

---

# 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 Clement Ader, 86960 Futuroscope Cedex, France  
prenom.nom@ensma.fr

---

*RÉSUMÉ.* De nombreuses larges bases de connaissances (BC) incertaines sont disponibles sur le Web où les faits sont associés avec un degré de confiance  $\alpha$ . En général, Les utilisateurs de ces BC n'ayant qu'une connaissance partielle de leur contenu, certaines de leurs requêtes échouent, c'est à dire qu'elles ne renvoient aucun résultat. Pour éviter cette situation frustrante, et au lieu de renvoyer un ensemble vide de réponses, nous avons proposé une approche expliquant l'échec (pour un degré  $\alpha$  et pour plusieurs degrés) en fournissant un ensemble de  $\alpha$ Minimal Failing Subqueries ( $\alpha$ MFS), et en calculant des requêtes alternatives, appelées  $\alpha$ MaXimal Succeeding Subqueries ( $\alpha$ XSS), qui sont aussi proches que possible de la requête initiale. Les expérimentations menées sur le benchmark WatDiv montrent l'intérêt de nos approches par rapport à une méthode de référence.

*ABSTRACT.* Several large uncertain Knowledge Bases (KBs) are available on the Web where facts are associated with a certainty degree  $\alpha$ . Usually, users have only a partial knowledge of the KBs contents, their queries may be failing i.e., they 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  $\alpha$  and for several degrees) with a set of  $\alpha$ Minimal Failing Subqueries ( $\alpha$ MFSs), and computes alternative relaxed queries, called  $\alpha$ MaXimal Succeeding Subqueries ( $\alpha$ XSSs), that are as close as possible to the initial failing query. The conducted experiments on the WatDiv benchmark show the relevance of our approaches compared to a baseline method.

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

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

---

## 1. Introduction

Une Base de Connaissance (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 grandes BC à partir du Web, comme DBpedia (Lehmann *et al.*, 2015) et Knowledge Vault (Dong *et al.*, 2014). Ces BC contiennent des milliards de faits représentés sous forme de triplets RDF (*sujet, prédicat, objet*) et sont interrogés avec le langage SPARQL. Comme ces BC ont été construites à partir de sources externes, leurs faits peuvent être *incertains* (c'est-à-dire potentiellement 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). ces travaux associent ainsi un degré de confiance explicite 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  $\alpha$ . Cependant, comme ces utilisateurs connaissent rarement la structure et le contenu d'une BC, ils peuvent formuler des requêtes trop restrictives et ainsi être confrontés au problème de réponse vide, c'est-à-dire qu'ils n'obtiennent aucun résultat. 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, le système peut l'aider à comprendre les raisons de cet échec. Une de ces approches est basée sur l'identification des sous-requêtes qui échouent (MFS pour *Minimal Failing Subqueries*), et fournit également un ensemble de sous-requêtes possédant des résultats (XSS pour *Maximal Succeeding Subqueries*) (Fokou *et al.*, 2017). Dans cet article, nous nous intéressons à la généralisation de la notion de MFS et de XSS dans le contexte des BC incertaines.

Cet article est organisé 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 auxquelles nos travaux précédents peuvent être directement adaptés pour trouver les  $\alpha$ MFS et  $\alpha$ XSS d'une requête RDF. La section 5 décrit deux approches proposées pour calculer les  $\alpha$ MFS et les  $\alpha$ XSS. La section 6 montre la mise en œuvre et l'évaluation expérimentale réalisée. La section 7 détaille les travaux connexes. Enfin, nous concluons et fournissons quelques perspectives dans la section 8.

## 2. Préliminaires et Problématique

Nous présentons ici les notions (selon les notations de Pérez et al. (Pérez *et al.*, 2009) ) nécessaires à la lecture de l'article et le modèle de confiance utilisé (selon Hartig (Hartig, 2009)).

**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 (indiqués par  $T_{RDF}$ ). Chaque triplet RDF est associé à un *degré de confiance* (ou un *degré de certitude*)

représentant la fiabilité. Ce degré est associé au triplet par une fonction  $tv : T_{RDF} \rightarrow [0, 1]$ .

**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 disjoint 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 triplet:  $Q = t_1 \wedge \dots \wedge t_n$ . Le nombre de patrons de triplet d'une requête  $Q$  est noté  $|Q|$  et les variables de cette dernière sont notés  $var(Q) = \bigcup var(t_i)$ .

**Évaluation d'une requête RDF.** 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 mapping  $\mu(var(t))$ . Le domaine de  $\mu$ ,  $dom(\mu)$ , est un sous-ensemble de  $V$  où  $\mu$  est défini. Deux mappings  $\mu_1$  et  $\mu_2$  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  $\Omega_1$  et  $\Omega_2$  des ensembles de mappings, on définit la *jointure* de  $\Omega_1$  et  $\Omega_2$  par:  $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ sont des mappings compatibles}\}$ . 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  $Q$  une requête, l'évaluation de  $Q$  sur  $D$  est définie par:  $[[Q]]_D = [[t_1]]_D \bowtie \dots \bowtie [[t_n]]_D$ . Cette évaluation peut être effectuée sous différents régimes d'implication tels que définis dans la spécification *SPARQL* (par exemple, le régime d'implication simple ou *RDF*). Soit  $\mu$  une solution de la requête  $Q = t_1 \wedge \dots \wedge t_n$  et *agg* une fonction d'agrégation (par exemple, le min), le degré de confiance de  $\mu$  est défini par  $tv(\mu, Q) = \text{agg}(tv(\mu(t_1)), \dots, tv(\mu(t_n)))$ . L'évaluation de  $Q$  sur  $D$  retournant les résultats pondérés par une confiance pour un seuil  $\alpha$  est définie par:  $[[Q]]_D^\alpha = \{\mu \in [[Q]]_D \mid tv(\mu) \geq \alpha\}$ .

**Notions de  $\alpha$ MFS et  $\alpha$ XSS.** Soit une requête  $Q = t_1 \wedge \dots \wedge t_n$ , une requête  $Q' = t_i \wedge \dots \wedge t_j$  est une *sous requête* de  $Q$ , i.e.  $Q' \subseteq Q$ , ssi  $\{i, \dots, j\} \subseteq \{1, \dots, n\}$ . Si  $\{i, \dots, j\} \subset \{1, \dots, n\}$ ,  $Q'$  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  $\alpha$  *requête minimale échouant ( $\alpha$ MFS) MFS* d'une requête  $Q$  pour un  $\alpha$  donné est définie par:  $[[MFS]]_D^\alpha = \emptyset \wedge \nexists Q' \subset MFS$  tel que  $[[Q']]_D^\alpha = \emptyset$ . L'ensemble de toutes les  $\alpha$ MFS de  $Q$  est noté par  $mfs^\alpha(Q)$ .

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

**Problématique.** Dans le cadre des BC incertaines nous nous intéressons au calcul des  $mfs^{\alpha_i}(Q)$  et des  $xss^{\alpha_i}(Q)$  pour un ensemble de seuils  $\{\alpha_1, \dots, \alpha_n\}$ .

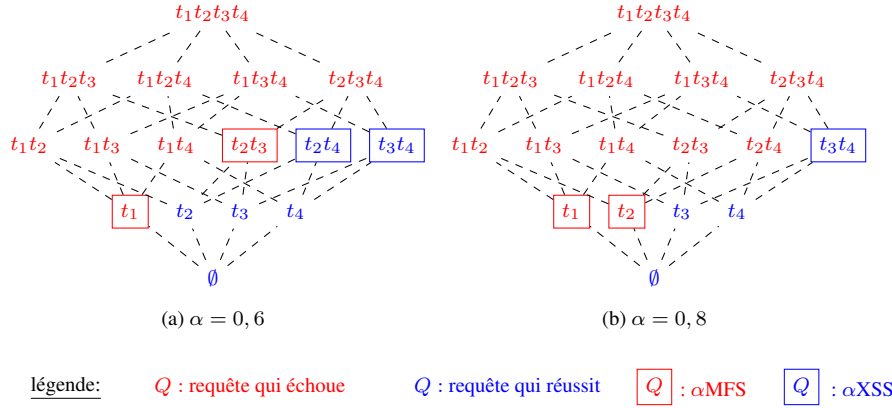


Figure 1. Treillis des sous-requêtes de  $Q$  pour différents  $\alpha$

### 3. Motivation

Considérons la requête suivante qui recherche les livres édités par Springer et écrits par Smith avec leurs nombres de pages. 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).

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

Les  $\alpha$ MFS et  $\alpha$ XSS pour deux seuils de cette requête sont présentées dans la figure 1 sous forme de treillis des 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  $\alpha$ MFS et  $\alpha$ XSS, nous pouvons fournir les explications suivantes à l'utilisateur pour un degré de confiance de 0,8 :

1. Il n'existe aucun élément écrit par Smith.
2. Il n'existe aucun élément édité par Springer.
3. Il existe cependant des livres avec leurs nombres de pages.

Si nous calculons les  $\alpha$ MFS et  $\alpha$ XSS pour le degré 0,6, en plus des retours n° 1 et 3 précédents :

1. Il n'existe aucun livre édité par Springer.
2. Il existe cependant des éléments édités par Springer avec leur nombre de page.

Ces retours peuvent aider l'utilisateur à reformuler sa requête, à ajuster ses attentes ou à mieux appréhender le contenu de la base de connaissances.

#### 4. Calcul des $\alpha$ MFS et $\alpha$ XSS pour un seuil $\alpha$

Dans cette section, nous proposons dans un premier temps une adaptation de l'algorithme *Lattice-Based Approach* (LBA) proposé dans (Fokou *et al.*, 2017) pour calculer les  $\alpha$ MFS et  $\alpha$ XSS d'une requête pour un  $\alpha$  donné. Notre adaptation possède la même complexité algorithmique que l'algorithme original. Notons que nous aurions aussi pu utiliser d'autres algorithmes et notamment ceux conçus pour la découverte d'ensembles fréquents maximaux comme, par exemple, *Dualize and Advance* (Gunopulos *et al.*, 2003). L'algorithme LBA présente l'avantage d'être spécifiquement optimisé pour la découverte des MFS et XSS.

##### 4.1. L'approche $\alpha$ LBA

Nous présentons ici le fonctionnement de notre algorithme  $\alpha$ LBA comme une adaptation directe de LBA dans le contexte des BC incertaines.  $\alpha$ LBA explore le treillis des sous-requêtes d'une requête  $Q$  en suivant trois étapes.

---

**Algorithme 1** : Découverte d'une  $\alpha$ MFS pour une requête  $Q$  qui échoue

---

**TrouverUne $\alpha$ MFS**( $Q, D, \alpha$ )

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

**sorties** : Une  $\alpha$ MFS de  $Q$  notée par  $Q^*$

1    $Q^* \leftarrow \emptyset; Q' \leftarrow Q;$

2   **foreach** *patron de triplet*  $t_i \in Q$  **do**

3      $Q' \leftarrow Q' - t_i;$

4     **if**  $[[Q' \wedge Q^*]]_D^\alpha \neq \emptyset$  **then**

5        $Q^* \leftarrow Q^* \wedge t_i;$

6   **return**  $Q^*;$

---

**1. Trouver une  $\alpha$ MFS  $Q^*$  de  $Q$ .** Suivant l'algorithme 1,  $\alpha$ LBA supprime itérativement chaque patron de triplet  $t_i$  de  $Q$ , ce qui l'emmène à évaluer des sous-requêtes propres  $Q' \wedge Q^*$  de  $Q$ . Si  $Q' \wedge Q^*$  échoue pour  $\alpha$ , alors  $Q' \wedge Q^*$  contient une  $\alpha$ MFS. Inversement, si  $Q' \wedge Q^*$  réussit, alors chaque  $\alpha$ MFS 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.

**2. Calculer les  $\alpha$ XSS potentielles** c'est-à-dire les requêtes maximales qui ne sont pas des super-requêtes de la  $\alpha$ MFS précédemment trouvée. L'ensemble des  $\alpha$ XSS potentielles est noté  $pxss(Q, Q^*)$ . Cet ensemble peut être calculé comme suit:

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

Cette deuxième étape est basée sur le fait que toutes les super-requêtes de  $Q^*$  (c'est-à-dire les requêtes incluant la  $\alpha$ MFS  $Q^*$  identifiée à 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. En effet, il est nécessaire qu'une requête, qui réussisse, ne puisse pas contenir de sous-requête qui échoue pour le  $\alpha$  donné. Cette condition est détaillée dans la section 4.2

**3. Tester les potentielles  $\alpha$ XSS.** Si une sous-requête trouvée lors de l'étape 2 réussit, alors il s'agit d'une  $\alpha$ XSS. Sinon, nous appliquons les deux étapes précédentes sur cette sous-requête pour trouver une nouvelle  $\alpha$ MFS ainsi que de nouvelles  $\alpha$ XSS potentielles qui lui sont associées. L'algorithme 2 illustre cette étape.

---

**Algorithme 2 :** Trouver les  $\alpha$ MFS et  $\alpha$ XSS d'une requête  $Q$

---

```

 $\alpha$ LBA( $Q, D, \alpha$ )
  entrées : Une requête qui échoue  $Q = t_1 \wedge \dots \wedge t_n$ ;
           une base de données RDF  $D$ ;
           un seuil  $\alpha$ 
  sorties : les  $\alpha$ MFS et  $\alpha$ XSS de  $Q$ 
1   $pxss \leftarrow \{Q\}; mfs^\alpha(Q) \leftarrow \emptyset; xss^\alpha(Q) \leftarrow \emptyset;$ 
2  while  $pxss \neq \emptyset$  do
3       $Q' \leftarrow pxss.element();$  // choisir une  $xss$  potentielle
4      if  $[[Q']]_D^\alpha \neq \emptyset$  then //  $Q'$  est une  $\alpha$ XSS
5           $xss^\alpha(Q) \leftarrow xss^\alpha(Q) \cup \{Q'\}; pxss \leftarrow pxss - \{Q'\};$ 
6      else //  $Q'$  contient au moins une  $\alpha$ MFS
7           $Q^* \leftarrow \text{TrouverUne}\alpha\text{MFS}(Q', D, \alpha);$ 
8           $mfs^\alpha(Q) \leftarrow mfs^\alpha(Q) \cup \{Q^*\};$ 
           // mise à jour des  $pxss$ 
9          foreach  $Q'' \in pxss$  tel que  $Q^* \subseteq Q''$  do
10              $pxss \leftarrow pxss - \{Q''\};$ 
11              $pxss \leftarrow pxss \cup \{Q_j \in pxss(Q'', Q^*) \mid \nexists Q_k \in$ 
                 $pxss \cup xss^\alpha(Q) \text{ tel que } Q_j \subseteq Q_k\};$ 
12  return  $\{mfs^\alpha(Q), xss^\alpha(Q)\};$ 

```

---

#### 4.2. Contraintes sur la fonction d'agrégation

Comme indiqué dans la section précédente, l'algorithme  $\alpha$ LBA repose sur le fait qu'une requête qui réussit ne peut pas contenir une requête qui échoue. Dans le contexte des BC incertaines, selon la fonction d'agrégation (*aggreg*) choisie, cette propriété n'est pas toujours vraie. 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 incertaine pour  $\alpha = 0, 4$ . Pour les fonctions d'agrégation *max* et *avg*,  $Q$  réussit alors

BC incertaine			
sujet	prédicat	object	tv
b <sub>1</sub>	type	Book	0.3
b <sub>1</sub>	nbPages	90	0.3
b <sub>2</sub>	type	Book	0.3
b <sub>2</sub>	nbPages	90	0.9
b <sub>3</sub>	type	Book	0.2
b <sub>3</sub>	nbPages	88	0.9
b <sub>4</sub>	type	Book	0.1
b <sub>4</sub>	nbPages	90	0.6
b <sub>5</sub>	type	Website	0.8
b <sub>5</sub>	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

que sa sous-requête  $Q'$  échoue. Ainsi, l'algorithme  $\alpha$ LBA ne peut pas être utilisé avec ces fonctions.

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

Par définition, une fonction décroissante est monotone. Les fonctions *minimum* et *produit* appliquées aux valeurs  $\in [0, 1]$  sont des exemples de fonctions d'agrégation décroissantes.

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

**PREUVE 5.** — On considère  $Q = t_1 \wedge \dots \wedge t_n$  et sa sous-requête propre  $Q' = t_i \wedge \dots \wedge t_j$  ( $\{i, \dots, j\} \subset \{1, \dots, n\}$ ). Supposons que  $Q'$  échoue et que  $Q$  réussisse :  $[[Q']]_D^\alpha = \emptyset$  et  $[[Q]]_D^\alpha \neq \emptyset$ . Donc,  $\exists \mu \in [[Q]]_D^\alpha$ . Étant donné que  $[[Q]]_D^\alpha \subseteq [[Q]]_D$  et  $[[Q]]_D \subset [[Q']]_D$ , nous avons  $\mu|_{var(Q')} \in [[Q']]_D$  où  $\mu|_{var(Q')}$  est la restriction de la fonction  $\mu$  aux variables de  $Q'$ . Par définition,  $tv(\mu, Q) = aggrege(tv(\mu(t_1)), \dots, tv(\mu(t_n))) \geq \alpha$  et  $tv(\mu|_{var(Q')}, Q') = aggrege(tv(\mu(t_i)), \dots, tv(\mu(t_j)))$  (en effet,  $tv(\mu, Q') = tv(\mu|_{var(Q')}, Q')$ ). Étant donné que  $aggreg$  est décroissante,  $aggrege(tv(\mu(t_i)), \dots, tv(\mu(t_j))) \geq aggrege(tv(\mu(t_1)), \dots, tv(\mu(t_n))) \geq \alpha$ . 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. ■

Nous passons maintenant au problème de l'identification des  $\alpha$ MFS et  $\alpha$ XSS pour un ensemble de seuils  $\alpha$ . En plus des causes d'échec basées sur les patrons de triplet,

1. Pour simplifier, cette définition est limitée aux ensembles mais pourrait être étendue aux multiset

les retours sur des seuils multiples peuvent permettre à l'utilisateur de réévaluer le seuil de confiance associé à sa requête.

## 5. Calcul des $\alpha$ MFS et $\alpha$ XSS pour différents seuils $\alpha$

Afin de trouver les  $\alpha$ MFS et  $\alpha$ XSS pour un ensemble de  $\alpha$ :  $\{\alpha_1, \dots, \alpha_n\}$ , la solution naïve est d'exécuter l'algorithme  $\alpha$ LBA pour chaque  $\alpha_i$ . Cette solution de base est appelée *NLBA*. Dans cette section, nous discutons différentes améliorations à cette approche. L'idée est d'utiliser les  $\alpha$ MFS et  $\alpha$ XSS trouvées pour un seuil donné pour déduire celles d'un seuil supérieur (ou inférieur). Nous commençons par étudier une approche ascendante, pour laquelle les seuils sont considérés par ordre croissant.

### 5.1. Approche ascendante

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

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

**PREUVE 7.** — Si  $Q^*$  est une  $\alpha_i$ MFS 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  $\alpha_j$ , ainsi  $Q^*$  est une  $\alpha_j$ MFS de  $Q$ . ■

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  $\alpha_j$ MFS correspondantes à l'ensemble des  $\alpha$ MFS découvertes, noté  $dmfs^{\alpha_j}(Q)$ . Dans le cas contraire ( $|Q^*| \geq 2$ ), prouver que  $Q^*$  est une  $\alpha_j$ MFS 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  $\alpha_j$ MFS. En revanche, l'algorithme *TrouverUne $\alpha$ MFS* de  $\alpha$ LBA (algorithme 1) requiert lui aussi l'exécution de  $|Q^*|$  requêtes mais garantit qu'une  $\alpha$ MFS soit trouvée. Ainsi, notre approche privilégie *TrouverUne $\alpha$ MFS* par rapport à l'exécution des sous-requêtes de la  $\alpha_i$ MFS, comme nous le verrons dans l'algorithme 3.

Quant aux  $\alpha$ XSS, une  $\alpha_i$ XSS de  $Q$  ne réussit pas nécessairement pour le seuil  $\alpha_j$ . La proposition suivante indique que si cette  $\alpha_i$ XSS réussit, elle est aussi une  $\alpha_j$ XSS.

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

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



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

Ainsi, la vérification qu'une  $\alpha_i$ XSS soit aussi une  $\alpha_j$ XSS requiert l'exécution d'une seule requête. Ceci nous permet de trouver un ensemble de  $\alpha_j$ XSS découvertes, notés  $dxss^{\alpha_j}(Q)$ .

L'algorithme 3 présente notre approche complète pour trouver un ensemble de  $\alpha_j$ MFS et de  $\alpha_j$ XSS en se basant sur les  $\alpha_i$ MFS et  $\alpha_i$ XSS. Toutes les  $\alpha_i$ MFS composées d'un seul patron de triplet (*requêtesElementaire*) sont insérées dans  $dmfs^{\alpha_j}(Q)$  (ligne 1). L'algorithme itère ensuite sur les  $\alpha_i$ MFS possédant au moins deux patrons de triplet (l'ensemble  $FQ$ ) pour lesquelles il cherche une  $\alpha_j$ MFS  $Q^*$  (ligne 6). Les sur-requêtes de  $Q^*$  sont alors retirées de l'ensemble  $FQ$  car elles ne peuvent plus être minimales. Les  $\alpha_j$ XSS sont ensuite identifiées en exécutant chaque  $\alpha_i$ XSS et en conservant celles qui réussissent pour le seuil  $\alpha_j$  (lignes 10-11).

---

**Algorithme 3 :** Découvrir des  $\alpha_j$ MFS et  $\alpha_j$ XSS pour l'approche ascendante

---

**Découvrir** $\alpha$ MFSXSS( $mfs^{\alpha_i}(Q)$ ,  $xss^{\alpha_i}(Q)$ ,  $D$ ,  $\alpha_j$ )

**entrées :** Les  $\alpha_i$ MFS  $mfs^{\alpha_i}(Q)$  d'une requête  $Q$  pour un seuil  $\alpha_i$ ;  
 Les  $\alpha_i$ XSS  $xss^{\alpha_i}(Q)$  d'une requête  $Q$  pour un seuil  $\alpha_i$ ;  
 une base de données RDF  $D$ ;  
 un seuil  $\alpha_j > \alpha_i$

**sorties :** Un ensemble de  $\alpha_j$ MFS de  $Q$  noté  $dmfs^{\alpha_j}(Q)$ ;  
 Un ensemble de  $\alpha_j$ XSS de  $Q$  noté  $dxss^{\alpha_j}(Q)$ ;

```

1  requêtesElementaire  $\leftarrow \{Q_a \in mfs^{\alpha_i}(Q) \mid |Q_a| = 1\}$ ;
2   $dmfs^{\alpha_j}(Q) \leftarrow requêtesElementaire$ ;
3   $FQ \leftarrow mfs^{\alpha_i}(Q) - requêtesElementaire$ ;
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  $\alpha_j$ MFS et  $\alpha_j$ XSS, une version optimisée de  $\alpha$ LBA est exécutée en prenant en entrée les  $\alpha_j$ MFS et  $\alpha_j$ XSS découvertes (voir l'algorithme 4). Cet algorithme calcule les  $\alpha_j$ XSS potentielles ( $pxss$ ) qui ne contiennent aucune  $\alpha_j$ MFS (lignes 2-6), supprime de cet ensemble les  $\alpha_j$ XSS (ligne 7), puis, itère sur l'ensemble  $pxss$  comme pour la version originale de  $\alpha$ LBA (voir l'algorithme 2).

---

**Algorithme 4** : Version améliorée de  $\alpha$ LBA

---

**Optimized- $\alpha$ LBA**( $Q, D, \alpha, dmfs^\alpha(Q), dxss^\alpha(Q)$ )

**entrées** : Une requête qui échoue  $Q$ ;  
une base de données RDF  $D$ ;  
un seuil  $\alpha$   
un ensemble de  $\alpha$ MFS de  $Q$  noté  $dmfs^\alpha(Q)$ ;  
un ensemble de  $\alpha$ XSS de  $Q$  noté  $dxss^\alpha(Q)$ ;

**sorties** : Les  $\alpha$ MFS et  $\alpha$ XSS de  $Q$

```
1  $mfs^\alpha(Q) \leftarrow dmfs^\alpha(Q); xss^\alpha(Q) \leftarrow dxss^\alpha(Q);$   
2  $Q^* \leftarrow dmfs^\alpha(Q).dequeue(); pxss \leftarrow pxss(Q, Q^*);$   
3 foreach  $Q^* \in dmfs^\alpha(Q)$  do  
4   foreach  $Q' \in pxss$  tel que  $Q^* \subseteq Q'$  do  
5      $pxss \leftarrow pxss - \{Q'\};$   
6      $pxss \leftarrow pxss \cup \{Q_i \in pxss(Q', Q^*) \mid \nexists Q_j \in pxss \cup xss^\alpha(Q) :$   
7        $Q_i \subseteq Q_j\};$   
7    $pxss \leftarrow pxss - dxss^\alpha(Q);$   
8   while  $pxss \neq \emptyset$  do  
9     // idem que les lignes 3-11 de  $\alpha$ LBA  
9   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  $\alpha$ MFS et  $\alpha$ XSS, nous montrons comment l'algorithme ascendant calcule les 0,8  $\alpha$ MFS et  $\alpha$ XSS. Comme  $t_1$  est une 0,6  $\alpha$ MFS et ne contient qu'un seul patron de triplet, cette dernière est une 0,8  $\alpha$ MFS (proposition 6). la seconde 0,6  $\alpha$ MFS est  $t_2t_3$ . Comme cette requête échoue nécessairement pour 0,8, nous utilisons l'algorithme *TrouverUne $\alpha$ MFS* pour identifier la 0,8  $\alpha$ MFS  $t_2$ . Compte tenu des 0,6  $\alpha$ XSS, seule  $t_3t_4$  réussit pour 0,8. Ainsi,  $t_3t_4$  est une 0,8  $\alpha$ XSS (proposition 8). Les  $\alpha$ MFS et  $\alpha$ XSS 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  $\alpha$ XSS potentielles (lignes 1-7) et ainsi, que toutes les 0,8  $\alpha$ MFS et  $\alpha$ XSS ont été trouvées.

## 5.2. Approche descendante

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

- les  $\alpha_j$ MFS qui échouent pour  $\alpha_j$  sont des  $\alpha_j$ MFS;
- les  $\alpha_j$ XSS d’une taille  $|Q| - 1$  sont des  $\alpha_j$ XSS;
- les  $\alpha_j$ XSS d’une taille  $< |Q| - 1$  réussissent aussi pour  $\alpha_j$  et contiennent donc une  $\alpha_j$ XSS.

Une fois un ensemble de  $\alpha_j$ MFS et de  $\alpha_j$ XSS découvert, l’algorithme Optimized- $\alpha$ LBA (algorithme 4) est exécuté pour trouver les  $\alpha_j$ MFS et  $\alpha_j$ XSS restantes.

## 6. Évaluation expérimentale

Dans cette section, nous étudions les performances de nos deux approches en les comparant avec l’approche de base *NLBA* (exécution de  $\alpha$ LBA pour chacun des  $N$  seuils).

**Environnement expérimental.** Nous avons implémenté nos algorithmes avec JAVA 1.8 64 bits. Les algorithmes prennent en entrée une requête  $Q$  qui échoue et un ensemble de seuils, et renvoient les ensembles de  $\alpha$ MFS et  $\alpha$ XSS de  $Q$  pour chaque seuil. Dans l’implémentation actuelle, les requêtes sont exécutées 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 permettant de 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.

**Données et requêtes.** Nous avons utilisé un jeu de données de 20M triplets 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 qui échouent (voir Tableau 1 en annexe). Ces requêtes contiennent entre 1 et 15 patrons de triplet et couvrent les principaux types de requêtes: étoile (jointure *sujet-sujet* entre les patrons de triplet), chaîne (jointure *objet-objet*) et composite (d’autres jointures).

**Expérimentation.** Notre expérimentation évalue nos algorithmes pour les seuils  $\{0, 2; 0, 4; 0, 6; 0, 8\}$  et un jeu de données de 20M. La figure 3a montre le temps d’exécution de chaque algorithme pour chaque requête. La figure 3b donne le nombre de requêtes exécutées.

Cette expérimentation montre les gains apportés par nos algorithmes par rapport à la méthode de base NLBA : nos algorithmes exécutent moins de requêtes pour trouver les  $\alpha$ MFS et  $\alpha$ XSS, soit en moyenne respectivement 39%, 40% et 39% moins de requêtes. Par conséquent, ces algorithmes ont des temps d’exécutions inférieurs, ré-

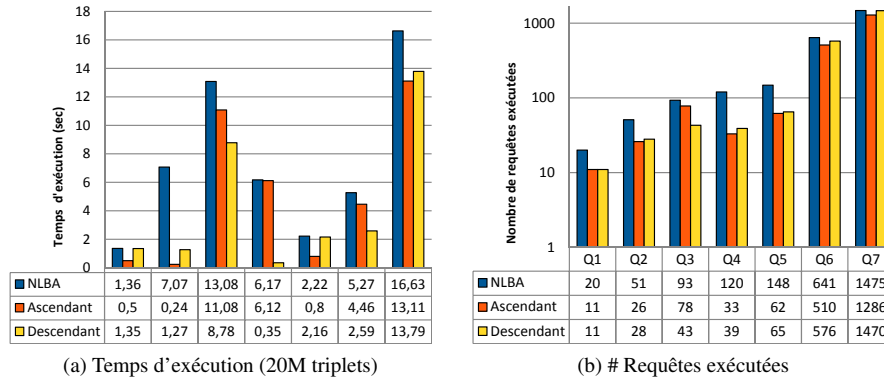


Figure 3. Résultats expérimentaux avec Jena TDB

duisant ainsi le temps d'exécution total de 30% et 42% respectivement. Par exemple, NLBA a besoin de 7 secondes pour trouver les  $\alpha$ MFS et  $\alpha$ XSS 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 évitent d'exécuter. Par exemple, nos algorithmes exécutent moins de 40 requêtes pour Q4 tandis que NLBA en exécute 120, pourtant, l'approche ascendante améliore le temps d'exécution de moins de 1%. Pour l'algorithme descendant, le gain de performance est significatif (94%). 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 expérimentation montre également qu'aucun de nos algorithmes ne fournit le meilleur résultat pour toutes les requêtes : l'approche ascendante est meilleure pour Q1, Q2 et Q5 tandis que l'approche descendante est meilleure pour Q3, Q4 et Q6. Malgré l'exécution de peu de requêtes, l'algorithme ascendant possède un temps d'exécution total supérieur à l'algorithme descendant. Ceci est dû au fait que ces algorithmes exécutent différentes requêtes avec des temps d'exécution différents. En particulier, l'algorithme descendant commence par les seuils les plus élevés. Les requêtes exécutées tendent à être sélectives étant donné que le critère de confiance est restrictif, et par conséquent, elles ont des temps d'exécution courts. Une fois les  $\alpha$ MFS et  $\alpha$ XSS trouvées pour les seuils les plus élevés, l'algorithme descendant évite ainsi l'exécution de requêtes avec un seuil inférieur, susceptibles d'être plus coûteuses. Comme l'algorithme ascendant suit l'approche duale, il tend à exécuter des requêtes peu sélectives avec des coûts plus importants.

## 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), un schéma relationnel émerge 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.

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.*, 2017 ; 2016) 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 la condition pour laquelle cet algorithme peut 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.

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 sommes 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  $\alpha$  attendu. Pour fournir à l'utilisateur un retour pertinent, nous avons proposé de calculer les  $\alpha$ MFS et  $\alpha$ XSS 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 la condition pour laquelle l'algorithme de nos travaux précédents, appelé  $\alpha$ LBA, peut 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  $\alpha$ MFS et  $\alpha$ XSS pour plusieurs seuils. La méthode de base, appelée NLBA, consiste à exécuter  $\alpha$ LBA pour chaque seuil. Cependant, nous avons observé et prouvé que les  $\alpha$ MFS et  $\alpha$ XSS pour un seuil donné peuvent être réutilisées pour trouver celles d'un seuil inférieur (ou supérieur). Ainsi, nous avons défini deux approches alternatives de NLBA, appelées ascendante et descendante, qui considèrent des seuils  $\alpha$  dans l'ordre croissant ou décroissant. 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 l'appliquant sur un contexte réel comme, par exemple, avec des requêtes exécutées sur la base de connaissances YAGO. 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 dans lesquelles 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 *Iswc'14*, p. 197–212.
- Calí A., Frosini R., Poulouvasilis A., Wood P. (2014). Flexible Querying for SPARQL. In *Odbase'14*, p. 473–490.
- Campinas S. (2014). Live SPARQL Auto-Completion. In *Iswc'14*, p. 477–480.
- Dolog P., Stuckenschmidt H., Wache H., Diederich J. (2009). Relaxing RDF queries based on user and domain preferences. *Journal of Intelligent Information Systems (JIIS)*, vol. 33, n° 3, 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 *Kdd'14*, p. 601–610.
- Elbassuoni S., Ramanath M., Weikum G. (2011). Query Relaxation for Entity-Relationship Search. In *ESWC'11*, p. 62–76.
- Fokou G., Jean S., Hadjali A. (2014). Endowing Semantic Query Languages with Advanced Relaxation Capabilities. In *ISMIS'14*, p. 512–517.
- Fokou G., Jean S., Hadjali A., Baron M. (2016). RDF Query Relaxation Strategies Based on Failure Causes. In *Eswc'16*, p. 439–454.
- Fokou G., Jean S., Hadjali A., Baron M. (2017). Handling Failing RDF Queries: From Diagnosis to Relaxation. *Knowledge and Information Systems (KAIS)*, vol. 50, n° 1.
- Galárraga L., Teflioudi C., Hose K., Suchanek F. M. (2015). Fast Rule Mining in Ontological Knowledge Bases with AMIE+. *VLDB Journal*, vol. 24, n° 6, p. 707–730.
- 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 *ESWC 2009*.
- Hogan A., Mellotte M., Powell G., Stampouli D. (2012). Towards Fuzzy Query-relaxation for RDF. In *ESWC'12*, p. 687–702.
- Huang H., Liu C., Zhou X. (2012). Approximating query answering on RDF databases. *Journal of the World Wide Web*, vol. 15, n° 1, p. 89–114.
- Hurtado C. A., Poulouvassilis A., Wood P. T. (2009). Ranking Approximate Answers to Semantic Web Queries. In *ESWC'09*, p. 263–277.
- 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. *Semantic Web*, vol. 6, n° 2, 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. *ACM Transaction on Database Systems (TODS)*, vol. 34, n° 3, p. 16:1–16:45.
- Pham M., Passing L., Erling O., Boncz P. A. (2015). Deriving an Emergent Relational Schema from RDF Data. In *Www'15*, p. 864–874.
- 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.
- Saleem M., Ali M. I., Hogan A., Mehmood Q., Ngomo A. N. (2015). LSQ: The Linked SPARQL Queries Dataset. In *Iswc'15*, p. 261–269.
- Tomaszok D., Pak K., Rybiński H. (2013). Trust in RDF graphs. In *Adbis'13*.

## Annexe

Tableau 1. 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