exécution préliminaire est effectuée mais non considérée.

6.2. Données et requêtes utilisées

Nous avons utilisé quatre jeux de données ayant respectivement 20, 40, 60 et 80 millions de triplets (20M, 40M, 60M et 80M) générés par le Benchmark WatDiv (Aluç *et al.*, 2014). Le degré de confiance de chaque triplet RDF a été généré aléatoirement avec une distribution uniforme sur [0 , 1]. Nous considérons 7 requêtes défaillantes présentées dans le tableau 2. Ces requêtes contiennent entre 1 et 15 patrons de triplet et couvrent les principaux types de requêtes.

– *Requêtes en étoile* : elles sont caractérisées par une jointure sujet-sujet, la jointure est réalisée par une même variable présente comme sujet de chaque patron de triplet de la requête.

– *Requêtes en chaîne* : elles sont caractérisées par une jointure objet-sujet. La variable de jointure est présente comme objet dans un patron de triplet de la requête et comme sujet dans l'autre.

– *Requêtes composites* : elles sont caractérisées par un mélange de jointures, sujet-sujet, objet- objet, sujet-objet ou objet-sujet. Au moins deux types différents de jointures sont présents dans la requête.

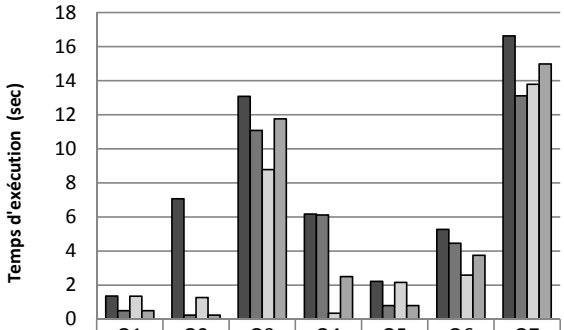
Tableau 2. Requêtes utilisées pour l'évaluation expérimentale (en format simplifié)

Q1 (3TP*)	`SELECT * WHERE { ?p friendOf ?f . ?f likes ?p . ?p type ProductCategory }`
Q2 (6TP)	`SELECT * WHERE { User666524 likes ?v0 . ?v0 hasGenre ?v1 . ?v1 tag Topic129 . ?v0 friendOf ?v2 . ?v2 Location ?v3 . ?v3 parentCountry Country17 }`
Q3 (7TP)	`SELECT * WHERE { ?v0 follows ?v1 . ?v1 follows ?v0 . ?v1 subscribes ?v2 . ?v0 subscribes ?v2 . ?v1 likes Product16770 . ?v0 nationality Country20 . ?v0 makesPurchase ?v3 }`
Q4 (8TP)	`SELECT * WHERE { ?v0 type User . ?v0 familyName 'Smith' . ?v0 subscribes Website362909 . ?v0 follows ?v1 . ?v0 friendOf ?v2 . ?v0 likes ?v3 . ?v0 userId ?v4 . ?v0 makesPurchase ?v5 . ?v0 Location ?v6 . ?v0 nationality ?v7 . ?v0 userId ?v8 }`
Q5 (10TP)	`SELECT * WHERE { ?p likes ?x . ?x likes ?p . ?p hasGenre SubGenre92 . ?x subscribe ?w1 . ?w1 language Language21 . Website121 hits ?h . ?x homepage Website120 . ?x familyName 'Smith' . ?x friendOf ?x2 . ?x2 email 'xxx@xxx.com' }`
Q6 (12TP)	`SELECT * WHERE { ?v0 eligibleRegion Country05 . ?v0 includes ?v1 . Retailer1257 offers ?v0 . ?v0 price '90' . ?v0 serialNumber ?v4 . ?v0 validFrom ?v5 . ?v0 validThrough ?v6 . ?v0 eligibleQuantity ?v8 . ?v0 priceValidUntil ?v11 . ?v1 tag ?v7 . ?v1 keywords ?v10 . ?v12 purchaseFor ?v1 }`
Q7 (15TP)	`SELECT * WHERE { ?v0 type ProductCategory7 . ?v0 tag Topic245 . ?v0 hasReview ?v4 . ?v0 contentSize ?v9 . ?v0 description ?v10 . ?v0 keywords ?v11 . ?v12 purchaseFor ?v0 . ?v2 tag ?v1 . ?v4 rating ?v5 . ?v4 reviewer ?v6 . ?v4 text ?v7 . ?v4 title ?v8 . ?v6 familyName ?v13 . ?v6 birthDate ?v14 . ?v0 gender ?v15 }`

* nombre de patrons de triplet

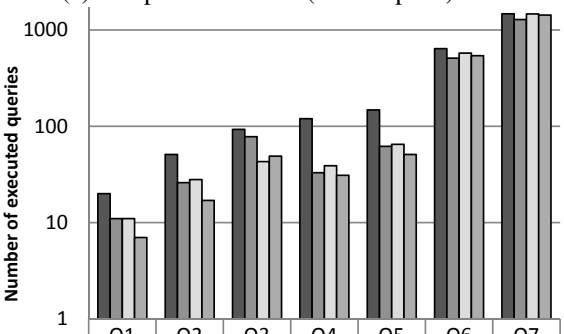
6.3. Expérimentation 1 : test des temps d'exécution des algorithmes

Cette expérimentation évalue le temps d'exécution des algorithmes descendant, ascendant, hybride et NLBA pour des seuils arbitrairement fixés à $\{0{,}2, 0{,}4, 0{,}6, 0{,}8\}$ et un jeu de données de 20M triplets. La figure 3(a) montre le temps d'exécution de chaque algorithme pour chaque requête. La figure 3(b) présente le nombre de requêtes exécutées par chaque algorithme pour chaque requête.



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
NLBA	1,36	7,07	13,08	6,17	2,22	5,27	16,63
Ascendant	0,5	0,24	11,08	6,12	0,8	4,46	13,11
Descendant	1,35	1,27	8,78	0,35	2,16	2,59	13,79
Hybride	0,5	0,24	11,76	2,5	0,8	3,75	14,98

(a) Temps d'exécution (20M triplets)



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
NLBA	20	51	93	120	148	641	1 475
Bottom-Up	11	26	78	33	62	510	1 286
Top-Down	11	28	43	39	65	576	1 470
Hybrid	7	17	49	31	51	541	1 431

(b) # Requêtes exécutées

Figure 3. Résultats expérimentaux avec Jena TDB

Cette expérience montre les gains apportés par nos algorithmes par rapport à la méthode de base NLBA, nos algorithmes exécutent moins de requêtes pour trouver

les αMFS et αXSS. Dans l'ensemble, nos trois algorithmes exécutent respectivement 39 %, 40 % et 39 % moins de requêtes. Par conséquent, ces algorithmes ont des temps d'exécutions inférieur, soit une baisse de temps d'exécution moyen de 30 %, 42 % et 33 %. En fonction des requêtes, ce gain peut être important. Par exemple, NLBA a besoin de 7 secondes pour trouver les αMFS et αXSS de la requête Q2, alors que nos algorithmes effectuent ce calcul en environ 1 seconde. La différence entre les temps d'exécutions dépend fortement des requêtes que nos algorithmes dispensent d'exécuter. Par exemple, nos algorithmes exécutent moins de 40 requêtes pour Q4 tandis que NLBA en exécute 120. Pour l'algorithme descendant et hybride, ce résultat est un gain de performance important ; ce n'est pas le cas pour l'algorithme ascendant qui a presque le même temps d'exécution que NLBA. En analysant les requêtes exécutées, nous constatons que l'algorithme ascendant évite d'exécuter des requêtes qui ont des temps d'exécution courts, et qu'il exécute tout de même les requêtes les plus coûteuses. Ainsi, le temps d'exécution global reste quasiment inchangé. Cette expérience montre également qu'aucun de nos algorithmes ne fournit le meilleur résultat pour toutes les requêtes. Les algorithmes ascendant et hybride ont les meilleurs temps d'exécution pour Q1, Q2 et Q5 tandis que l'algorithme descendant est meilleur pour Q3, Q4 et Q6. Malgré l'exécution de moins de requêtes, l'algorithme ascendant a le pire temps d'exécution. Inversement, l'algorithme descendant exécute le plus grand nombre de requêtes mais a le meilleur temps d'exécution. Ceci est dû au fait que ces algorithmes exécutent différentes requêtes avec des temps d'exécution différents. En particulier, l'algorithme descendant commence par rechercher les αMFS et αXSS pour les seuils les plus élevés. Les requêtes exécutées tendent à être sélectives étant donné que le seuil de confiance est restrictif, et par conséquent, elles ont des temps d'exécution courts. Une fois les αMFS et αXSS trouvées pour les seuils les plus élevés, l'algorithme descendant évite l'exécution des requêtes avec un seuil inférieur qui sont susceptibles d'être plus coûteuses qu'avec un seuil plus élevé. Comme l'algorithme ascendant suit l'approche duale, il tend à exécuter des requêtes non sélectives avec un coût plus important. Dans cette expérimentation, l'algorithme hybride n'est jamais la meilleure approche. Puisque les degrés de confiance ont été générés de façon aléatoire et que les quatre seuils considérés sont séparés par une marge significative, les requêtes partagent peu de αMFS et αXSS entre les seuils. Ainsi, la propriété spécifique à l'algorithme hybride (proposition 10) est rarement exploitée. Comme perspective, nous prévoyons de faire des expériences sur un jeu de données réelles, où les degrés de confiance peuvent avoir d'autres distributions, afin de montrer l'impact de l'algorithme hybride.

6.4. Expérimentation 2 : augmentation de la taille du jeu de données

La deuxième expérimentation consiste à évaluer le passage à l'échelle des algorithmes avec l'augmentation de la quantité de données. Le tableau 3 et la figure 4 présentent les temps d'exécution des algorithmes pour la requête Q2 avec les jeux de données 20M, 40M, 60M et 80M (20M est un sous-ensemble de 40M, etc.).

Tableau 3. Temps d'exécution par rapport à la quantité de données

	NLBA	Ascendant	Descendant	Hybride
20M	7,04	0,24	1,26	0,24
40M	6,86	1,59	4,62	1,57
60M	8,2	1,66	4,94	1,64
80M	9,58	1,72	5,29	1,71

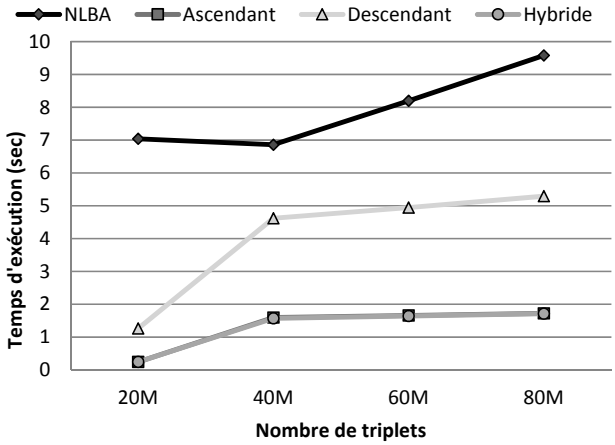


Figure 4. Passage à l'échelle : temps d'exécution par rapport à la quantité de données

Les temps d'exécution de nos algorithmes n'augmentent pas significativement entre les ensembles de données 40M et 80M. Sur ces deux jeux de données, nous avons observé que les αMFS et αXSS de Q2 restent les mêmes et donc que les mêmes requêtes sont exécutées (environ 25 requêtes pour nos algorithmes et 46 pour NLBA). Par conséquent, l'évolutivité des algorithmes dans ce cas dépend uniquement du temps d'exécution des requêtes sur un jeu de données plus important.

Pour l'ensemble de données 20M, Q2 dispose d'une αMFS de plus, et pour tous les seuils. Le nombre de αXSS reste le même, mais celles-ci sont plus petites en terme de nombre de patrons de triplets. En conséquence, les algorithmes exécutent des requêtes différentes. Cela a un impact direct sur leurs performances. En particulier, le temps d'exécution de l'algorithme descendant est d'environ 1 seconde sur le jeu de données 20M (5 secondes sur les autres jeux de données) malgré le fait qu'il exécute plus de requêtes (28 requêtes sur 20M et 24 requêtes sur les autres jeux de données). Ainsi, lorsque les αMFS et αXSS changent, le passage à l'échelle des algorithmes dépend des requêtes exécutées et de leurs temps de réponse respectifs.

6.5. Expérimentation 3 : augmentation du nombre de seuils

Dans cette expérimentation, nous évaluons le passage à l'échelle des algorithmes lorsque le nombre de seuils augmente. Cette expérience a été réalisée avec la requête Q6 sur le jeu de données 20M. Nous avons exécuté les algorithmes pour un seuil $\{0{,}1\}$, deux seuils $\{0{,}1, 0{,}2\}$, trois seuils $\{0{,}1, 0{,}2, 0{,}3\}$, et ainsi de suite. Le résultat de cette expérimentation est présenté dans la figure 5.

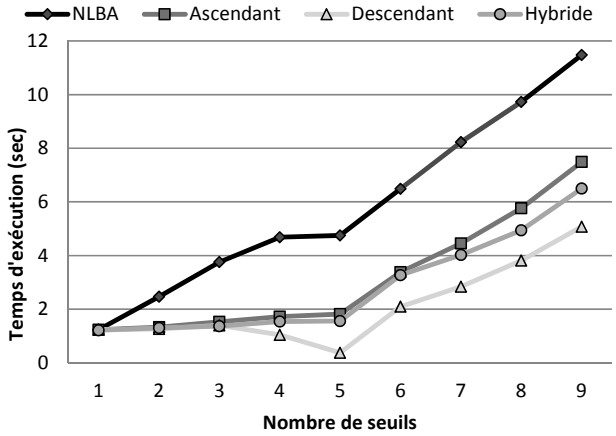


Figure 5. Passage à l'échelle : temps d'exécution par rapport au nombre de seuils

Nos algorithmes possèdent un meilleur temps d'exécution que NLBA à partir de deux seuils. Ceci est dû au fait que pour chaque nouveau seuil, NLBA exécute la version originale de l'algorithme αLBA. En conséquence, le temps de réponse de NLBA est à peu près linéaire en fonction du nombre de seuils. Par comparaison, le passage à l'échelle de nos algorithmes dépend des αMFS et αXSS découvertes. Si les αMFS et αXSS reste majoritairement identiques entre les différents seuils, nos algorithmes ne nécessitent que quelques millisecondes supplémentaires pour trouver les αMFS et αXSS pour un nouveau seuil de l'ensemble des α. C'est le cas dans nos expériences pour les seuils compris entre $0{,}1$ et $0{,}5$. En revanche, si les αMFS et αXSS changent significativement (c'est le cas entre 0,5 et 1), nos algorithmes s'approchent de NLBA car la version optimisée de αLBA exploite peu de αMFS et αXSS découvertes. Nous pouvons également observer dans cette expérimentation que l'algorithme descendant a un meilleur temps d'exécution avec 5 seuils qu'avec un seul seuil. En effet, αLBA a un temps d'exécution court pour 0,5 (comme on peut le voir sur la courbe NLBA). Une fois les αMFS et αXSS trouvées pour 0,5, l'algorithme descendant ne nécessite que quelques millisecondes pour les seuils inférieurs (entre 0,1 et 0,5). L'exécution de αLBA pour 0,1 nécessite un temps d'exécution plus long (les requêtes exécutées étant moins sélectives qu'avec 0,5).

7. Travaux connexes

Dans le contexte des BC, le problème des réponses vides a été abordé par plusieurs approches complémentaires telles que la complétion de la BC en utilisant des règles de déduction logiques (Galárraga *et al*., 2015), la vérification des données lors de la formulation de la requête (Campinas, 2014), l'émergence d'un schéma relationnel à partir des données de la BC pour aider les utilisateurs à formuler des requêtes (Pham *et al*., 2015) ou à relaxer la requête pour retourner des réponses alternatives (Hurtado *et al*., 2009 ; Huang *et al*., 2012 ; Fokou *et al*., 2014 ; Calí *et al*., 2014 ; Hogan *et al*., 2012 ; Elbassuoni *et al*., 2011 ; Dolog *et al*., 2009). Dans cette section, nous résumons les principales contributions sur la relaxation des requêtes RDF et évoquons les liens qu'il y a avec les travaux faits dans le contexte des bases de données relationnelles.

Il existe plusieurs travaux qui ont proposé des opérateurs de relaxation dans le contexte RDF. Ces opérateurs sont principalement basés sur la sémantique RDFS (par exemple, la généralisation des patrons de triplet utilisant des hiérarchies de classes et de propriétés) (Hurtado *et al*., 2009 ; Huang *et al*., 2012 ; Fokou *et al*., 2014 ; Calí *et al*., 2014), les mesures de similarité (Hogan *et al*., 2012 ; Elbassuoni *et al*., 2011) et les préférences utilisateur (Dolog *et al*., 2009). Ces opérateurs génèrent un ensemble de requêtes relaxées, ordonnées par similarité avec la requête d'origine et exécutées dans cet ordre (Hurtado *et al*., 2009 ; Huang *et al*., 2012 ; Reddy, Kumar, 2013). Les opérateurs de relaxation peuvent être directement utilisés par l'utilisateur dans sa requête (Fokou *et al*., 2014 ; Calí *et al*., 2014) ou utilisés conjointement avec des règles de réécriture de requêtes pour effectuer la relaxation (Dolog *et al*., 2009). Avec ces approches, les causes d'échec de la requête sont inconnues, ce qui peut conduire à exécuter des requêtes relaxées inutiles. Fokou et al. (Fokou *et al*., 2016 ; 2017) ont abordé ce problème en définissant d'abord les approches LBA et MBA pour calculer les MFS et XSS de la requête (Fokou *et al*., 2017) et en proposant des stratégies de relaxation basées sur les MFS qui identifient les requêtes relaxées échouant nécessairement (Fokou *et al*., 2016). Notre approche est basée sur l'algorithme LBA proposé dans ce travail, que nous avons étendu en identifiant les conditions nécessaires pour que cet algorithme puisse être utilisé dans le contexte des BC incertaines. Notre travail est parmi les travaux pionniers visant à explorer le problème de la relaxation de requêtes dans le contexte des BC incertaines. Pour autant que nous sachions, le seul autre travail déjà réalisé dans ce contexte est (Reddy, Kumar, 2013). Toutefois, ce travail utilise uniquement la valeur de confiance pour ordonner les résultats par leur fiabilité. Ils ne considèrent pas, comme nous le faisons dans cet article, les requêtes qui ne renvoient aucun résultat satisfaisant le degré de confiance fourni.

La question du calcul des MFS et des XSS a également été abordée dans le contexte des bases de données relationnelles (Godfrey, 1997), des systèmes de recommandation (Jannach, 2009) et des requêtes floues (Pivert, Smits, 2015). Le travail le plus proche du nôtre est celui de Pivert et Smits (Pivert, Smits, 2015) fait dans le contexte de l'interrogation floue, où les auteurs ont proposé une approche pour calculer les MFS progressives, c'est-à-dire les requêtes qui ne sont que faiblement satisfaites car elles ne renvoient aucune réponse avec un degré de satisfaction au moins égal à un seuil

défini par l'utilisateur. Cette approche est basée sur un résumé de la partie pertinente de la base de données. Ils calculent les MFS et XSS progressives pour différents seuils comme dans notre approche. Cependant, ce travail propose une approche pour calculer les MFS et XSS dans le contexte de l'interrogation floue sur des bases de données certaines, alors que nous visons les requêtes classiques sur les BC incertaines. Dans ce nouveau contexte, un résumé de la partie pertinente des BC ne peut pas être efficacement calculé (Fokou *et al.*, 2017).

Enfin, nous notons que le problème original de découverte des MFS et des XSS d'une requête SPARQL est analogue à la découverte d'ensembles fréquents maximaux (Mannila, Toivonen, 1997 ; Gunopulos *et al.*, 2003). En effet, les deux problèmes reviennent à chercher des bordures positives et négatives d'une propriété monotone dans l'espace des solutions représenté par un treillis. Cependant, nous nous intéressons principalement dans cet article à un nouveau problème de découverte des MFS et XSS pour plusieurs seuils. Ce problème n'est pas équivalent à la découverte d'ensembles fréquents maximaux car plusieurs treillis doivent être considérés. D'un point de vue théorique, celui-ci s'apparente à la recherche d'ensembles fréquents maximaux pour plusieurs seuils de fréquence, qui est à notre connaissance peu étudié dans la littérature.

8. Conclusion

Dans cet article, nous nous somme intéressés au problème des réponses vides dans le contexte des BC incertaines où une requête échoue si elle ne renvoie aucun résultat ou si elle renvoie un résultat qui ne satisfait pas le degré de confiance α attendu. Pour fournir à l'utilisateur un retour pertinent, nous avons proposé de calculer les αMFS et αXSS de la requête, car elles donnent un aperçu clair des causes d'échec et un ensemble de requêtes alternatives retournant des résultats utiles.

Nous avons d'abord défini les conditions nécessaires pour que l'algorithme de nos travaux précédents, appelé αLBA, puisse être directement adapté au contexte des BC incertaines. Dans ce cas, l'utilisateur doit définir un degré de confiance attendu. Cependant, l'utilisateur peut vouloir savoir ce qui se passe s'il assouplit cette condition sur la confiance. Ainsi, nous avons étudié le problème du calcul des αMFS et αXSS pour plusieurs seuils. La méthode de base, appelée NLBA, consiste à exécuter αLBA pour chaque seuil. Cependant, nous avons observé et prouvé que les αMFS et αXSS pour un seuil donné peuvent être réutilisées pour trouver celles d'un seuil inférieur (ou supérieur). Ainsi, nous avons défini trois approches alternatives de NLBA, appelées ascendante, descendante et hybride, qui considèrent des seuils α dans des ordre différents. Nous avons effectué une mise en œuvre complète de ces algorithmes et montré expérimentalement sur différents jeux de données du benchmark WatDiv que nos approches sont plus performantes que la méthode de base.

À court terme, nous envisageons d'illustrer l'intérêt de nos approches en les appliquant sur un contexte réel comme, par exemple, avec des requêtes exécutées sur la base de connaissances YAGO. En particulier, nous espérons montrer que l'approche

hybride peut permettre d'obtenir de meilleures performances que les autres approches lorsque les degrés de confiance ne sont pas générés aléatoirement et/ou lorsque de nombreux seuils de confiance sont considérés. Dans nos expérimentations, nous avons aussi observé qu'aucun de nos algorithmes n'offre les meilleures performances pour toutes les requêtes. Comme autre perspective, nous envisageons d'étudier les conditions qui font qu'un algorithme fournit les meilleurs résultats. Notre idée est d'utiliser les statistiques des BC et le modèle de coût du système de gestion des triplets pour trouver l'algorithme susceptible de proposer les meilleures performances. Enfin, à plus long terme, nous pensons qu'il serait intéressant d'adapter nos approches au contexte des données massives.

Bibliographie

Aluç G., Hartig O., Özsu M. T., Daudjee K. (2014). Diversified Stress Testing of RDF Data Management Systems. In *International semantic web conference (iswc 2014)*, p. 197–212. Springer.

Calí A., Frosini R., Poulovassilis A., Wood P. (2014). Flexible Querying for SPARQL. In *Ontologies, databases, and applications of semantics for large scale information systems (odbase 2014)*, p. 473-490.

Campinas S. (2014). Live SPARQL Auto-Completion. In *International semantic web conference (iswc 2014) (posters & demos)*, p. 477–480.

Carlson A., Betteridge J., Kisiel B., Settles B., Hruschka Jr E. R., Mitchell T. M. (2010). Toward an architecture for never-ending language learning. In *Aaai*, vol. 5, p. 3.

Dellal I., Jean S., Hadjali A., Chardin B., Baron M. (2017). Traitement coopératif des requêtes rdf dans le contexte des bases de connaissances incertaines. In *Informatique des organisations et systèmes d'information et de décision.*

Deshpande O., Lamba D. S., Tourn M., Das S., Subramaniam S., Rajaraman A. *et al.* (2013). Building, maintaining, and using knowledge bases: a report from the trenches. In *Proceedings of the 2013 acm sigmod international conference on management of data*, p. 1209–1220.

Dolog P., Stuckenschmidt H., Wache H., Diederich J. (2009). Relaxing RDF queries based on user and domain preferences. In *Journal of intelligent information systems (jiis)*, vol. 33, p. 239–260.

Dong X., Gabrilovich E., Heitz G., Horn W., Lao N., Murphy K. *et al.* (2014). Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In *Proceedings of the 20th acm sigkdd international conference on knowledge discovery and data mining (kdd'14)*, p. 601–610.

Elbassuoni S., Ramanath M., Weikum G. (2011). Query Relaxation for Entity-Relationship Search. In *Extended Semantic Web Conference (ESWC 2011)*, p. 62–76.

Fokou G., Jean S., Hadjali A. (2014). Endowing Semantic Query Languages with Advanced Relaxation Capabilities. In *International Symposium on Methodologies for Intelligent Systems (ISMIS 2014)*, p. 512–517.

Fokou G., Jean S., HadjAli A., Baron M. (2016). RDF Query Relaxation Strategies Based on Failure Causes. In *Extended semantic web conference (eswc 2016)*, p. 439–454.

Fokou G., Jean S., Hadjali A., Baron M. (2017). Handling Failing RDF Queries: From Diagnosis to Relaxation. In *Knowledge and information systems (kais)*, vol. 50, p. 167–195.

Galárraga L., Teflioudi C., Hose K., Suchanek F. M. (2015). Fast Rule Mining in Ontological Knowledge Bases with AMIE+. In *very large data bases (VLDB 2015) journal*, vol. 24, p. 707–730.

Godfrey P. (1997). Minimization in Cooperative Response to Failing Database Queries. In *International journal of cooperative information systems*, vol. 6, p. 95–149.

Gunopulos D., Khardon R., Mannila H., Saluja S., Toivonen H., Sharm R. S. (2003). Discovering All Most Specific Sentences. *ACM Trans. on Database Systems*, vol. 28, n° 2, p. 140–174.

Hartig O. (2009). Querying Trust in RDF Data with tSPARQL. In *Extended Semantic Web Conference (ESWC 2009)*, p. 5–20.

Hoffart J., Suchanek F. M., Berberich K., Weikum G. (2013). YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. In *Artificial intelligence*, vol. 194, p. 28–61.

Hogan A., Mellotte M., Powell G., Stampouli D. (2012). Towards Fuzzy Query-relaxation for RDF. In *Extended Semantic Web Conference (ESWC 2012)*, p. 687–702.

Huang H., Liu C., Zhou X. (2012). Approximating query answering on RDF databases. , vol. 15, n° 1, p. 89–114.

Hurtado C. A., Poulovassilis A., Wood P. T. (2009). Ranking Approximate Answers to Semantic Web Queries. In *Extended Semantic Web Conference (ESWC 2009)*, p. 263-277.

Jannach D. (2009). Fast Computation of Query Relaxations for Knowledge-based Recommenders. In *Ai communications*, vol. 22, p. 235–248.

Lehmann J., Isele R., Jakob M., Jentzsch A., Kontokostas D., Mendes P. N. *et al*. (2015). DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. In *Semantic web*, vol. 6, p. 167–195.

Mannila H., Toivonen H. (1997). Levelwise Search and Borders of Theories in Knowledge Discovery. *Data Mining and Knowledge Discovery*, vol. 1, n° 3, p. 241–258.

Pérez J., Arenas M., Gutierrez C. (2009). Semantics and Complexity of SPARQL. In *ACM Transaction on Database Systems (TODS)*, vol. 34, p. 16:1–16:45.

Pham M., Passing L., Erling O., Boncz P. A. (2015). Deriving an Emergent Relational Schema from RDF Data. In *International world wide web conference (www 2015)*, p. 864–874.

Pivert O., Smits G. (2015). How to Efficiently Diagnose and Repair Fuzzy Database Queries that Fail. In *Fifty years of fuzzy logic and its applications, studies in fuzziness and soft computing*, p. 499-517. Springer.

Prud'hommeaux E., Seaborne A. (2008). Sparql query language for rdf (january 2008). *W3C Proposed Recommendation*. Consulté sur http://www.w3.org/TR/rdf-sparql-query/.

Reddy K. B., Kumar P. S. (2013). Efficient Trust-Based Approximate SPARQL Querying of the Web of Linked Data. In *Uncertainty Reasoning for the Semantic Web II*, p. 315–330. Springer.

Saleem M., Ali M. I., Hogan A., Mehmood Q., Ngomo A. N. (2015). LSQ: The Linked SPARQL Queries Dataset. In *International semantic web conference (iswc 2015)*, p. 261–269.

Tomaszuk D., Pak K., Rybiński H. (2013). Trust in RDF graphs. In *Advances in databases and information systems (adbis 2013)*, p. 273–283.

Vrandečić D. (2012). Wikidata: A new platform for collaborative data collection. In *Proceedings of the 21st international conference on world wide web*, p. 1063–1064. New York, NY, USA, ACM.