Gestion et exploitation de larges bases de connaissances en présence de données incomplètes et incertaines

Ibrahim Dellal

THÈSE

pour l'obtention du Grade de

Docteur de l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique

(Diplôme National - Arrêté du 25 mai 2016)

Ecole Doctorale : Sciences et Ingéniérie des Systèmes, Mathématiques, Informatique
Secteur de Recherche : Informatique et Applications

Présentée par :

Ibrahim DELLAL

****

Gestion et Exploitation de Larges Bases de Connaissances en Présence de Données Incomplètes et Incertaines

****

Directeur de thèse : **Allel HADJALI**
Co-encadrants de thèse : **Stéphane JEAN et Brice CHARDIN**

****

Soutenue le 10 décembre 2019
devant la Commission d'Examen

****

JURY

Président :	
François GOASDOUÉ	Professeur, Université de Rennes 1
Rapporteurs :	
Nadine CULLOT	Professeur, Université de Bourgogne
Guy DE TRÉ	Professeur, Université Gent
Membres du jury :	
Brice CHARDIN	Maître de conférences, ISAE-ENSMA
Allel HADJALI	Professeur, ISAE-ENSMA
Stéphane JEAN	Maître de conférences, Université de Poitiers
Béatrice MARKHOFF	Maître de conférences - HDR, Université de Tours

Remerciements

Au terme de ma thèse, je tiens à remercier :

- Mes directeurs de thèse Allel Hadjali, Stéphane Jean et Brice Chardin pour m'avoir suivi durant ces trois années. Je les remercie pour tout ce qu'ils m'ont apporté.

- Mes rapporteurs Nadine CULLOT et Guy DE TRE pour avoir accepté cette tâche, avoir pris le temps de relire et de comprendre ma thèse et pour meurs retours et commentaires si constructifs.

- Mes examinateurs Béatrice MARKHOFF et François GOUASDOUE d'avoir accepté d'être dans mon jury et de m'avoir orienté vers d'autres perspectives de mon travail à travers leurs questions.

- Je tiens à remercier Ladjel Bellatreche responsable de l'équipe IDD pour ses conseils et pour m'avoir transmis son enthousiasme et sa passion pour la recherche.

- Également, je remercie Mickael Baron pour tout le soutient et l'aide technique qu'il m'a apportés tout au long de ma thèse.

- Je remercie le directeur du laboratoire, Emmanuel Grolleau, pour sa bienveillance, sa largesse et son esprit très ouvert.

- Un Grand merci à mes collègues et mes amis Anh Toan et Jorge, qui m'ont accompagné pendant ces années, et pour leur bonne humeur au travail et hors travail.

- Claudine Rault et Bénédicte Boinot pour leur convivialité, leur générosité et toutes les tâches qu'elles ont effectuées pour moi.

- Un merci Spécial à tonton Mamar, Hamid, Zied, Zizou, Larbi, Youcef Darbouka et à mon poto Ghani qui ont grandement contribué à la réussite de ma thèse.

- Tous les permanents du LIAS pour leur convivialité et leurs précieux conseils.

- Tous mes amis doctorants qui m'ont accompagné, soutenu et grâce à qui ces trois années m'ont paru courtes.

- Mes parents, Ali et Malika qui ont fait d'énormes sacrifices pour mon éducation, qui m'ont toujours montré la valeur du travail et appris le goût de l'effort. Ils m'ont constamment encouragé et donné l'amour et la confiance dont j'avais besoin à chaque fois pour aller de l'avant. À vous mes parents, je vous dis mille fois merci.

- Mes frères et soeurs Mohamed, Khadidja, Safia, Abdelkader et Zakaria qui ont su maintenir la joie et la bonne humeur à la maisons pendant que moi j'étais absent.

Table des matières

Liste des figures

Liste des tableaux

Introduction

Contexte

L'émergence et la prolifération des applications du Web sémantique ont conduit à la construction de nombreuses larges bases de connaissances (BC) accessibles via le Web. Parmi les BC les plus connues, citons DBpedia [LIJ+15], YAGO [SKW07], Wikidata [Vra12b] et NELL [CBK+10] construites dans le cadre de projets académiques. Des BC ont également été conçues dans le cadre des projets commerciaux telles que celles définies par Google [DGH+14a] et Walmart [DLT+13a]. Ces BC contiennent des entités (nommées) et des faits sur ces entités. Elles contiennent également les classes sémantiques de ces entités et leurs liens mutuels. De plus, plusieurs BC peuvent être interconnectées au niveau entités, formant ainsi le noyau du Web des données liées.

Une caractérisation essentielle de ces BC est qu'elles contiennent des millions à des billions de faits représentés sous forme de triplets RDF (sujet, prédicat, objet) et peuvent être interrogées au moyen du langage SPARQL [SH13]. Par exemple, en 2019, YAGO possède une connaissance sur plus de 10 millions d'entités et contient plus de 120 millions de faits sur ces entités (source Wikipédia). Comme ces BC sont construites au moyen d'un processus d'extraction et de fusion d'informations provenant de multiples sources externes (et notamment celles du Web), leurs faits peuvent être entachés d'incertitude (c'est-à-dire, dont la véracité ou la cohérence n'est pas totale). A titre d'exemple, l'information dans la BC YAGO est extraite à partir de trois sources : Wikipedia, WordNet et GeoNames. Pour prendre en compte le caractère incertain et le manque de confiance des éléments de ces BC, des extensions du formalisme RDF et du langage SPARQL ont été proposées afin de supporter des données pondérées par une confiance [Har09, TPR13]. Ainsi, un degré de confiance explicite est associé aux faits de la BC cible et aux résultats des requêtes SPARQL.

Problématique

Lors de l'interrogation des BC Incertaines (BCI), les utilisateurs s'attendent à obtenir des résultats de qualité en termes de satisfaction et de fiabilité, c'est-à-dire des résultats satisfaisant au mieux leurs requêtes et possédant des niveaux de confiance supérieurs à un seuil donné α (défini par l'utilisateur). Cependant, comme ces utilisateurs connaissent rarement la structure et le contenu de la BC cible, ils peuvent formuler des requêtes trop restrictives ou trop permissives et ainsi être confrontés au problème de réponses insatisfaisantes ou inexploitables. Dans le cas des requêtes trop sélectives, les utilisateurs n'obtiennent aucun résultat ou seulement des résultats avec un degré de confiance inférieur au seuil donné. Ce problème est connu sous le nom de Problème à Réponse Vide (PRV). Une étude réalisée par Saleem et al. [SAH+15] sur les points d'entrée SPARQL montre que 10% des requêtes soumises à DBpedia entre mai et juillet 2010 retournaient des résultats vides. Au lieu de retourner à l'utilisateur un ensemble vide comme réponse à sa requête, un traitement coopératif et intelligent peut l'aider à comprendre les rai-

sons 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. Cette approche a été proposée dans [FJHB16] dans le contexte des BC Traditionnelles (BCT). 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 des résultats non vides. L'utilisateur peut ainsi choisir d'exécuter l'une de ces requêtes pour obtenir une réponse non vide.

Quant aux requêtes trop peu sélectives, les utilisateurs se voient noyés par une pléthore de réponses dont l'examen et l'exploitation à des fins de décision s'avèrent compliqués et difficiles. Ce problème est connu sous le nom de Problème de Réponses Pléthoriques (PRP). La solution à ce problème consiste à réduire l'ensemble des réponses en un ensemble de taille acceptable et facilement exploitable. De nombreux travaux ont été proposés dans la littérature pour traiter ce problème dans le cadre des bases de données et des bases de connaissances BCT. Trois familles de travaux peuvent être distinguées : (i) les travaux orientés requête qui agissent sur la requête par augmentation ou intensification afin de la rendre plus sélective ; (ii) les travaux visant à introduire un ordre dans l'ensemble des réponses pour en sélectionner les K meilleures ; (iii) les travaux orientés résultats qui agissent sur l'ensemble des réponses obtenues pour en choisir un sous ensemble de taille raisonnable selon un critère bien défini, comme par exemple, la diversité.

Motivation et contributions

Le processus de construction des bases BCI a connu beaucoup de progrès et a bénéficié de techniques et d'outils d'extraction d'information avancés combinés avec des méthodes d'analyse statistique et de fusion pertinentes. Cependant, l'exploitation des BCI n'a fait l'objet que de très peu de travaux. La plupart de ces travaux ne répondent pas, d'une manière satisfaisante, à toutes les attentes des utilisateurs dans les différentes situations possibles et n'exhibent aucun comportement coopératif et intelligent pour le traitement de certaines requêtes problématiques. Les travaux de cette thèse s'inscrivent dans le cadre de l'interrogation et de l'exploitation des bases BCI. Ils visent à améliorer l'ergonomie et la convivialité des systèmes modernes d'exploitation des BCI. Plus particulièrement, ils proposent des techniques coopératives et intelligentes aidant l'utilisateur dans ses prises de décisions quand ses recherches retournent des résultats insatisfaisants en termes de quantité ou de fiabilité.

Nous avons considéré comme point de départ les travaux de Fokou [FJHB16] pour le traitement du problème PRV dans le cadre des bases BCT. Nous avons étendu les notions de MFS et de XSS dans le contexte des bases BCI pour expliquer l'échec de la requête utilisateur et pour proposer des requêtes alternatives, sémantiquement proches à la requête initiale, dont les réponses sont non vides. Cette extension a d'abord été réalisée dans le cas des requêtes paramétrées avec un seul degré de confiance, puis avec un ensemble de degrés de confiance pour offrir plus de flexibilité aux utilisateurs dans leurs prises de décisions. Dans un second temps, nous avons abordé l'étude du problème dual au problème cité ci-dessus, c'est-à-dire le problème PRP. Des contreparties des MFS et des XSS ont été établies, ce qui a permis d'identifier, d'une part, les causes du problème et, d'autres part, des requêtes alternatives dont les résultats peuvent être directement et facilement exploitables à des fins de décision. Ces premiers résultats ont été obtenus dans le contexte des bases BCT.

Organisation du mémoire

Le chapitre 1 présente les notions de base et technologies utilisées dans ce manuscrit. Nous commençons par introduire les langages de définition et de modélisation des BCT, puis leur langage d'interrogation SPARQL. Ensuite, nous discutons l'origine de l'incertitude des données et comment les BCT ont été étendues pour prendre en compte cette incertitude sous la forme d'un degré de confiance associé à chaque fait de la BC, constituant ainsi les BCI. L'extension du langage d'interrogation SPARQL aux triplets incertains est présentée par la suite. Enfin, nous donnons un panorama sur les BCI et BCT existantes dans la littérature et sur leurs systèmes de gestion de données.

Le chapitre 2 commence par un passage en revue des travaux sur les réponses coopératives concernant différentes requêtes problématiques de point de vue utilisateur, parmi lesquels nous retrouvons les deux problèmes traités dans cette thèse, à savoir les problèmes PRV et PRP. Ensuite, dans le but de positionner nos travaux par rapport à la littérature, l'état de l'art est présenté en deux parties : la première présente les approches d'interrogation coopératives réalisées dans le cadre du problème PRV tandis que la seconde présente les approches d'interrogation coopératives proposées dans le cadre du problème PRP.

Dans le chapitre 3, nous présentons notre première contribution. Ainsi, nous nous intéressons à la généralisation des notions de MFS et de XSS dans le contexte des BCI pour le problème PRV. 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 définissons une condition nécessaire pour que l'approche de calcul des MFS et des XSS proposée par Fokou et al. [FJHB17] puisse être directement utilisée pour calculer les αMFS et αXSS d'une requête qui ne retourne pas réponse. Une série d'expérimentations sur le banc d'essai WatDiv [AHÖD14], avec le QuadStore Jena TDB, a été réalisée.

Dans le chapitre 4, nous nous intéressons également au problème PRV. Dans l'approche proposée au chapitre 3, l'utilisateur doit fournir un seul seuil α. Cependant, comme il peut ne pas avoir une idée du niveau d'incertitude présent dans la BCI cible, nous souhaitons lui suggérer des requêtes relaxées avec de multiple seuils α. Ce type de relaxation nécessite le calcul des αMFS et des αXSS pour un ensemble de seuils. Pour optimiser le temps de calcul, certaines propriétés entre les αMFS et les αXSS correspondant à différents seuils sont établies et exploitées. Pour calculer ces αMFS et αXSS pour plusieurs degrés α, nous proposons trois algorithmes. Les expérimentations menées sur le banc d'essai WatDiv montrent l'intérêt de ces approches en comparaison avec une approche naïve.

Dans le chapitre 5, nous considérons le problème dual de celui abordé dans les chapitres précédents, à savoir le problème PRP. Pour autant que nous sachions, il n'existe pas de travaux sur l'identification des causes d'échec d'une requête RDF pour ce problème. Ainsi, nous avons choisi dans un premier temps de considérer ce problème dans le contexte des BCT. Notre contribution dans ce chapitre consiste à définir des contre-parties des MFS et des XSS pour le problème PRP, puis à proposer des approches et des algorithmes pour l'identification de ces contre-parties des MFS et des XSS d'une requête RDF retournant plus de K réponses.

Les travaux présentés dans les chapitres 3 et 4 ont fait l'objet de plusieurs publications [DJH$^+$17, DJH$^+$18a, DJH$^+$18b, DJH$^+$19]. La contribution du chapitre 5 n'a, à la date de rédaction de ce mémoire, pas été publiée.

Chapitre 1

Modélisation et interrogation des bases de connaissances incertaines

Sommaire

1.1 Introduction

Le Web est une ressource en constante évolution incluant la production de grandes masses d'informations. Le partage, l'échange et la réutilisation d'une manière efficace de ces masses d'informations entre les utilisateurs sont devenus primordiaux. Pour que ces informations soient accessibles à tous : services, humains et machines d'une façon automatique et permanente, Tim Berners Lee a proposé une extension du Web qualifiée de sémantique qui définit un modèle de représentation et d'échange des informations.

Le travail réalisé dans cette thèse s'inscrit dans le cadre d'une exploitation intelligente des données du Web sémantique, où nous traitons deux problèmes issus de l'interrogation de ces connaissances : (1) le problème des requêtes RDF incertaines retournant zéro résultat (réponses vides), et (2) le problème des requêtes RDF retournant trop de résultats (réponses pléthoriques). Avant de présenter ces deux problèmes dans les prochains chapitres, nous présentons dans ce premier chapitre les notions de base utilisées pour la modélisation, l'interrogation et le stockage des données du Web sémantique.

Dans la première partie de ce chapitre (section 1.2), nous introduisons les notions utilisées dans nos approches proposées, et nous décrivons trois langages standardisés et recommandés par le W3C pour la définition et la modélisation des données du Web sémantique (section 1.3) :

— le langage Resource Description Framework (RDF) [Kly04], qui permet de définir les données du Web sémantique sous la forme de triplets (sujet, prédicat, objet) ;

— RDF Schema (RDFS) [Bri07] et Web Ontology Language (OWL [1]), qui sont deux langages utilisés pour la définition d'un schéma pour les données RDF. Ainsi, ces deux langages permettent de définir des classes, propriétés, etc. pour caractériser les données RDF.

Dans la section 1.4, nous nous intéressons à l'interrogation des données RDF où nous décrivons le langage de requêtes SPARQL [SH13]. Les données RDF peuvent être *incertaines*, c'est-à-dire potentiellement fausses et incohérentes. 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 [Har09, TPR13]. Ainsi un degré de confiance explicite est associé aux données RDF et aux résultats des requêtes SPARQL. Dans la section 1.5, nous décrivons les sources de cette incertitude et nous montrons comment les langages RDF et SPARQL ont été étendus afin de prendre en compte cette caractéristique. Dans la section 1.6 nous donnons un aperçu des bases de données RDF existantes. Enfin, nous introduisons les systèmes de gestion et de stockage des données RDF, appelés *TripleStores*.

1.2 Notions de base

Avant de pouvoir présenter les principales notions liées à la gestion et à l'exploitation de connaissances, il est nécessaire de définir le sens du mot *connaissance*. Nous précisons ainsi cette notion en faisant le parallèle avec les termes *donnée* et *information*.

Donnée, information et connaissance

Les trois notions *donnée*, *information* et *connaissance* [Thi99] sont généralement représentées par une pyramide (voir la figure 1.1). Ces trois notions se définissent comme suit :

— les données constituent le point le plus bas de la pyramide et sont une collection non structurée de faits et de chiffres ;

1. https://www.w3.org/TR/owl-guide/

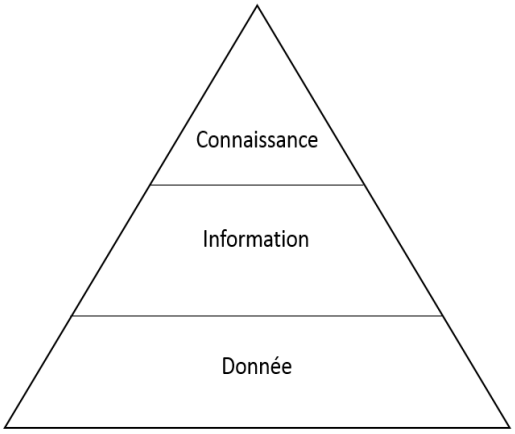


FIGURE 1.1 – Articulation des notions de *donnée*, *information* et *connaissance*

— l'information est le niveau suivant et elle est considérée comme une donnée structurée dans un contexte donné ;

— la connaissance au plus haut niveau est définie comme de l'information s'appuyant sur un référentiel collectif et apte à être partagée.

Données

Nous nous basons sur la définition des données énoncée par Thierauf [Thi99]: « faits et chiffres non structurés qui ont le moins d'impact sur la gestion ». Ainsi, nous considérons les données comme une description élémentaire en rapport avec une mesure ou un fait spécifique, mais qui ne sont aucunement organisés et qui ne fournissent aucune autre information concernant le modèles, le contexte, etc.

Informations

Pour que les données deviennent des informations, elles doivent être contextualisées, catégorisées, calculées et condensées [Row07]. L'information se différencie de la données dans le sens où elle est *utile*, et que « l'information est déduite des données » [RH17]. On trouve essentiellement des informations « dans les réponses à des questions commençant par des mots tels que qui, quoi, où, quand et combien » [Ack99].

Les technologies de l'information ont généralement une valeur inestimable pour transformer les données en informations, en particulier dans les grandes entreprises qui génèrent de grandes quantités de données sur plusieurs départements et fonctions. Le cerveau humain est donc principalement nécessaire pour aider à la contextualisation.

Connaissances

La connaissance est étroitement liée à la pratique et implique le savoir-faire et la compréhension. Les connaissances que possède chaque individu sont le produit de son expérience et englobent les normes selon lesquelles il évalue les nouvelles entrées de son environnement [DP$^+$98]. Nous nous basons sur la définition présentée par Gamble et Blackwell [GB01], qui s'appuie étroitement sur une définition antérieure de Davenport et Prusak [DP$^+$98] : les connaissances sont un mélange d'expériences encadrées, de valeurs, d'informations contextuelles, de connaissances

d'experts et d'intuition qui sont un environnement et un cadre d'évaluation et d'incorporation de nouvelles expériences et informations. Elles sont créées et appliquées par l'esprit des spécialistes. Elles sont souvent intégrées dans les documents ou les référentiels, mais également dans les routines, les pratiques et les normes organisationnelles.

Maintenant que nous avons défini les limites entre les connaissances, les informations et les données, nous donnons différentes définitions concernant les bases de connaissances.

1.3 Langages de modélisation des bases de connaissances

Dans cette section, nous commençons par présenter le langage RDF qui est à la base de langages plus expressifs comme RDF Schéma et OWL que nous décrivons succinctement.

1.3.1 RDF

RDF [2] est le langage de base du Web sémantique. C'est un langage qui a été standardisé et recommandé par le W3C en 1999. Une deuxième version a été publiée en 2004 [Kly04], qui a été suivie par la dernière version RDF 1.1 apparue en 2014 [3]. Le langage RDF est destiné à la description de ressources Web qui peuvent représenter toute sorte de choses, comme par exemple une personne, un objet, une page Web ou un lieu. Chaque ressource RDF est unique et décrite à travers des *triplets RDF*. L'ensemble des triplets RDF constitue un *graphe RDF* que l'on appelle *base de connaissances* (BC).

1.3.1.1 Triplet RDF

Un triplet RDF est une association (sujet, prédicat, objet) qui permet d'exprimer une connaissance concernant une ressource Web. Les trois attributs (sujet, prédicat, objet) d'un triplet RDF sont définis comme suit :

— Un *sujet* représente une ressource Web qui est définie d'une manière unique à l'aide d'une URI (Unique Resource Identifier) [BLFM+98]. Cette URI permet le renseignement et l'accès aux connaissances concernant la ressource. Pour illustrer cette notion, nous prenons un exemple issu de la base de connaissances de Waterloo SPARQL Diversity Test Suite (WatDiv) [AHÖD14], où différentes catégories de produits (livre, film, concert de musique classique, etc.) sont définies. Un livre "Book1" est une ressource qui a comme URI *http://db.uwaterloo.ca/galuc/wsdbm/Book1*. Une URI est préfixée d'un espace de nom *http://db.uwaterloo.ca/galuc/wsdbm/*, abrégé en *wdb:*, et est suivie d'un identifiant de ressource *Book1*. Pour simplifier l'écriture, nous ignorons l'espace de nom dans le reste du manuscrit. Ainsi la ressource *wdb:Book1* sera représentée par *Book1*.

— Un *prédicat* est une relation entre un sujet et un objet servant à caractériser une ressource. *Author* est par exemple l'un des prédicats de la ressource *Book1* présent dans la BC Watdiv.

— Un *objet* est une valeur d'un prédicat pour le sujet donné. Il peut être une ressource référencée par une URI comme par exemple *http://db.uwaterloo.ca/galuc/wsdbm/VictorHugo* qui est la valeur du prédicat *Author* de la ressource *Book1*. Il peut aussi être une valeur primitive (réel, entier, chaîne de caractères, etc.) que l'on appelle *littéral*. Il est également possible d'utiliser des nœuds blancs (ou anonymes) pour indiquer qu'une ressource possède une valeur de prédicat sans devoir en préciser la valeur. Par exemple, on peut employer un

2. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/
3. http://www.w3.org/TR/rdf11-concepts/

nœud blanc pour indiquer que *Book1* possède un nombre de pages donné sans spécifier le nombre de pages.

La figure 1.2 donne une représentation d'un triplet RDF décrivant la ressource *Book1* par la déclaration suivante « *Book1* est écrit par (*Author*) *VictorHugo* ». Ce triplet est représenté sous la forme d'un graphe, où le sujet et l'objet sont des nœuds, et le prédicat est un arc.

FIGURE 1.2 – Représentation d'un triplet RDF

1.3.1.2 Graphe RDF

Un graphe RDF est formé d'un ensemble de triplets RDF. L'exemple de la figure 1.3 décrit la ressource *Book1* avec les déclarations « il existe une ressource *Book1* de type *Book* » *(Book1 type Book)*, « la ressource *Book1* est éditée par *ACM* » *(Book1 editor ACM)*, et « *Book1* est écrit par *VictorHugo* » *(Book1 author VictorHugo)*. Ainsi, le graphe RDF de la figure 1.3 est composé de trois triplets.

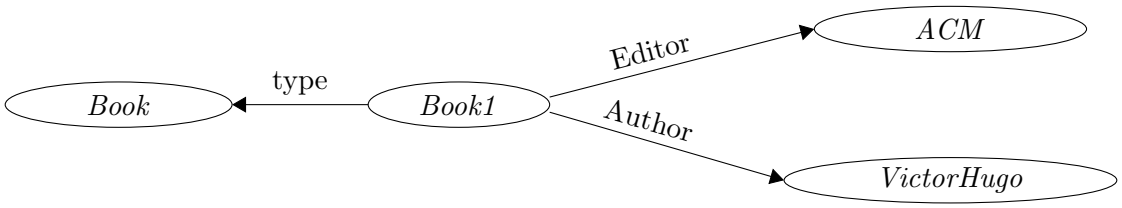


FIGURE 1.3 – Graphe de données RDF

Dans la section suivante, nous introduisons le langage RDF Schéma (RDFS) [Bri07] qui permet de définir un schéma pour des données RDF.

1.3.2 RDFS

Dans la section précédente, nous avons introduit le langage RDF et montré comment RDF définit les ressource Web sous la forme de triplets. Ce langage s'intéresse peu au schéma des données qu'il permet de définir. En général il aide à typer les ressources avec *rdf:type* comme par exemple le triplet *(Book1 rdf:type Book)* qui permet de spécifier que *Book1* est une instance de *Book*. Dans cette section nous présentons RDFS qui est une extension de RDF avec un vocabulaire qui définit le schéma des données RDF. Ainsi, RDFS définit un ensemble de mots réservés (voir le tableau 1.1) pour lesquels la sémantique et le vocabulaire sont prédéfinis (vocabulaire RDFS). Parmi le vocabulaire RDFS nous identifions les éléments suivants :

— *rdfs:class* permet de spécifier qu'une ressource RDF est une classe. Par exemple le triplet *(Book rdf:type rdfs:class)* signifie que *Book* est une classe.

— *rdfs:subClassOf* permet de définir une relation d'héritage entre deux classes. Par exemple le triplet *(Book rdfs:subClassOf Document)* spécifie qu'entre les deux classes *Book* et *Document*, il existe une relation d'héritage. Ainsi la classe *Book* est une sous-classe de la classe *Document*. Une classe peut hériter de plusieurs classes (héritage multiple).

— *rdfs:domain* permet la définition du domaine d'une propriété *(rdf:property)*. Ainsi, *rdfs:domain* permet de spécifier l'ensemble des ressources RDF qui peuvent être le sujet d'une propriété

Vocabulaire RDFS	Triplet
Classe	(s, rdfs:class, o)
Sous classe	(s, rdfs:subClassOf, o)
Domaine	(s, rdfs:domain, o)
Co-domaine	(s, rdfs:range, o)

TABLEAU 1.1 – Vocabulaire RDFS

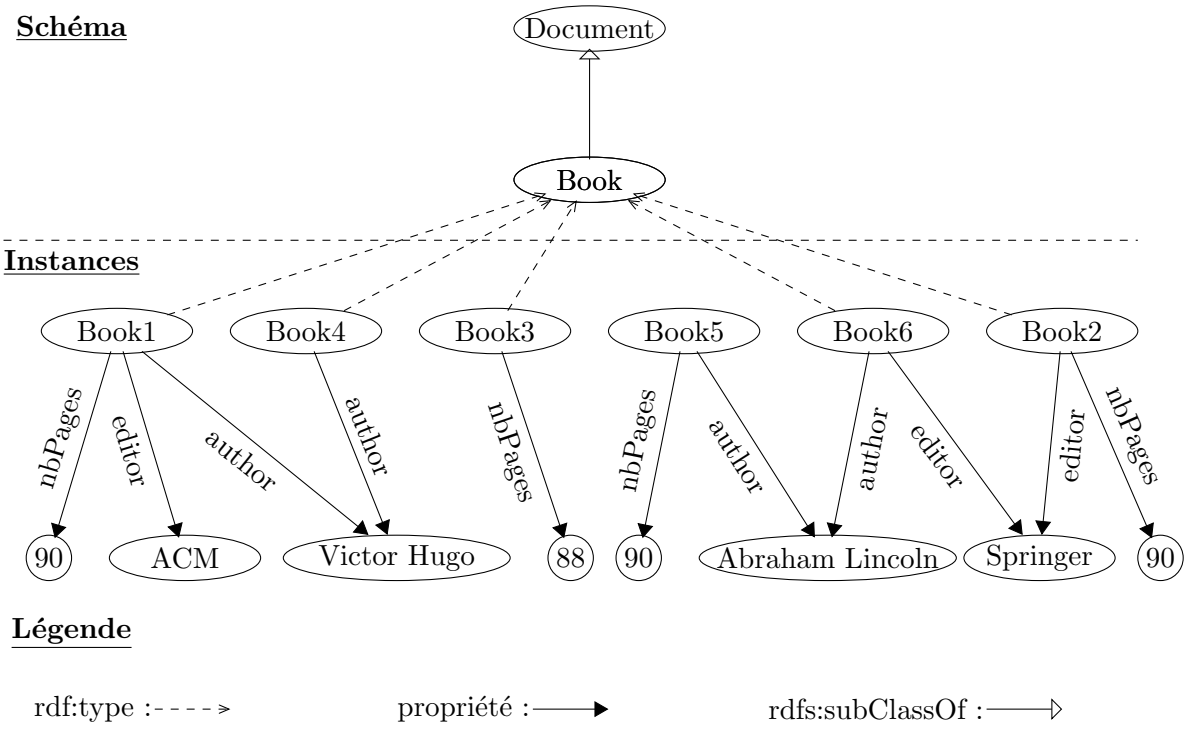


FIGURE 1.4 – Graphe RDF des données et du schéma associé

donnée. Par exemple le triplet *(Author rdfs:domain Book)* spécifie que seules les ressources de type *Book* peuvent être définies avec la propriété *Author*.

— *rdfs:range* permet de définir le co-domaine d'une propriété RDF. *rdfs:range* spécifie le type d'une valeur qu'une propriété donnée peut avoir. Par exemple le triplet *(Author rdfs:range Person)* signifie que l'auteur d'un livre est une personne.

Dans la figure 1.4 nous présentons des données RDF concernant six livres (instances) avec leur schéma représenté à l'aide de deux propriétés introduites précédemment. La première propriété liée au schéma est la propriété RDF rdf:type, qui signifie que cinq livres (Book1, Book2, Book3, Book4 et Book6) sont de type Book (sur cet exemple, Book5 est une ressource qui n'a pas de type). La deuxième propriété appartient à RDFS, à savoir la propriété rdfs:subClassOf signifiant que la classe Book hérite de la classe Document.

Un graphe RDF peut aussi être présenté dans une table de 3 colonnes, à savoir sujet, prédicat et objet. Le tableau 1.2 montre cette représentation pour l'exemple de la figure 1.4 que nous utiliserons dans la suite de ce manuscrit.

BC		
sujet	**prédicat**	**objet**
Book	subClassOf	Document
Book1	type	Book
Book1	author	Victor Hugo
Book1	editor	ACM
Book1	nbPages	90
Book2	type	Book
Book2	editor	Springer
Book2	nbPages	90
Book3	type	Book
Book3	nbPages	88
Book4	type	Book
Book4	author	Victor Hugo
Book5	author	Abraham Lincoln
Book5	nbPages	90
Book6	type	Book
Book6	author	Abraham Lincoln
Book6	editor	Springer

Tableau 1.2 – Une base de connaissances (BC)

	Règles ⇒ Déduction
rdf_2	(s, p, o), (p, rdfs:domain, u) ⇒ (s, rdf:type, u)
rdf_3	(s, p, o), (p, rdfs:range, u) ⇒ (o, rdf:type, u)
rdf_5	(p1, rdfs:subPropertyOf, p2), (p2, rdfs:subPropertyOf, p3) ⇒ (p1, rdfs:subPropertyOf, p3)
rdf_7	(s, p1, o), (p1, rdfs:subPropertyOf, p2) ⇒ (s, p2, o)
rdf_9	(s, rdf:type, c1), (c1, rdfs:subClassOf, c2) ⇒ (s, rdf:type, c2)
rdf_11	(c1, rdfs:subClassOf, c2), (c2, rdfs:subClassOf, c3) ⇒ (c1, rdfs:subClassOf, c3)

Tableau 1.3 – Exemples de règles d'inférences RDFS [HPS14]

1.3.3 Raisonnement

Le langage RDFS offre la possibilité de raisonner sur des données RDF. Cette technique de raisonnement repose sur la déduction de nouveaux triplets RDF en se basant sur ceux qui existent déjà. Afin de déduire de nouveaux triplets RDF, il est nécessaire d'avoir des règles logiques appelées règles d'inférence qui s'appliquent sur les données RDF présentes. Si une règle logique est vérifiée pour des triplets RDF, alors un nouveau triplet est déduit et ajouté à la BC. Ainsi, la déduction de données RDF, basée sur des règles d'inférence, implique la complétion de la BC par de nouveaux triplets déduits à partir des triplets existants. Nous présentons quelques règles d'inférences dans le tableau 1.3. La liste complète des règles d'inférences est donnée par Hayes et al. [HPS14].

Exemple

Nous présentons un exemple de déduction basée sur la règle d'inférence rdf_9 du tableau 1.3 : (s, rdf:type, c1), (c1, rdfs:subClassOf, c2) ⇒ (s, rdf:type, c2). Pour comprendre cette

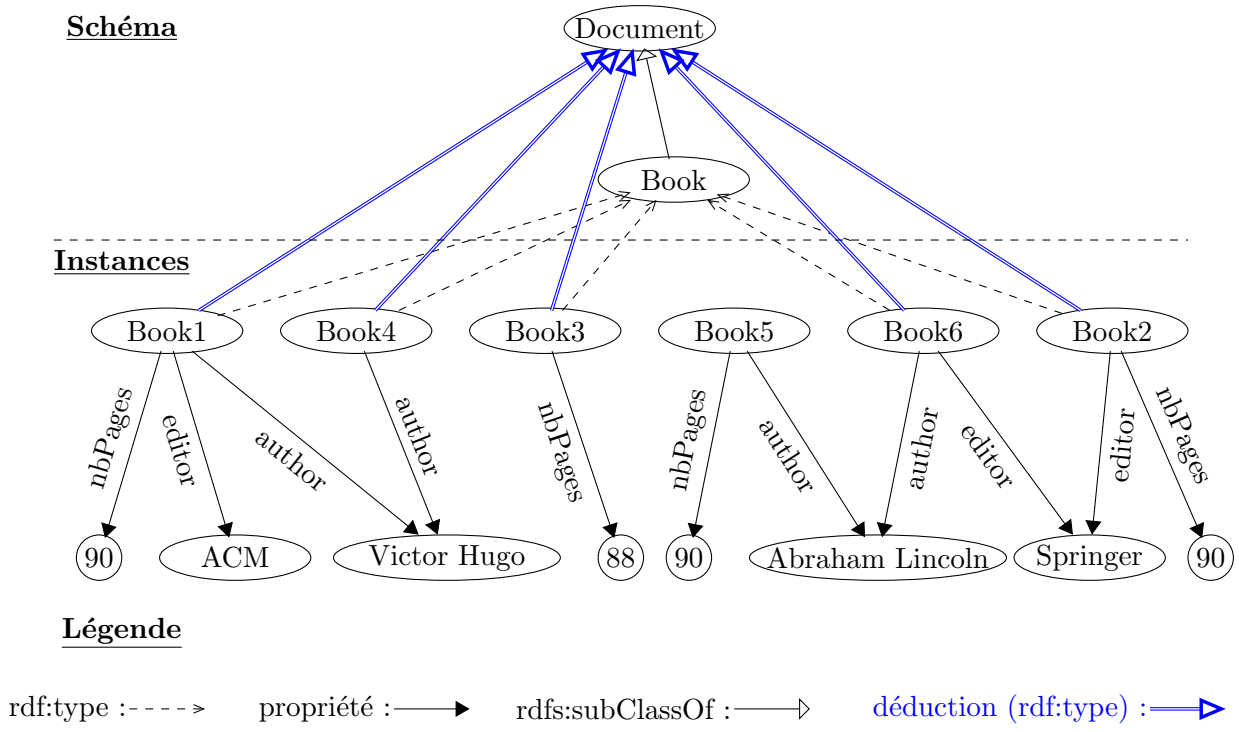


FIGURE 1.5 – La déduction des données RDF

règle d'inférence, nous l'appliquons directement sur notre graphe RDF (voir la figure 1.4). Si des ressources sont de type Book (Book1, Book2, Book3, Book4, Book6), et si Book hérite (subClasseOf) de Document, alors, les ressources (Book1, Book2, Book3, Book4, Book6) sont également de type Document. Suite à cette déduction, de nouveaux triplets déduits sont ajoutés à la BC présentée dans la figure 1.5.

Nous donnons un exemple concret pour comprendre l'utilité de la déduction. Supposons qu'avant d'appliquer la déduction sur notre graphe RDF présente dans la figure 1.4, un utilisateur lance une recherche sur ce graphe pour rechercher tous les Documents présents. Le système va retourner à l'utilisateur un ensemble vide de réponses, car il n'exploite pas le fait qu'un Livre soit un Document. Si la BC avait appliqué les règles d'inférence, le système aurait retourné, pour la même recherche, l'ensemble des cinq réponses : Book1, Book2, Book3, Book4, Book6. Ainsi, toutes les ressources appartenant à la classe Book peuvent également être considérées comme des ressources qui appartiennent à la classe Document. Les travaux réalisés lors de cette thèse peuvent s'appliquer sur des requêtes qui prennent ou ne prennent pas en compte le raisonnement.

Avec RDF Schema, il est uniquement possible de définir les relations concernant la hiérarchie des classes, des propriétés, ou de définir le domaine et le co-domaine de ces propriétés. Pour des domaines d'application pour lesquels RDF schéma n'est pas suffisant pour définir un modèle complet, un langage, appelé OWL [AVH04], a été proposé.

1.3.4 OWL

Le langage OWL a été normalisé par le World Wide Web Consortium (W3C) en 2004 [4]. OWL est une fusion de deux langages non normalisés, DAML [HM00] et OIL [FVHH+01]. OWL est un langage de schéma pour les données RDF. Nous présentons par la suite le vocabulaire OWL.

4. http://www.w3.org/TR/owl-features/

Vocabulaire OWL concernant les classes

— *owl:disjointWith :* cette propriété s'applique à deux classes A et B pour préciser que les classes A et B sont disjointes, c'est-à-dire que si une ressource est membre de la classe A, elle ne peut pas être membre de la classe B. Par exemple, un *un maître de conférence* ne peut pas en même temps être un *professeur des universités*.

— *owl:intersectionOf :* cette propriété implique la création d'une nouvelle classe C, à partir de deux classes A et B. La nouvelle classe C contient les éléments de la classe A et de la classe B. Les membres de la nouvelle classe créée sont ceux appartenant à la fois à la classes A et à la classe B. Par exemple le *Vin Rouge* est l'intersection de la classe *Vin* et de la classe *Couleur Rouge*. Cela signifie que si une instance de la classe *Rouge* est également une instance de la classe *Vin*, alors c'est une instance de la classe *Vin Rouge*

— *owl:equivalentClass :* cette propriété signifie que deux classes sont équivalentes. Par exemple, entre les deux classes *Voiture* et *Automobile*, nous pouvons créer une relation d'équivalence.

Vocabulaire concernant les prédicats

Ce type de vocabulaire concerne les prédicats RDF.

— *owl:minCardinality* définit l'ensemble des ressources qui ont au moins N valeurs distinctes pour un prédicat donné. Par exemple, un cours doit être enseigné par au moins un enseignant. La déclaration formelle de cet exemple de cardinalité minimale se présente comme suit avec la syntaxe Turtle [5].

```
:Cours owl:subclassOf [
    rdf:type            owl:Restriction;
    owl:minCardinality  "1"^^xsd:nonNegativeInteger;
    owl:onProperty      :enseignePar
] .
```

— *owl:maxCardinality* définit l'ensemble des ressources qui ont au plus N valeurs distinctes pour un prédicat donné. Par exemple, nous pouvons spécifier que la classe département compte au maximum 30 membres.

```
:Departement owl:subclassOf [
    rdf:type            owl:Restriction;
    owl:maxCardinality  "30"^^xsd:nonNegativeInteger;
    owl:onProperty      :membre
] .
```

— *owl:equivalentProperty* signifie que deux prédicats sont équivalents. Par exemple, entre les deux prédicats *étudiantEn* et *suitLeParcours*, nous pouvons créer une relation d'équivalence.

```
:etudiantEn  owl:equivalentProperty  otherOnt:suitLeParcours .
```

Dans nos travaux nous utilisons essentiellement le langage de schéma RDFS, même si nos propositions peuvent s'appliquer à des données RDF dont le schéma serait modélisé avec OWL. La prochaine section présente le langage standard permettant l'interrogation des données RDF.

1.4 Interrogation des bases de connaissances avec SPARQL

SPARQL [SH13] est un langage d'interrogation des données RDF. En janvier 2008, le langage SPARQL est devenu un standard et une recommandation du World Wide Web Consortium (W3C).

5. https://www.w3.org/TR/turtle/

1.4.1 Syntaxe de SPARQL

La figure 1.6 représente une décomposition simple d'une requête SPARQL, qui est faite sur deux blocs. Le premier bloc est l'en-tête de la requête, qui précise le type d'interrogation SELECT, ASK, DESCRIBE ou CONSTRUCT suivi par des variables [6] qui permettent d'associer des noms aux résultats après exécution de la requête.

— *SELECT* permet de retourner les résultats satisfaisants les contraintes de la requête.

— *ASK* est utilisé pour renvoyer un résultat booléen, VRAI s'il existe au moins un résultat qui correspond au modèle de la requête, FAUX sinon.

— *DESCRIBE* est utilisé pour renvoyer un seul graphe RDF avec des informations sur les ressources sélectionnées. Le graphe résultat est calculé par le moteur de requête SPARQL, et l'utilisateur n'a pas d'influence sur ce propos.

— *CONSTRUCT* peut produire des triplets composés de données RDF, non trouvées dans la BC interrogée. Autrement dit, ce type de requête permet de compléter les données de la BC en fonction des contraintes présentes dans ce type de requête.

Parmi les quatre types de requêtes que nous avons cités, nous nous focalisons sur les requêtes de type SELECT. En effet, 94% des requêtes exécutées par DBPedia [LIJ+15], Semantic Web Dog Food [MHHD07] et Linked Geo Data [SLHA12] sont de type SELECT. Le deuxième bloc d'une requête SPARQL, appelé Patron de Graphe, commence par la clause WHERE et contient l'ensemble des contraintes de sélection sous forme de triplets RDF (patron de triplet) contenant des variables permettant la sélection des données.

1.4.1.1 Patron de triplet

Un patron de triplet est un triplet RDF qui représente une contrainte de sélection pour une requête RDF. Dans un patron de triplet, nous pouvons remplacer chacun des éléments du triplet (sujet, prédicat ou objet) par une variable. Prenant par exemple le patron de triplet *(?b type Book)* que nous appliquons sur le graphe RDF de la figure 1.4. Ce dernier consiste à trouver toutes les ressources qui sont de type *Book*. Cet exemple retourne comme résultat *Book1, Book2, Book3, Book4* et *Book6*. Une requête peut contenir plusieurs contraintes de sélection sous forme d'une conjonction de triplets RDF. Cet ensemble de contraintes est appelé un patron de graphe.

1.4.1.2 Patron de graphe

Comme décrit dans la section précédente, un patron de graphe est un ensemble de patrons de triplets. Deux patrons de triplets sont joints lorsqu'ils ont au moins une variable commune. L'exemple de la figure 1.6 montre un patron de graphe dans lequel il existe une variable de jointure ?b entre ses deux patrons de triplets. Ce patron de graphe demande la recherche de toutes les ressources qui sont de type *Book* avec leurs auteurs (Author). L'annotation associée aux différents patrons de triplets t_1, t_2, nous servira par la suite pour référencer plus facilement les parties d'une requête RDF. Par exemple, la requête de la figure 1.6 peut être représentée comme la conjonction de ses deux patrons de triplet : $t_1 \wedge t_2$.

Il existe aussi une représentation des requêtes SPARQL sous forme de graphe. La figure 1.7 nous montre cette représentation, qui adopte le principe suivi pour la représentation d'un graphe RDF. Ainsi les triplets présents dans la clause WHERE se transforment en nœuds et arcs.

6. Une variable est une chaîne de caractères qui commence par le caractère '?', comme par exemple ?e.

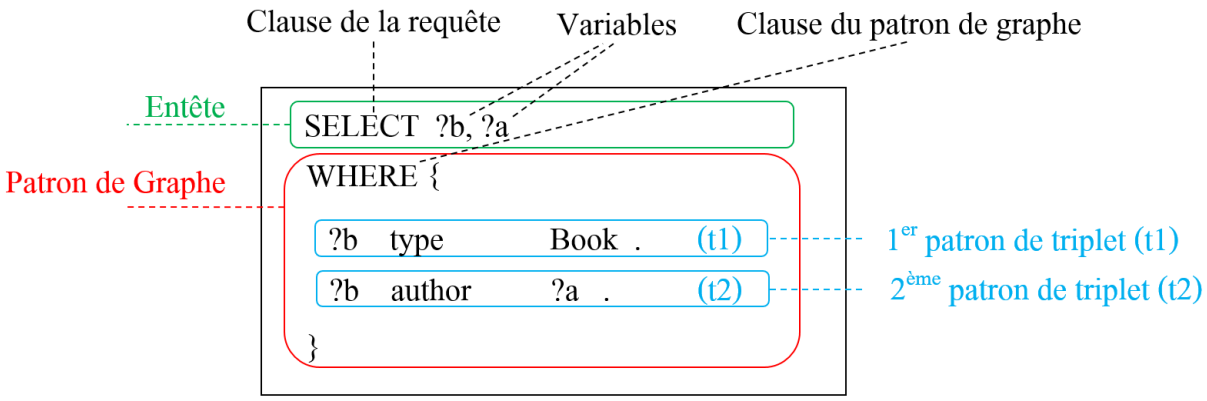


FIGURE 1.6 – Exemple de requête SPARQL

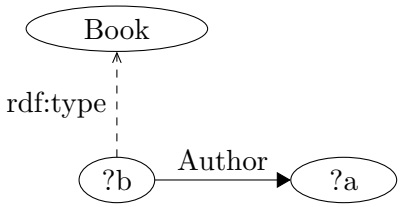


FIGURE 1.7 – Patron de graphe SPARQL

1.4.2 Évaluation d'une requête SPARQL

Avant d'illustrer l'évaluation d'une requête SPARQL (requête RDF), nous donnons tout d'abord une définition formelle des données RDF.

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}).

Evalutation des requêtes RDF

Pour définir comment une requête RDF est évaluée, nous avons besoin de la notion de *mapping* introduite par Pérez et al. [PAG09]. Un *mapping* est une fonction partielle $\mu : V \rightarrow T$. Pour un patron de triplet 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 : $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ sont des mappings compatibles}\}$. 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 $Q = t_1 \wedge \cdots \wedge t_n$ 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'implications (*entailment regimes*), tels que définis dans la spécification SPARQL (par exemple, le régime d'implication simple ou RDF). Dans ce manuscrit, les exemples et les expérimentations sont basés sur le régime d'implication simple, même si n'importe quel autre régime d'implication

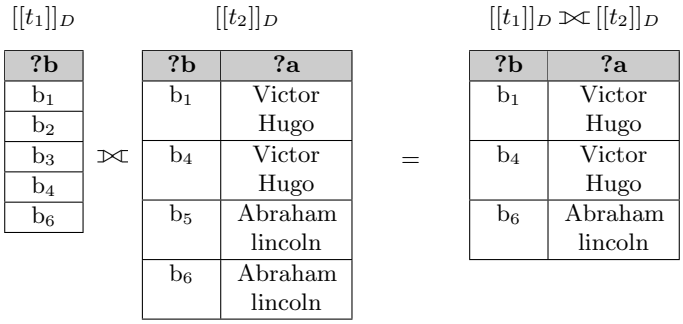


FIGURE 1.8 – Évaluation de la requête Q

pourrait être utilisé. Le nombre de patrons de triplets d'une requête Q est noté $|Q|$ et les variables de cette dernière sont notées $var(Q) = \bigcup var(t_i)$.

Afin d'illustrer l'évaluation d'une requête SPARQL, considérons l'évaluation de la requête de la figure 1.6. La figure 1.8 illustre l'évaluation de chaque patron de triplet $[[t_1]]_D$ et $[[t_2]]_D$, ainsi que leur jointure $[[Q]]_D = [[t_1]]_D \bowtie [[t_2]]_D$.

Dans la première partie de ce chapitre, nous avons vu les principales définitions des données RDF, les définitions des langages de schéma RDF et du langage d'interrogation des données RDF. Nous allons par la suite présenter la prise en compte de l'incertitude dans les bases de données RDF.

1.5 Prise en compte de la confiance

Avec la vulgarisation du Web, l'accès à la connaissance est de moins en moins confiné. Pourtant la fiabilité de ces connaissances n'est pas toujours garantie. La conséquence de cette non-fiabilité est l'apparition très récente des BC incertaines dont la première est YAGO [SKW07], proposée par l'institut Max Planck de l'université de Saarland en 2007. Nous pouvons également citer Google's Knowledge vault [DGH+14b], ou la BC Nell [CBK+10] réalisée par l'université de São Carlos. Dans ces BC incertaines, chaque instance ou chaque triplet RDF est associé à un degré de confiance, qui informe l'utilisateur du taux de fiabilité de ce triplet RDF. La spécificité de ces BC incertaines par rapport aux BC classiques se résume à l'association d'un degré de confiance qui varie entre 0 et 1 à chaque triplet RDF.

1.5.1 Sources de l'incertitude des données RDF

L'information contenue dans les documents du Web peut présenter différentes imperfections [Ker15]. Elle peut être, par exemple, incomplète, incertaine ou encore ambiguë. Ceci peut remettre en question la nature de l'information véhiculée. Il devient donc nécessaire de qualifier et éventuellement quantifier ces imperfections, afin de présenter à l'utilisateur une exploitation de connaissances de bonne qualité. Dans le cadre de cette thèse, nous nous intéressons à l'aspect incertain des données RDF. Nous allons donc voir par la suite comment les données RDF dans ce nouveau contexte ont été étendues avec la prise en compte de la confiance.

1.5.2 Extension de RDF avec la confiance

Pour évaluer la fiabilité des données RDF, Hartig dans [Har09] a proposé une modélisation uniforme des données RDF en prenant en compte la confiance. Plus concrètement, Hartig a proposé un nouveau modèle associant chaque triplet RDF à une valeur de confiance (modèle de confiance) qui varie entre 0 et 1 ou une valeur de non-confiance variant entre 0 et -1. Le modèle Trust-RDF ou T-RDF, exprime la fiabilité ou le degré de confiance associé au triplet RDF en tant que sujet de croyance et d'incrédulité d'informations représentées par ce triplet. En fait, la croyance (incrédulité) en un triplet RDF est exprimée par un degré de confiance de l'information. En effet, la mesure de confiance, notée t, est une valeur comprise dans l'intervalle [-1,1] ou inconnue. Dans le cas où t vaut 1, l'utilisateur est absolument certain de la fiabilité du triplet. Une valeur de confiance positive inférieure à 1 représente une conviction plus ou moins forte dans la véracité de l'information. À l'opposé, une valeur négative exprime une incrédulité, et une valeur t égale à -1 exprime une certitude dans le mensonge informationnel. Au cours de cette dernière décennie, des efforts de recherche considérables ont été déployés pour développer le modèle de confiance RDF, tels que [Har09, TPR13, AHÖD14, CNFM14, FG15].

Dans nos travaux, nous ne considérons que la partie liée à la croyance, positive. Ce choix a été fait suite à l'analyse des BC incertaines qui existent sur le Web. Ces BC ne considèrent pas la non-confiance. Ainsi tous triplets RDF des BC incertaines qui existent sur le Web sont associés à un degré de confiance qui varie entre 0 et 1. Nous allons par la suite présenter l'intégration de cette croyance aux triplets et aux graphes RDF.

1.5.2.1 Triplet RDF avec confiance

Dans le contexte de l'incertain, un triplet RDF est enrichi par une valeur réelle entre 0 et 1. Cette valeur représente le degré de confiance qu'un expert peut accorder à un triplet RDF. Dans l'exemple de la figure 1.9, la valeur 0.5 ajoutée au triplet RDF signifie que la connaissance que Book1 est écrit par (Author) VictorHugo, est d'une confiance moyenne.

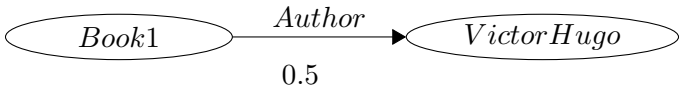


FIGURE 1.9 – Triplet RDF avec un degré de confiance

1.5.2.2 Graphe RDF avec confiance

Comme pour un graphe RDF, un graphe RDF incertain est formé d'un ensemble de triplets RDF incertains. De manière analogue à un triplet RDF (voir figure 1.9), les degrés de confiance dans un graphe RDF sont des poids sur les arêtes reliant les nœuds.

Un graphe RDF incertain (BC incertaine) peut se représenter sous forme d'une table de 4 colonnes, à savoir, sujet, prédicat, objet et confiance. La table 1.4 illustre cette représentation, que nous utilisons dans le reste de ce manuscrit. Dans la section suivante, nous introduisons le langage d'interrogation des BC incertaines.

1.5.3 Interrogation des bases de connaissances incertaines

Pour interroger les BC incertaines, une extension du langage SPARQL a été proposée.

BC incertaine			
sujet	**prédicat**	**objet**	**confiance**
Book	rdfs:subClassOf	Document	0.5
b_1	author	Victor Hugo	0.5
b_1	editor	ACM	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	author	Victor Hugo	0.5
b_4	type	Book	0.1
b_5	author	Abraham Lincoln	0.5
b_5	nbPages	90	0.3
b_6	author	Abraham Lincoln	0.3
b_6	editor	Springer	0.3
b_6	type	Book	0.3

TABLEAU 1.4 – Exemple de représentation tabulaire d'une BC incertaine

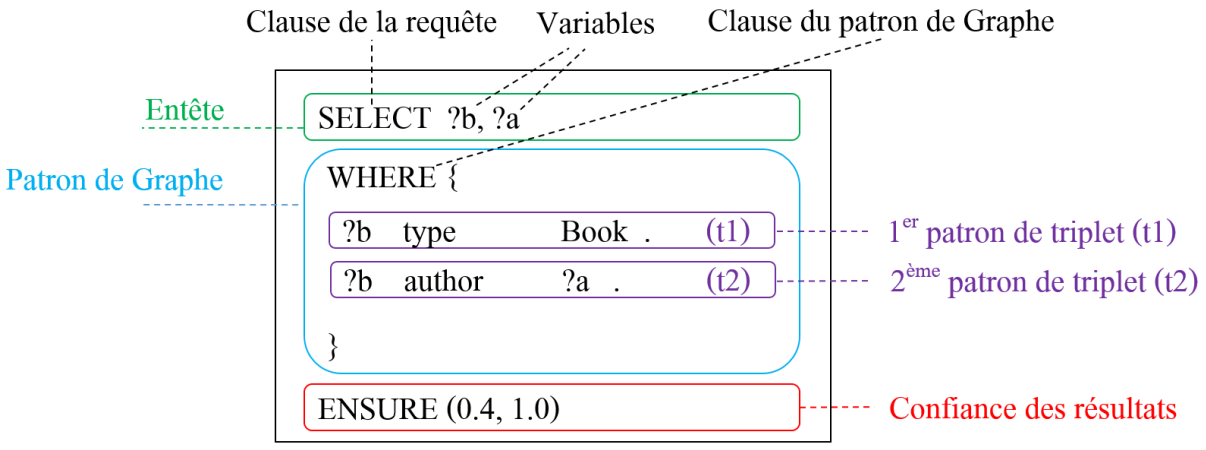


FIGURE 1.10 – Exemple de requête TSPARQL

$[[t_1]]_D$

?b	confiance
b_1	0.9
b_2	0.3
b_3	0.7
b_4	0.1
b_6	0.3

$\rtimes$

$[[t_2]]_D$

?b	?a	confiance
b_1	Victor Hugo	0.5
b_4	Victor Hugo	0.5
b_5	Abraham lincoln	0.5
b_6	Abraham lincoln	0.3

$=$

$[[t_1]]_D \rtimes [[t_2]]_D$

?b	?a	confiance
b_1	Victor Hugo	min(0.9,0.5)=0.5
b_4	Victor Hugo	min(0.1,0.5)=0.1
b_6	Abraham lincoln	min(0.3,0.3)=0.3

FIGURE 1.11 – Évaluation d'une requête RDF incertaine

1.5.3.1 Définition formelle

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 $tv : T_{RDF} \rightarrow [0, 1]$.

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 : $[[Q]]^{\alpha}_D = \{\mu \in [[Q]]_D \mid tv(\mu) \geq \alpha\}$. Dans le langage TSPARQL [7] proposé par Hartig [Har09], cette condition sur le degré de confiance est exprimé à l'aide de la clause ENSURE. La figure 1.10 présente un exemple de requête TSPARQL où la clause ENSURE(0.4 , 1.0) signifie que l'utilisateur souhaite des résultats avec un degré de confiance minimum (α) égale à 0.4.

Afin d'illustrer l'évaluation d'une requête SPARQL avec un degré de confiance α, considérons l'évaluation de la requête de la figure 1.10. La figure 1.11 donne l'évaluation de chaque patron de triplet $[[t_1]]_D$ et $[[t_2]]_D$, ainsi que leurs jointures $[[Q_1]]_D = [[t_1]]_D \wedge [[t_2]]_D$. L'agrégation des valeurs de confiance est ensuite effectuée sur le résultat de cette requête, en fonction de la valeur de confiance associée à chaque mapping. Dans cet exemple, la fonction d'agrégation choisie est le minimum.

BC	Nature	Domaine
DBpedia	classique	académique
Yago	incertaine	académique
Freebase	classique	académique
NELL	incertaine	académique
SigmaKB	incertaine	académique
Knowledge Graph de Google	classique	commerciale
Knowledge Vault de Google	incertaine	commerciale
Satori de Microsoft	classique	commerciale

TABLEAU 1.5 – Aperçu sur les BC

1.6 Bases de connaissances existantes et systèmes de gestion

Dans cette section, nous commençons par introduire quelques BC qui existent dans le Web sémantique. Le tableau 1.5 récapitule ces BC selon leur nature (classique ou incertaine) et leur domaine (académique ou commerciale). En seconde partie, nous donnons un descriptif plus détaillé de ces BC.

Le terme "Knowledge Graph" synonyme de BC, a été inventé par Google lorsqu'il a introduit son graphe de connaissances en tant que colonne vertébrale d'une nouvelle stratégie de recherche sur le Web : passer du traitement de texte pur à une représentation plus symbolique de la connaissance, en utilisant le slogan « nous traitons des objets et non pas des chaînes de caractères » [AS12].

Différentes BC publiques (classiques et incertaines) sont disponibles sur le Web, y compris DBpedia [ABK+07] et YAGO [SKW07], qui ont été créées en extrayant des informations de Wikipedia (Yago s'enrichit aussi de WordNet [8] et GeoNames [9]), Freebase [10] qui a été intégrée dans WikiData [Vra12a], NELL [CBK+10] et SigmaKB [RGW16b] qui utilise l'algorithme Consensus Maximization Fusion [RGW16a] pour valider, fusionner et extraire des connaissances en se basant sur 71 bases de connaissances incertaines comme par exemple YAGO et NELL. Il existe d'autres bases de connaissances qui sont le fruit d'une fusion de plusieurs BC, comme Wal-Mart [DLT+13b]. Cette démarche est mise en œuvre afin de résoudre le problème de données incomplètes.

Il existe des BC commerciales, telles que Knowledge Graph et Knowledge Vault de Google, [DGH+14b], Knowledge Graph de Yahoo [BCMT13], le Satori de Microsoft et Entities Graph de Facebook. Cependant, celles-ci ne sont pas accessibles au public, ne sont donc pas adaptées à la création d'applications par des parties autres que les propriétaires, et ne peuvent pas être analysées en profondeur. Souvent, la taille, la structure, le contenu et la qualité de ces BC ne sont pas rendus publique.

1.6.1 Descriptifs des bases de connaissances

Le tableau 1.6 donne un aperçu des BC que nous décrivons par la suite. Ce tableau récapitule le nombre d'instances et de faits, ainsi que le nombre de types différents et de relations définis dans leurs schémas. *Instances* dénote le nombre d'instances ou de concepts définis dans la BC, *Faits* indique le nombre d'instructions relatives à ces instances. Les BC Satori et SigmaKB ne sont pas présentées car, à notre connaissance, aucune donnée chiffrée récente détaillant ces BC n'a été rendue publique.

1.6.1.1 Dbpedia

Les données RDF de Dbpedia [BLK+09] sont extraites de données structurées dans *Wikipedia*. Les sources principales de cette extraction sont les paires clé-valeur inscrites dans les *WikipediaInfoBox*. Dans un processus de type *foule*, les types de infobox sont mappés sur le schéma DBpedia et les clés utilisées dans ces infobox sont mappées sur des propriétés de ce schéma. La version la plus récente de DBpedia remonte à 2015 (DBpedia 2015, extraite de la version anglaise de Wikipédia de février–mars 2015). Elle contient 4,8 millions d'instances et 176

7. SPARQL est une extension de SPARQL, qui prend en compte les degré de confiances
8. https://wordnet.princeton.edu/
9. http://www.geonames.org/
10. https://developers.google.com/freebase/

BC	Instances	Types	Relations	Faits
DBpedia	4 806 150	735	2 813	176 043 129
Yago	4 595 906	488 469	77	25 946 870
Freebase	49 947 845	26 507	37 781	3 041 722 635
NELL	2 006 896	285	425	432 845
Knowledge Graph de Google	570 000 000	1 500	35 000	18 000 000 000
Knowledge Vault de Google	45 000 000	1 100	4 469	271 000 000

TABLEAU 1.6 – Détails concernant les BC

millions de triplets RDF. Le schéma comprend 735 classes et 2 800 relations.

1.6.1.2 Yago

Comme DBpedia, YAGO est également extraite de Wikipedia. YAGO [SKW07] construit son schéma implicitement à partir du système de catégories de Wikipedia et des ressources WordNet [Mil95], avec des propriétés infobox mappées manuellement à un ensemble fixe d'attributs. Bien que DBpedia crée différents graphes de connaissances inter-connectés pour chaque édition linguistique de Wikipédia [BB14], YAGO vise une fusion automatique des connaissances extraites de différentes éditions à l'aide d'heuristiques. La dernière version de YAGO, c'est-à-dire YAGO3 [MBS13], contient 4,6 millions d'entités et 26 millions de faits. Le schéma comprend environ 488 000 types et 77 relations.

1.6.1.3 NELL

Never-Ending Language Learning system (NELL)[CBK$^+$10], est un système de construction d'une BC incertaine développé par une équipe de recherche à l'Université Carnegie Mellon. Des parties du système sont en cours d'exécution sur un Cluster de calcul intensif fourni par Yahoo. Le principe de Nell est de continuer l'extraction et l'apprentissage de fiabilité des triplet RDF sans jamais s'arrêter, d'où le nom "Never-Ending Language Learning system". Nell extrait des informations sans interruption depuis 2010, il a créé une BC incertaine qui contient déjà des millions d'objets du Web. L'objectif de Nell est de développer un système coopératif qui répond aux questions posées par des humains en langage naturel.

Dans sa version la plus récente, NELL contient 2 millions d'entités et 433 000 relations entre elles. Le schéma des données RDF de la BC NELL définit 285 classes et 425 relations.

1.6.1.4 Knowledge Graph de Google

Le Knowledge Graph de Google [AS12] a été présenté au public en 2012. Il n'existe que quelques sources externes qui analysent certains des mécanismes du Knowledge Graph de Google [11]. À partir de cela, on peut supposer que les principales sources Web semi-structurées, telles que Wikipedia, contribuent au graphe de connaissances, ainsi que le balisage structuré (Microdata [MPB14]) sur les pages Web et les contenus du réseau social en ligne de Google (Google+). D'après [DGH$^+$14b], le Knowledge Graph de Google contient 18 milliards d'instance concernant 570 millions d'entités, avec un schéma de 1 500 types d'entités et de 35 000 types de relations.

11. https://www.techwyse.com/blog/search-engine-optimization/seo-efforts-to-get-répertoriés-dans-google-knowledge-graph

1.6.1.5 Knowledge Vaut de Google

Dans le cadre commercial, Google a lancé en 2014 son projet Knowledge Vault [DGH+14b], une BC incertaine contenant 1.6 milliard de triplets RDF qui sont collectés automatiquement à travers l'ensemble du Web. Chaque triplet RDF est entaché d'une valeur de confiance probabiliste comprise entre 0 et 1. Cette valeur de confiance est calculée en fonction des sources qui ont fourni l'information et d'autres facteurs. En attendant que la BC Knowledge Vault remplace la BC classique Knowledge Graph, utilisée actuellement par le moteur de recherche Google. Ce dernier considère que tous les triplets RDF avec un degré de confiance inférieur à 0.2 sont des triplets incorrects, et ils sont supprimés de la BC actuelle Knowledge Graph si ces derniers y sont présents. Dans le cas où les triplets ont une valeur de confiance supérieur à 0.8, ces derniers sont ajoutés à Knowledge Graph si ils n'y sont pas encore. Ainsi, la BC knowledge Vault est utilisée pour entretenir la BC Knowledge Graph.

Dans la prochaine section nous introduisons brièvement les systèmes de gestion des données RDF.

1.6.2 Systèmes de gestion des bases de connaissances

Pour les données RDF, les systèmes de gestion sont appelés des *TripleStores*. Ces derniers permettent le stockage des données et offrent un système d'interrogation SPARQL. Les TripleStores se dérivent en deux familles :

— *TripleStore natif* qui stocke et interroge les données sous forme de graphe tels que nous l'avons vu dans la section précédente (voir figure 1.4),

— *TripleStore non natif* qui stocke et interroge les données sous format relationnel dans un SGBD classique, tout en offrant une interface RDF. Pour ces systèmes, l'utilisateur fournit en entrée des données au format RDF, transformées en données relationnelles par le système tel que présenté dans la figure 1.4. Pour leur interrogation, les requêtes SPARQL sont fournies en entrée du système et sont transformées en requêtes SQL par réécriture.

Le tableau 1.7, donne des exemples de TripleStores présentés selon la nature de leur stratégie de stockage, un classement basé sur leur popularité [12], leur type de licence, le langage avec lesquels ils sont développés et les langages d'interrogation supportés.

Grâce au classement des TripleStores, nous avons accès à plusieurs informations concernant leur utilisation. La figure 1.12 présente une capture de l'interface du site *db-engines* effectuée au mois de juillet 2019. Selon cette source, Jena TDB et Virtuoso sont les TripleStores ayant une licence open source les plus utilisés. Dans cette thèse, ces deux TripleStores servent de support aux évaluations expérimentales réalisées. Ces deux TripleStores introduisent par ailleurs une diversité dans nos expérimentations grâce à leurs natures différentes, native pour Jena-TDB et non native pour Virtuoso.

12. `https://db-engines.com/en/ranking/rdf+store`, réalisé au mois de juillet de 2019

Rank Jul 2019	Rank Jun 2019	Rank Jul 2018	DBMS	Database Model	Score Jul 2019	Score Jun 2019	Score Jul 2018
1.	1.	1.	MarkLogic	Multi-model	13.81	+0.30	+2.50
2.	2.	↑ 3.	Virtuoso	Multi-model	3.12	+0.01	+1.13
3.	3.	↓ 2.	Apache Jena - TDB	RDF	2.28	+0.00	+0.08
4.	4.	4.	Amazon Neptune	Multi-model	1.39	+0.15	+0.61
5.	5.	↑ 7.	GraphDB	Multi-model	1.11	+0.03	+0.62
6.	6.	↓ 5.	AllegroGraph	Multi-model	0.95	+0.02	+0.36
7.	7.	↓ 6.	Stardog	Multi-model	0.73	+0.01	+0.20
8.	8.	↑ 12.	Blazegraph	Multi-model	0.56	0.00	+0.37
9.	9.	↑ 10.	Redland	RDF	0.47	-0.01	+0.23
10.	10.	↓ 9.	RDF4J	RDF	0.37	0.00	+0.12
11.	11.	↓ 8.	Algebraix	RDF	0.35	+0.01	+0.09
12.	12.	↓ 11.	4store	RDF	0.33	+0.01	+0.10

FIGURE 1.12 – Classement des TripleStores effectué en juillet 2019

RDF Store	Nature	Classement	Licence	Langage	SQL	SPARQL
MarkLogic	Non-Native	1	Commerciale	C++	Oui	Oui
Virtuoso	Non-Native	2	Open Sources	C	Oui	Oui
JenaTDB	Native	3	Open Source	JAVA	Non	Oui
Amazon Neptune	Native	4	Commerciale	SAAS	Non	Oui
GraphDB	Native	5	Les deux	JAVA	Non	Oui

TABLEAU 1.7 – Récapitulatifs des meilleurs TripleStores

1.7 Conclusion

Ce chapitre est une introduction aux bases de connaissances, où toutes les notions et technologies utilisées dans nos travaux ont été décrites. Ces descriptions concernent le langage de modélisation des données RDF, les langages de schéma RDFS et OWL, le langage d'interrogation SPARQL, ainsi que les systèmes de stockage des données RDF (TripleStores).

La formalisation du langage d'interrogation des bases de connaissances SPARQL a eu lieu en 2007, bien plus récent que le langage SQL apparu en 1974. Cette chronologie donne un sens au manque de techniques coopératives liées à l'interrogation de bases de connaissances réalisée à l'aide du langage SPARQL. Dans cette thèse, nous cherchons à accompagner l'utilisateur pour adapter les requêtes qu'il a exprimé à ses attentes. Dans le chapitre suivant, nous passons en revue les techniques coopératives existantes de la littérature dans des domaines variés, comme les bases de données relationnelles, les systèmes d'information et les bases de connaissances. Nous positionnons également nos travaux par rapport à l'existant.

Chapitre 2

Approches de traitement coopératif : Un panorama

Sommaire

2.1 Inroduction

De nombreux utilisateurs sont confrontés à différents problèmes lors de leur interrogation des bases de données. Ceci est dû principalement au fait qu'ils n'ont qu'une connaissance partielle de la source de données interrogée. Ces utilisateurs peuvent, par exemple, formuler des requêtes dont les résultats ne correspondent pas à leurs attentes, conduisant à plusieurs questionnements. Pour répondre à ces questionnements, différents travaux ont vu le jours dans plusieurs domaines, citons par exemple les langages naturels, les bases de données (relationnelles et RDF) ou les systèmes de recommandation. L'origine de ces travaux sur les réponses coopératives remonte aux années 80 pour le traitement du langage naturel [Kap82]. Le système doté d'un comportement coopératif a pour objectif de fournir des réponses aux requêtes des utilisateurs de manière analogue aux humains, et non comme des machines, dans le but de minimiser la frustration qu'un utilisateur pourrait rencontrer, comme par exemple dans le cas d'un ensemble vide de résultats.

Dans la section 2, nous passons en revue les différents questionnements des utilisateurs. Ensuite, dans la section 3, nous nous intéressons plus particulièrement aux travaux réalisés dans le contexte du problème des réponses vides, ce qui nous permettra de positionner nos travaux sur ce problème dans le contexte des données RDF. De la même façon, dans la section 4, nous détaillons les travaux réalisés dans le contexte du problème des réponses pléthoriques, afin aussi de nous positionner par rapport à l'existant.

2.2 Typologie des problèmes : Why Empty? Why So Many? Why Not? Why answers?

Il existe quatre principaux problèmes que les systèmes coopératifs sont sensés surmonter (résumés dans le tableau 2.1).

— *Why-Empty [BHP08] :* lorsqu'un utilisateur lance une recherche et que la réponse retournée est vide. Par exemple, si un touriste exprime une requête pour trouver un hôtel mais qu'il n'obtient aucun résultat.

— *Why So Many [VTBL16] :* lorsqu'un utilisateur obtient trop de résultats. Ce problème apparaît lorsque les critères de sa recherche sont peu sélectifs. Par exemple, un touriste, qui cherche un hôtel en France, aura comme réponse tous les hôtels existant en France, ce qui ne va pas l'aider à choisir.

— *Why-not [CJ09] :* lorsqu'un utilisateur obtient un résultat qui ne contient pas un élément qu'il attendait. Par exemple, un touriste qui aurait exprimé une requête pour trouver les lieux à visiter à Paris dont le résultat ne contiendrait pas la tour Eiffel.

— *Why Answer [ILZ13] :* un élément non attendu par l'utilisateur est présent dans l'ensemble de réponses. C'est le problème dual au précédent qui survient, par exemple, si la requête permettant de trouver les lieux à visiter à Paris retourne le Mont-Saint-Michel.

Type de requête	Problème	Résultat
Why Empty?	Nombre de résultats (r) égal à zéro	$\|r\| = 0$
Why So Many?	Nombre de résultats (r) supérieur à un seuil (K)	$\|r\| > K$
Why this Answer?	Un élément non attendu (e) est dans le résultat (r)	$e \in r$
Why Not?	Un élément attendu (e) n'est pas dans le résultat (r)	$e \notin r$

Tableau 2.1 – Synthèse de différents problèmes sur le résultat d'une requête

2.3 Approches coopératives pour le problème des réponses vides (Why Empty)

Dans cette section, nous proposons tout d'abord une revue complète des approches existantes pour résoudre le Problème des Réponses Vides (PRV) dans le contexte des bases de connaissances. Une comparaison entre les principales approches est également réalisée. Ensuite, une analyse critique des approches les plus proches de notre proposition est présentée.

Étude comparative

Dans le contexte des bases de connaissances, le problème des réponses vides a été résolu par plusieurs approches dont voici les idées de base :

— compléter la base de connaissances à l'aide de règles logiques [GTHS15],

— vérifier les données lors de la formulation de la requête pour éviter les réponses vides [Cam14],

— générer un schéma relationnel à partir des données de la base de connaissances pour aider les utilisateurs à formuler des requêtes [PPEB15],

— rendre plus flexible les conditions de la requête pour renvoyer des réponses alternatives [HPW09, HLZ12, FJH14, CFPW14, HMPS12, ERW11, DSWD09].

Le tableau 2.2 compare les familles de ces approches sur la base des critères suivants :

— *orientée données ou requête :* certaines approches se concentrent sur l'amélioration de la base de connaissances (sa complétion ou l'extraction d'un schéma relationnel) afin d'éviter le problème de réponses vides, tandis que d'autres se concentrent sur les requêtes des utilisateurs ayant posé le problème ;

— *avant/après la formulation de requête :* ce critère spécifie si l'approche considérée anticipe le problème de réponse vide, c'est-à-dire essaie de le résoudre avant la formulation de la requête ou agit dès que ce problème apparaît (pendant ou après la formulation de la requête) ;

— *intervention de l'utilisateur :* les approches peuvent demander des efforts à l'utilisateur pour que celui-ci fournisse certaines informations utiles à la résolution du problème des réponses vides ;

— *aide à comprendre les données :* les approches peuvent aider ou non l'utilisateur à mieux comprendre le contenu ou la structure de la base de connaissances.

Ainsi, deux grandes catégories d'approches peuvent être distinguées.

1. Les approches qui complètent la base de connaissances ou en dérivent un schéma relationnel considèrent que le problème de réponses vides provient de la base de connaissances, car celle-ci est incomplète et que sa structure est masquée pour l'utilisateur. Ainsi, ces approches anticipent le problème de réponse vide en modifiant la base de connaissances.
2. Les approches qui vérifient les données lors de la formulation de la requête ou qui proposent une relaxation de la requête utilisateur se concentrent sur la requête fournie par l'utilisateur. Les approches qui vérifient les données aident l'utilisateur à formuler sa requête pour éviter le problème de réponse vide. L'inconvénient de ces approches est qu'elles nécessitent généralement de nombreuses interventions de l'utilisateur. Inversement, les approches qui se basent sur la relaxation de requêtes prennent la requête défaillante comme point de départ et fournissent à l'utilisateur des requête alternatives qui réussissent. Le processus de relaxation peut être complètement automatique [FJHB16] ou guidé par l'utilisateur [DSWD09].

Approche	Orientée données/-requête	Avant/après la formulation de requête	Intervention d'utilisateur	Aide à comprendre les données
Complétion des données	Données	Avant	Non	Non
Formulation de requête interactive	Requête	Pendant	Forte	Oui
Schéma relationnel émergent	Données	Avant	Non	Non
Relaxation de requête	Requête	Après	Variable	Oui

Tableau 2.2 – Comparaison de différentes approches pour le problème PRV

Toutes ces approches peuvent être complémentaires pour traiter le problème de réponse vide. Dans la section suivante, nous détaillons les approches les plus proches de nos travaux.

2.3.1 Reformulation de requêtes

Dans ce cadre, les données sont considérées comme étant correctes et le but est de corriger la requête qui est donc mal formulée. Cette technique vise à transformer la requête d'une manière à ce que cette dernière soit correctement formulée et réponde aux attentes de l'utilisateur. Deux types de reformulations existent : (i) la première consiste à impliquer l'utilisateur dans le processus de reformulation (approche interactive), (ii) la deuxième fonctionne d'une manière automatique.

2.3.1.1 Reformulation interactive

Cette approche se base sur les retours utilisateur afin de reformuler la requête. Dans un premier temps, l'utilisateur lance une recherche qui retourne des résultats non convaincants. Le système coopératif récupère ensuite les retours de l'utilisateur concernant les résultats et procède à la transformation de la requête.

Dans leur travail réalisé sur la reformulation des requêtes [ILZ13], Islam et al. demandent à l'utilisateur d'annoter l'ensemble des résultats retournés. Les annotations sont appelées *User Feedbacks*. Le principe de système de reformulation est, dans un premier temps, l'examen préalable des informations que le système récolte auprès de l'utilisateur (User Feedbacks). Puis, dans un second temps, le système prend en entrée les retours utilisateur et décide de la manière de reformuler la requête considérée.

La reformulation interactive se limite au nombre d'interactions entre le système et l'utilisateur. Si l'utilisateur n'est pas satisfait suite à une première itération, le système est amené à en faire une autre, jusqu'à ce que l'utilisateur soit satisfait. Le processus peut ainsi être très long pour trouver la bonne reformulation correspondant au mieux aux attentes de l'utilisateur. Pour contourner ce problème, une solution automatisant le processus est proposée : la reformulation automatique de requêtes.

2.3.1.2 Reformulation automatique

Dans cette approche, l'utilisateur est exclu du processus. Ainsi, l'idée consiste à remplacer l'utilisateur par des modèles, comme les préférences, les besoins, les contextes et d'autres éléments décrivant l'utilisateur [SKP11, KI05].

Il existe plusieurs travaux dans le contexte de la reformulation automatique des requêtes. Koutrika et Ioannidis [KI05] utilisent l'historique des User Feedbacks qu'un utilisateur aurait déjà envoyé afin de construire un modèle qui permet d'ajuster la requête. Lidong et al. [BLW11] exploitent les Logs de l'utilisateur. Plusieurs algorithmes sont proposés pour définir des modèles utilisateur [VBEZ10, PPNH07], il existe des algorithmes génériques [Kob01], mais aussi des algorithmes spécifiques comme, par exemple, dans le E-commerce [LTYX07].

Comme nous le constatons, la coopération avec l'utilisateur est un élément essentiel dans les techniques de reformulation de requêtes, que ce soit pour les approches interactives qui impliquent directement l'utilisateur ou pour les approches automatiques qui se basent sur les préférences et/ou l'historique de l'utilisateur afin de créer un modèle qui remplace ce dernier.

2.3.2 Suggestion de requêtes

Cette approche coopère avec l'utilisateur dans le but d'exprimer son besoin d'une manière correcte, tant au niveau de la sémantique qu'au niveau de la syntaxe. La suggestion de requêtes a été initiée dans le but de satisfaire l'utilisateur ayant de fausses présuppositions. Wesley [WW08] a mené une étude sur cette question dans le contexte des bases de données relationnelles. Il a défini deux catégories de fausses présuppositions menant à l'insatisfaction des utilisateurs. La première catégorie de présuppositions concerne le modèle des données. La deuxième catégorie se base sur les données que l'utilisateur suppose être présentes dans la base qu'il interroge, alors qu'elles n'y sont pas (Wesley qualifie cela de *misconception*). Afin de faire face à ces deux catégories de présuppositions, trois méthodes de suggestion de requêtes ont été proposées, la première concerne les interfaces aidant l'utilisateur à construire sa requête ; la deuxième est un système d'auto-complétion qui assiste l'utilisateur pendant la construction de sa requête ; la troisième évalue et vérifie les résultats de la recherche utilisateur en même temps que la construction de la requête.

2.3.2.1 Interfaces graphiques

Les solutions proposant des interfaces graphiques pour assister un utilisateur dans la tâche d'interrogation des bases de données, utilisent des composantes visuelles [CCLB97]. L'utilisateur peut trouver plusieurs intérêts dans ces approches, car :

— elles transforment le langage de requête sous forme d'objets visuels. Toutes les fonctionnalités du langage de requêtes sont conservées par la représentation visuelle ;

— elles suivent l'intuition humaine afin de forger l'interaction coopérative avec l'utilisateur.

Dans le contexte des bases de données relationnelles, Catarci et al. [CCLB97] ont présenté trois approches de systèmes qui reposent sur des interfaces graphiques : les diagrammes, les formulaires et les systèmes mixes qui fusionnent diagrammes et formulaires. Smart et al. [SRB+08] ont développé NITELIGHT, un outil pour aider à la création de requêtes SPARQL. Ce système fait partie de l'approche par diagrammes où les composants de la requête SPARQL sont modélisés, par exemple, sous forme de rectangles, de carrés ou de lignes.

La transformation radicale du langage de requête en objets visuels peut causer des soucis implicites dans le sens où l'utilisateur doit se familiariser avec un environnement visuel afin de

formuler sa recherche. Aussi une approche aidant l'utilisateur pour formuler sa requête d'une manière explicite a été proposée ; il s'agit de l'approche d'auto-complétion.

2.3.2.2 Complétion automatique de la requête

En plus d'aider l'utilisateur à éviter les fausses présuppositions, cette approche propose de compléter la requête qu'un utilisateur a commencé à formuler par plusieurs propositions. L'utilisateur choisit ensuite la proposition qui complète sa requête. Les éléments utilisés par les systèmes d'auto-complétions sont les suivants.

— le modèle des données [BYK11],

— le journal des requêtes déjà exécutées [XQW+13],

— l'ensemble des préférences utilisateurs [JKCC14].

2.3.2.3 Vérification continue

Les deux techniques précédentes consistent à assister l'utilisateur pour la reformulation de sa requête afin que cette dernière corresponde aux données de la base. Une troisième approche, de Arnab et Jagadish [NJ07], assiste l'utilisateur avec un retour de résultats continu, qui se fait en même temps que la reformulation de la requête. Ainsi, le principe de la vérification continue est de retourner des réponses dès le début de la formulation de la requête ; si l'utilisateur se rend compte que les résultats intermédiaires ne correspondent pas à ses attentes, il peut alors changer sa requête avant même de finir sa construction.

2.3.3 Interrogation par l'exemple

Le caractère coopératif apporté par l'approche de suggestion de requêtes dans la construction de requêtes a été rapidement critiquée par Motro [Mot96], qui a démontré que l'assistance à la formulation des requête était une tâche fastidieuse. En effet, dans le cas où l'utilisateur n'a pas une connaissance précise de son besoin, la construction de requête par interaction avec l'utilisateur devient une tâche complexe. L'interrogation par l'exemple est une solution qui assouplit cette difficulté. Elle a été initiée par IBM dans le contexte des données relationnelles. En 1974, Zloof et al. [Zlo75a] ont pensé à cette approche pour aider les utilisateurs non spécialistes des bases de données, dans le but de leurs permettre d'exécuter des requêtes précises d'une manière simple et sans fausses présuppositions. Dans cette section, nous présentons cette approche pionnière qui concerne les base de données relationnelles. Nous verrons ensuite comment elle a été appliquée dans le contexte des bases de connaissances.

Interrogation par l'exemple pour des données relationnelles

Zloof et al. [Zlo75a] ont proposé un système qui prend en entrée des exemples de réponses attendues par les utilisateurs, et qui, en sortie, renvoie des réponses semblables à au moins un de ces exemples. Le système propose une interface graphique qui offre à l'utilisateur une visualisation des objets présents dans la base de données, comme les tables et leurs attributs à partir desquels l'utilisateur constitue les exemples de réponses. Plusieurs systèmes de bases de données ont intégré les travaux de Zloof et al., comme par exemple, Access du pack Office de Microsoft.

Interrogation par l'exemple pour les données RDF

Dans le contexte des données RDF, Nandish et al. [JKL+14, JKL+15] ont adapté l'interrogation par l'exemple pour les données RDF, où un utilisateur propose au système un Graphe RDF contenant les réponses souhaitées. Le système cherche ensuite dans l'ensemble des données RDF présentes dans la base de connaissances des sous-graphes semblables à la proposition de l'utilisateur.

2.3.4 Relaxation de requêtes

La dernière méthode coopérative considérée consiste à modifier les contraintes de la requête. Elle se divise en deux catégories.

— Relaxation par relâchement de conditions : cette catégorie fait en sorte que la requête soit moins sélective. Ainsi, la requête relaxée retourne plus de résultats, qui contiennent les résultats de la requête initiale.

— Relaxation par suppression de conditions : dans le même objectif de retourner plus de résultats, cette démarche supprime une ou plusieurs conditions de la requête initiale.

2.3.4.1 Relaxation par relâchement de conditions

Le relâchement de conditions de requête mène à une requête moins contraignante dans le but de retourner plus de résultats [FJHB15]. Pour ce faire, un modèle de données est nécessaire pour guider la relaxation.

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 schémas de classes et de propriétés) [HPW09, HLZ12, FJH14, CFPW14], les mesures de similarité [HMPS12, ERW11] et les préférences utilisateur [DSWD09]. Ces opérateurs génèrent un ensemble de requêtes relaxées, ordonnées par similarité avec la requête d'origine et exécutées dans cet ordre [HPW09, HLZ12, RK13]. Les opérateurs de relaxation peuvent être directement utilisés par l'utilisateur dans sa requête [FJH14, CFPW14] ou utilisés conjointement avec des règles de réécriture de requêtes pour effectuer la relaxation [DSWD09]. Avec ces approches, les causes d'échec de la requête sont inconnues, ce qui peut conduire à exécuter des requêtes relaxées inutiles. La prochaine section présente la relaxation des requêtes par suppressions de conditions qui inclue l'identification des causes d'échec de la requête.

2.3.4.2 Relaxation par suppression de conditions

Cette catégorie consiste à réduire le nombre de conditions afin que la requête ne retourne pas zéro résultat. Godfrey [God97] a proposé le concept des XSS (MaXimal Successful Subqueries) qui correspondent aux plus grandes parties de la requête qui retourne au moins un résultat. Souvent les XSS sont calculées suite à l'identification des plus petites partie de la requêtes qui causent l'échec (les raison pour lesquelles la requête retourne zéro résultat), et qui sont appelées MFS pour Minimal Failing Subqueries.

Fokou et al. [FJHB16, FJHB17] ont abordé ce problème dans le contexte des données RDF, en définissant d'abord les approches LBA et MBA pour calculer les MFS et XSS de la requête [FJHB17] et en proposant des stratégies de relaxation basées sur les MFS qui identifient les

requêtes relaxées échouant nécessairement [FJHB16]. Nos deux premières contributions présentés dans le chapitre 3 et 4, sont 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 celui proposé par [RK13]. Toutefois, ce travail utilise uniquement les valeurs de confiance pour ordonner les résultats par leur fiabilité. Ils ne considèrent pas, comme nous le faisons dans nos contributions, les requêtes qui ne renvoient aucun résultat satisfaisant la requête d'utilisateur.

La question du calcul des MFS et des XSS a également été abordée dans le contexte des bases de données relationnelles [God97], des systèmes de recommandation [Jan09] et des requêtes floues [PS15]. Le travail le plus proche du nôtre est celui de Pivert et Smits [PS15] 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 incertaines sur les BC incertaines. Dans ce nouveau contexte, un résumé de la partie pertinente des BC ne peut pas être efficacement calculé [FJHB17].

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 [MT97, GKM$^+$03]. 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 notre travail 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.

2.4 Approches coopératives pour le traitement du problème des réponses pléthoriques (Why So Many)

L'un des problèmes, rencontrés dans les systèmes d'interrogation de bases de données est le Problème des Réponses Pléthoriques (PRP). D'une part les conditions présentes dans la requêtes sont toutes satisfaites et d'autre part, ces conditions sont très permissives parce que l'utilisateur n'est pas précis dans sa recherche. Donc, ce problème entraîne un nombre important de réponses par rapport à ce que l'utilisateur attend. Le but de cette section est d'introduire les travaux déjà réalisés pour faire face au problème PRP.

Dans l'état de l'art sur les requêtes pléthoriques nous retrouvons plusieurs travaux qui peuvent être classés en trois familles (voir le tableau 2.3).

1. La première famille est celle des travaux consistant à reformuler la requête initiale [BHP06, SAM14] afin de rendre ces conditions plus restrictives et précises. Cette famille de travaux est duale à la relaxation des requêtes où on cherche à relâcher les conditions pour avoir plus de résultats.
2. La deuxième famille concerne les travaux visant à ordonner les réponses afin de retourner les K meilleurs réponses.

Approche	Orientée données/-requête	Avant/après la formulation de requête	Intervention d'utilisateur	Aide à comprendre les données
Renforcement des requêtes	requêtes	Après	Non	Non
Classement des réponses	Données	Après	Oui	Non
Classification des réponses	Données	Pendant	Non	Non

Tableau 2.3 – Comparaison de différentes approches pour le problème PRP

3. La troisième et dernière famille vise à partager l'ensemble des réponses en différentes classes conventionnellement choisies.

2.4.1 Renforcement des conditions de la requête

Cette approche est duale à la relaxation des requêtes vue en section 2.3.4. Inversement à cette dernière, le renforcement consiste à rendre une requête plus sélective. Ainsi la requête retourne moins de résultats.

Le renforcement de requête se divise aussi en deux catégories :

— renforcement par intensification de conditions [BHPS10],

— renforcement par ajout de conditions de filtre [Oza94].

2.4.1.1 Renforcement par intensification de conditions

Dans cette approche nous supposons qu'un utilisateur qui lance une recherche reçoit trop de réponses à sa requête. De façon analogue à la relaxation de requête, cette approche se base sur le modèle des données. Suivant le modèle de données adressé, le système va proposer un renforcement de type. Par exemple, si l'utilisateur cherche tous les Documents qui existe dans une base de données, le système va proposer de chercher tous les livres ou tous les magazines. Ainsi, l'ensemble de résultats retournés à l'utilisateur est un sous ensemble des réponses de la requête initiale.

2.4.1.2 Renforcement par ajout de conditions de filtre

L'objectif de cette approche, qui traite le problème de réponses pléthoriques, se base sur l'extension de la requête en ajoutant de nouvelles conditions. Les conditions que le système choisit, doivent être sémantiquement cohérentes avec les conditions de la requête initiale. Cette approche est particulièrement adaptée avec les requêtes contenant peu de conditions (entre 1 et 3). Par exemple, considérons un utilisateur qui cherche des livres avec leurs nombres de pages et qui reçoit trop de résultats. L'idée est de rajouter une condition corrélée avec cette recherche comme par exemple de chercher les livres avec leurs nombres de pages, qui sont édités par Springer.

2.4.2 Classement des réponses

Cette famille de travaux [CDHW04, SWHL06a] s'intéresse à trouver des techniques de classement des réponses pour présenter à l'utilisateur celles ayant les meilleurs scores. Afin de permettre le calcul des scores des réponses, un poids ou une valeur réelle est calculée et associée à chaque réponse retournée par la requête initiale. L'objectif est de retourner à l'utilisateur les K réponses les plus pertinentes à l'aide de fonctions de calcul de score.

A titre d'exemple, l'approche Top-K se base fondamentalement sur des approches calculant (1) les scores des résultats et (2) permettant de choisir les K meilleurs résultats parmi un ensemble de réponses potentielles. Cette section est décomposée en deux grandes sous-parties. D'abord, nous présentons les travaux pionniers dédiés au problème des réponses pléthoriques et qui concernent les systèmes de recherche d'information (RI). Ensuite, nous étudions les travaux réalisés dans le contexte des BD relationnelles.

2.4.2.1 Modèles de classement dans les systèmes RI

Le classement automatique des réponses de requêtes a été largement abordé dans les système RI [SM86, Rob97, Kle99]. Les pages Web retournées en résultats des recherches portant sur des mots clés sont des pages automatiquement ordonnées par ordre de pertinence (les plus pertinentes précèdent les moins pertinentes). Ce qui nous intéresse dans ces systèmes de recherche est le mécanisme d'ordonnancement ou les techniques utilisées pour calculer la pertinence des pages Web. Les modèles qui ont été utilisés sont les modèles d'espace vectoriel et de recherche probabilistes. Ils sont combinés avec des méthodes de classement basées sur les liens pour offrir à l'utilisateur des documents pertinents.

Modèle vectoriel

Le modèle vectoriel [SM86] représente les documents d'un espace de recherche sous forme de vecteurs de N-dimension où N correspond au nombre de termes présents dans la collection. Chaque élément du vecteur représente un terme associé à un pois calculé avec la méthode TF-IDF [1] (de l'anglais Term Frequency-Inverse Document Frequency). Cette dernière permet la pondération souvent utilisée dans les système RI. La méthode a pour but de mesurer l'importance d'un terme contenu dans un document, lié à une collection. Le poids augmente en fonction du nombre d'occurrences du terme dans le document.

Modèle probabiliste

Dans ce modèle plusieurs approches ont tenté de définir le calcul de la pertinence des pages Web en s'appuyant sur les probabilités. Le modèle probabiliste estime la probabilité d'observer des termes liés au document et à la requête par la probabilité de pertinence du document vis à vis de la requête.

Dans notre domaine, ce modèle de probabilité sert à donner un ordre de pertinence des résultats [Rob97] (Ranking Documents In Decreasing Order Of Probability Of Relevance). Il fournit ainsi un classement des documents dans l'ordre décroissant de probabilité de pertinence

1. TF-IDF [Wikipédia] est une méthode de pondération souvent utilisée en recherche d'information et en particulier dans la fouille de textes. Cette mesure statistique permet d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus.

pour l'utilisateur ayant soumis la requête, où les probabilités sont estimées en utilisant toutes les informations disponibles.

Modèle basé sur les liens

Les modèles basés sur les liens se fondent sur les liens qu'une page Web a avec toute autre page [BP98, BH98, Kle99]. Pour donner un exemple concret nous considérons le moteur de recherche Google qui utilise son système PageRank [BP98] basé sur les liens entre les pages Web afin de les classer. PageRank calcule le score d'une page Web en exploitant le graphe inféré à partir de la structure des liens du Web. L'essentiel dans ce système est qu'une page Web référencée par plusieurs autres pages est plus importante ou est mieux classée qu'une page ayant moins de liens. Dans leur article, Manning et al. [MRS08] détaillent les systèmes RI se basant sur les liens entre les pages.

Système de classement dans le contexte des BD

Le classement des résultats des requêtes BD est un problème largement abordé dans la littérature [CD03, SCW04, SWHL06b, SDMB02, LWP00]. Ces travaux prennent en entrée une requête, puis utilisent diverses sources de connaissances afin de classer les résultats de la requête. Des travaux tels que [CD03, SCW04, SWHL06b] utilisent la charge de travail et les statistiques de la BD. D'autres travaux comme [LWP00, SDMB02] se basent sur les retours utilisateur.

Système de Chaudhuri et al [CD03]

Chaudhui et al. [CD03] proposent une fonction de classement (QF_W) qui exploite la charge de travail de la BD afin de classer les résultats d'une requête SQL [CD03]. Leurs approches se basent sur la fréquence d'occurrence des valeurs d'attributs non spécifiés dans la requête.

Plus formellement, soit une relation R, chaque tuple de R se compose de N attributs $A_1, ..., A_N$. Et soit Q une requête avec des attributs (par exemple $A_1, ..., A_i$ et $i < N$). Le reste des attributs $A_{i+1}, ..., A_N$ n'est pas spécifié dans la requête. Le score de pertinence de chaque résultat t de Q est calculé comme suit : $QF_W(t) = \sum_{K=i+1}^{N} \frac{F(t.A_K)}{F_{max}}$.

Où $F(t.A_K)$ est la fréquence d'occurrence de la valeur $t.A_K$ de l'attribut A_K dans la charge de travail de la BD, et F_{max} est la fréquence des valeurs les plus fréquentes dans la charge de travail.

Par exemple considérons un marchand de téléphones qui cherche dans sa base de données, une requête avec des conditions peu sélectives comme une capacité de stockage de 32 Go et une couleur noire. Cette requête retourne de nombreux résultats. La proposition de [CD03] utilise la charge de travail de la base de données pour examiner d'autres attributs qui ne sont pas présents dans la requête initiale afin de classer les résultats. Donc, si dans la charge de travail de la BD il existe plus de requêtes recherchant des téléphones noirs de 32 Go et de marque *Iphone* que des requêtes qui cherchent des téléphones noirs de 32 Go et de marque *Samsung*, alors le système va classer les téléphones de marque *Iphone* avant les téléphones de marque *Samsung*.

Système de Recherche Probabiliste d'Informations (PIR)

Pour répondre au problème des réponses pléthoriques dans le contexte des bases de données relationnelles, [SCW04] a proposé une fonction de classement des résultats qui se fonde sur des

modèles RI (les modèles vectoriels et probabilistes cités dans la section précédente). Dans ce travail, les auteurs ont adapté les deux modèles cités précédemment en exploitant la structure des données ainsi que la corrélation présente entre la structure des données et la charge de travail de la base de données. Plus concrètement, ils font la proposition d'une fonction de classement des résultats d'une requête retournant trop de résultats qui dépend de deux points : (1) un score global qui prend en considération l'importance générale des attributs non spécifiés dans la requête initiale (2) un score conditionnel qui considère les dépendances (ou les corrélations) entre les valeurs des attributs qui sont spécifiés dans la requête et ceux non spécifiés. Ces scores sont estimés en utilisant la charge de travail de la BD ainsi que l'analyse des données. Par exemple, pour une requête d'une personne cherchant à acheter une maison à Marseille avec vue sur la mer, une maison avec un anneau sur un quai adjacent aura un score élevé car les personnes désirant une maison face à la mer sont possiblement à la recherche d'une zone d'amarrage. L'ancienne charge de travail de la BD doit révéler, via les requêtes déjà lancées, que les utilisateurs cherchant une maison face à la mer recherchent aussi un anneau (corrélation entre les valeurs des attributs).

Système de classement pour E-commerce (QRRE)

Dans les système QRRE (Query Result Ranking for E-commerce) [SWHL06b], les auteurs ont proposé une méthode de classement automatique permettant le tri des réponses d'une requête exécutée sur une BD Web d'E-commerce, en utilisant seulement les techniques d'analyses de données. Considérant un tuple $t = (t.A_1, ..., t.A_N)$ dans l'ensemble des résultats T_Q d'une requête Q lancée par un acheteur, QRRE assigne un poids w_i à chaque attribut A_i qui reflète l'importance de ce dernier par rapport à l'utilisateur. w_i est évalué par la différence entre la distribution des valeurs de A_i dans l'ensemble de résultats T_Q et leur distribution dans la BD entière. Plus la divergence est grande, plus A_i est important pour un acheteur. Prenons l'exemple d'une BD contenant des maisons à vendre en France et considérons la requête Q avec la condition d'avoir vue sur la mer. Intuitivement, la valeur du prix d'un résultat de Q se distribue dans un petit intervalle avec une très grande moyenne, Alors que la valeur du prix d'un tuple dans la BD se distribue dans un grand intervalle avec une moyenne plus basse. La différence entre les distributions montre une corrélation très proche entre l'attribut non spécifié, le prix, et le critère de vue sur la mer. En revanche, un attribut portant sur la taille sera considéré comme moins important pour l'utilisateur puisque sa distribution dans les maisons qui sont face à la mer peut être similaire à sa distribution dans la BD entière. En plus du poids associé à chaque attribut, QRRE assigne aussi un score de préférence p_i à chaque valeur d'attribut $t.A_i$. Cette préférence est calculée suivant les deux hypothèses suivantes.

— Un produit avec un prix bas est toujours plus souhaité par un acheteur qu'un produit plus cher si les autres attributs des deux produits ont les mêmes valeurs. Par conséquent QRRE assigne un score de préférence plus bas à un prix élevé, et une préférence plus grande à un prix moins élevé.
— Une valeur d'un attribut autre que le prix avec une grande préférence de l'utilisateur correspond à un prix cher. Donc, dans le cas d'un attribut A_i autre que le prix, QRRE convertit premièrement la valeur $t.A_i$ en une valeur de prix pv qui est le prix moyen du produit pour $A_i = t.A_i$ dans la BD. Ensuite, QRRE assigne un large score de préférence à $t.A_i$ si pv est large. Par exemple, une maison avec une surface importante, que la majorité des acheteurs préfèrent, est souvent plus coûteuse qu'une petite maison.

Finalement, le poids de l'attribut et sa valeur de préférence sont combinés afin de trouver le score de classement de chaque tuple t, comme suit : $QRRE(t) = \sum_{i=1}^{n} w_i p_i$.

Les scores de classement des résultats sont stockés et les Top K résultats sont retournés à l'utilisateur. QRRE est une approche utilisée pour surmonter le problème d'obtenir trop de

résultats. Cette approche ne dépend pas du domaine ni de la charge de travail de la BD. Cependant cette solution peut avoir un temps d'exécution considérable, notamment dans le cas des requêtes peu sélectives, car différents histogrammes nécessitent d'être calculés au moment de l'exécution de la requête.

Système basés sur les retours utilisateur (Feedback-based Systems)

Une autre approche permet de classer les réponses d'une requête et d'exploiter les retours utilisateur afin de savoir quels tuples de la BD sont intéressants. Les techniques des retours utilisateur ont été étudiées dans le contexte de la recherche d'images [LWP00, SDMB02], et ont souvent été reliées aux approches des requêtes par l'exemple (query-by-example) [Zlo75b]. La procédure générale suit le scénario suivant :

— l'utilisateur exécute une requête,
— le système retourne un ensemble de résultats,
— l'utilisateur marque certaines réponses comme pertinentes ou non pertinentes,
— le système reconstruit une requête sensée être plus proche des réponses pertinentes et plus éloignée des réponses non pertinentes,
— le système affiche les réponses les plus similaires à la requête.

Cette procédure peut être itérative jusqu'à ce que l'utilisateur soit satisfait des réponses de la requête. Ces approches proposent une méthode pour la réduction du nombre de résultats de la requête. Cependant, elles ne sont pas forcement populaires. En effet, les utilisateurs peuvent ne pas souhaiter faire de retours, ou simplement ne pas vouloir intéragir avec le système.

2.4.3 Classification des réponses pléthoriques

La classification des résultats pléthoriques est une méthode de regroupement des résultats dans des groupes distincts. Cela permet à l'utilisateur de sélectionner et d'exploiter les groupes qui sont les plus proches de sa recherche et d'ignorer ceux qui ne le sont pas. Cette approche a été utilisée pour l'organisation des résultats dans les moteurs de recherche [JR71, vRC75], mais a été adaptée dans le contexte des BD relationnelles [KCwH04, BBM05]. Nous détaillons par la suite les travaux réalisés dans le contexte des bases de données relationnelles.

2.4.3.1 Système de Chakrabarti et al.

[KCwH04] ont proposé une méthode de catégorisation automatique des résultats d'une requête. Cette méthode crée un arbre de navigation pour chaque requête Q en se basant sur le contenu de l'ensemble des réponses T_Q. Une catégorisation hiérarchique de T_Q est un partitionnement récursif des tuples de T_Q en se basant sur les attributs et leurs valeurs. c'est-à-dire que les résultats de la requête sont regroupés dans des catégories imbriquées. Les valeurs et les attributs qui s'affichent dans l'arbre et qui permettent la séparation des niveaux ont un ordre calculé par l'analyse de l'agrégation des connaissances présentes dans la charge de requêtes de la BD.

2.4.3.2 Système QSQR (Overlapping CluStering of Query Results)

Dans [BBM05], les auteurs ont proposé une approche pour la classification des résultats d'une requête SQL en se basant sur l'approche du regroupement single-link [AKJF99]. Considérant

une requête SQL en entrée, QSQR explore les résultats de la requête, et identifie l'ensemble des termes qui sont les plus pertinents par rapport à celle-ci ; chaque terme dans cet ensemble est associé à un score représentant sa pertinence. Ensuite, QSQR exploite les scores des termes et leurs associations avec les résultats de la requête afin de définir une mesure de similarité entre les termes. Cette mesure de similarité est utilisée par la suite pour regrouper les termes. Plus concrètement, nous considérons une requête Q, une table R et T_Q l'ensemble des résultats de la requête Q. QSQR fonctionne comme suit.

1. Associer un score s_x à chaque valeur d'attribut x (ou à un terme) dans T_Q. Les scores des termes (similaires au score TF-IDF utilisé dans les systèmes RI) sont définis dans le but de distinguer les valeurs d'attributs les plus populaires en leur associant un score plus élevé.
2. Calculer le contexte de Q.

Comme nous venons de le voir de nombreuses techniques ont été proposées pour traiter le problème des réponses pléthoriques et ceci principalement dans le contexte des BD relationnelles. Nous remarquons cependant qu'aucun de ces travaux ne s'est intéressé à identifier les causes d'échec d'une requête pour ce problème spécifique. C'est que nous proposons dans le chapitre 5 dans le contexte des bases de connaissances.

2.5 Conclusion

Les systèmes exhibant un comportement coopératif pour assister l'utilisateur dans la recherche d'informations sont nombreux et concernent des contextes différents. Avec l'émergence de jeux de données volumineux, leur interrogation devient de plus en plus complexe. A cause de cette complexité d'interrogation, l'utilisateur ne sait généralement pas comment réagir afin de trouver les réponses qu'il souhaite. Dans ce chapitre nous avons récapitulé ce qu'a été fait dans la littérature pour éviter la frustration des utilisateurs. Ainsi, nous avons présenter différentes approches coopératives qui s'adaptent à des scénarios divers. Dans la section 2, nous avons présenté et comparé les travaux réalisés dans le contexte du problème des réponses vides. Nous avons situé nos travaux réalisé dans ce contexte et que nous présentons dans les deux prochains chapitres. Dans la section 3, nous nous sommes intéressés au problème de réponses pléthoriques dans lequel se situe notre dernière contribution, présentée au chapitre 5.

Chapitre 3

Traitement de requêtes RDF incertaines à réponse vide

Sommaire

3.1 Introduction

Comme nous l'avons expliqué dans les chapitres précédents, 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 [LIJ+15], YAGO [HSBW13], Wikidata [Vra12b] et NELL [CBK+10] 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 [DGH+14a] et Walmart [DLT+13a]. Ces BC contiennent des milliards de faits représentés sous forme de triplets RDF *(sujet, prédicat, objet)* et peuvent être interrogées avec le langage SPARQL [SH13]. 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 ou 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 [Har09, TPR13]. 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, noté 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 [SAH+15]. 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 [FJHB17]. Deux catégories de sous requêtes sont ainsi identifiées et représentées par deux ensembles : les sous requêtes minimales qui échouent (MFS pour Minimal Failing Subqueries) et les sous requêtes maximales qui réussissent (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 ensemble non vide de résultats. L'utilisateur peut ainsi choisir d'exécuter l'une de ses requêtes pour obtenir une réponse non vide.

La suite de ce chapitre est organisée comme suit. La section 3.2 fournit la formalisation des notions de base en complément des définitions du premier chapitre (données RDF et requêtes RDF incertaines), et formalise le problème abordé à savoir les requêtes retournant zéro résultats dans le contexte des données RDF incertaines. Cette section motive notre contribution avec un exemple de requête sur une BC incertaine. La section 3.3 définit la condition nécessaire pour que l'algorithme de Fokou et al. [FJHB17] puisse être directement utilisé pour trouver les αMFS et αXSS d'une requête RDF incertaine défaillante. La section 3.4 décrit la mise en œuvre et l'évaluation expérimentale réalisée. Enfin, nous concluons sur les résultats présentés dans ce chapitre dans la section 3.5.

3.2 Notions de αMFS et αXSS

3.2.1 Définitions et illustrations

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, noté $Q' \subseteq Q$, ssi $\{i, \cdots, j\} \subseteq \{1, \cdots, n\}$. Si $\{i, \cdots, j\} \subset \{1, \cdots, n\}$, alors Q' est dite une *sous requête propre* de Q ($Q' \subset Q$).

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

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 1.4. La figure 3.1 présente le treillis des sous-requêtes de Q.

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

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 :

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

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

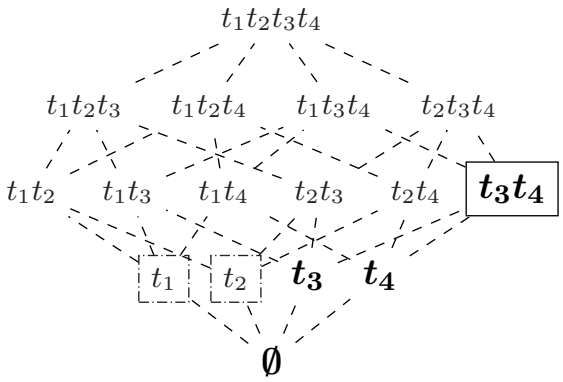
Définition 2 *Une XSS (requêtes maximale réussissant) Q^* de Q est définie comme suit : $[[Q^*]]_D \neq \emptyset \ \wedge \nexists\ Q'$ tel que $Q^* \subset Q' \wedge [[Q']]_D \neq \emptyset$. Par extension, une αXSS Q^* de Q, pour un α donné, est définie par : $[[Q^*]]^\alpha_D \neq \emptyset \ \wedge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

La requête Q a une seule 0,8 αXSS (t_3t_4) qui représente la plus grande requête alternative proposant des résultats :

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

3.2.2 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. L'objectif est de trouver un algorithme efficace pour calculer les αMFS et αXSS de Q pour un seuil α.



$\alpha = 0,8$

légende : Q : Échec **Q : Réussite** $\boxed{Q}$: αMFS $\boxed{\boldsymbol{Q}}$: **α**XSS

FIGURE 3.1 – Treillis des sous-requêtes de Q pour un $\alpha = 0.8$

3.3 Calcul des αMFS et αXSS

Dans cette section, nous proposons dans un premier temps une approche de base (naïve) s'appelant DFS (Depth-first search) qui réalise un parcours en profondeur du treillis afin de trouver l'ensemble des αMFS et αXSS d'une requête RDF défaillante pour un α donné. Dans un second temps, nous présentons notre adaptation de l'algorithme *Lattice-Based Approach* (LBA) proposé par Fokou et al. [FJHB17] pour calculer les αMFS et αXSS d'une requête RDF défaillante pour un α donné.

3.3.1 Approche de base : Depth-first search (DFS)

Dans cette approche de base, nous développons un algorithme qui parcourt l'intégralité du treillis pour trouver les αMFS et αXSS d'une requête RDF pour un α donné. Cet algorithme (voir algorithme 1) exécute ainsi $2^n - 2$ requêtes [1] où n est le nombre de patrons de triplets de la requête initiale Q. DFS est donc peu efficace dès que le nombre de patrons de triplets devient important. L'algorithme applique une recherche en profondeur, d'abord, pour chaque requête Q' du treillis (ligne 3), si Q' échoue pour le degré α (ligne 5) et toutes ses sous-requêtes réussissent (ligne 6), alors Q' est insérée dans la liste des αMFS. Ensuite la liste se met à jour, en supprimant tous les éléments contenant Q' (seuls les éléments minimaux sont conservés) (lignes 8). Si Q' réussit et ses super-requêtes échouent, alors Q' est insérée dans la liste des αXSS (lignes 9-12).

Pour donner un exemple d'illustration de DFS, nous considérons la requête $Q = t_1t_2t_3t_4$ et le treillis de notre exemple de motivation. DFS exécute $2^4 - 2 = 14$ requêtes pour trouver les deux αMFS (t_1, t_2) et la αXSS t_3t_4. Dans la section suivante, nous présentons notre deuxième approche qui améliore l'algorithme DFS en utilisant des techniques dont le but est d'éviter l'exécution de certains éléments du treillis.

1. C'est à dire l'algorithme DFS exécute tous les éléments du treillis excepté la requête initiale Q qui échoue de base et l'ensemble vide qui réussit par convention

Algorithm 1: Trouver les αMFS et αXSS d'une requête Q pour un seuil α

```
DFS(Q, D, α)
    inputs  : Une requête qui échoue Q = t1 ∧ ... ∧ tn ;
              une base de données RDF D ;
              un seuil α
    outputs: les αMFS et αXSS de Q
1   mfs^α(Q) ← ∅; xss^α(Q) ← ∅;
2   while List ≠ ∅ do
3       Q' ← List.element(); // Choisir un élément de la liste List
4       if Q' ⊄ MarkedQueries then // Q' n'est pas encore testée
5           if Q' échoue then
6               if ToutesSousRequêtesDe(Q') réussissent then
7                   mfs^α(Q) ← Q';
8                   mfs^α(Q).MettreAJour(); // Supprimer toute αMFS incluant Q'
9           else // Q' réussit
10              if ToutesSuperRequêtesDe(Q') échouent then
11                  xss^α(Q) ← Q';
12          MarkedQueries.add(Q');
13  return mfs^α(Q), xss^α(Q);
```

3.3.2 L'approche αLBA

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

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, c'est-à-dire les requêtes maximales qui n'incluent pas la αMFS trouvée précédemment et (3) exécuter les αXSS *potentielles*; si elles réussissent, ce sont des αXSS sinon, les étapes précédentes sont faites sur les αXSS *potentielles* qui échouent. Ces trois étapes sont détaillées par la suite.

3.3.2.1 Calcul d'une αMFS Q^* de Q

L'algorithme 2 permet de trouver une αMFS de Q en exécutant n requêtes, où n est le nombre de parton de triplets 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 [FJHB17].

Pour illustrer cet algorithme, nous considérons la requête de l'exemple introduit dans la section 3.2 avec $\alpha = 0,2$ (voir figure 3.2). 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 (figure 3.2.1), 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 (figure 3.2.2) 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 (figure 3.2.3). Ainsi, elle contient Q^*. Enfin, t_4 est supprimé et la sous-requête t_1 réussit (figure 3.2.4). Donc,

Algorithm 2: 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 $Q' \leftarrow Q' - t_i$;
5 **if** $[[Q' \wedge Q^*]]_D^\alpha \neq \emptyset$ **then**
6 $Q^* \leftarrow Q^* \wedge t_i$;
7 **return** Q^*;

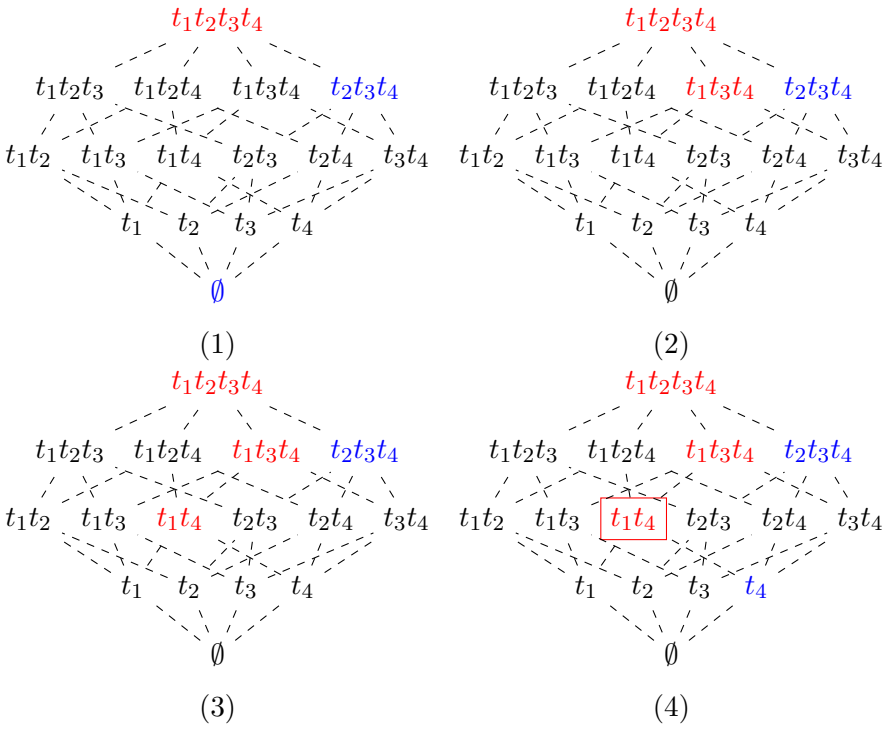


FIGURE 3.2 – Illustration de l'algorithme TouverUneαMFS

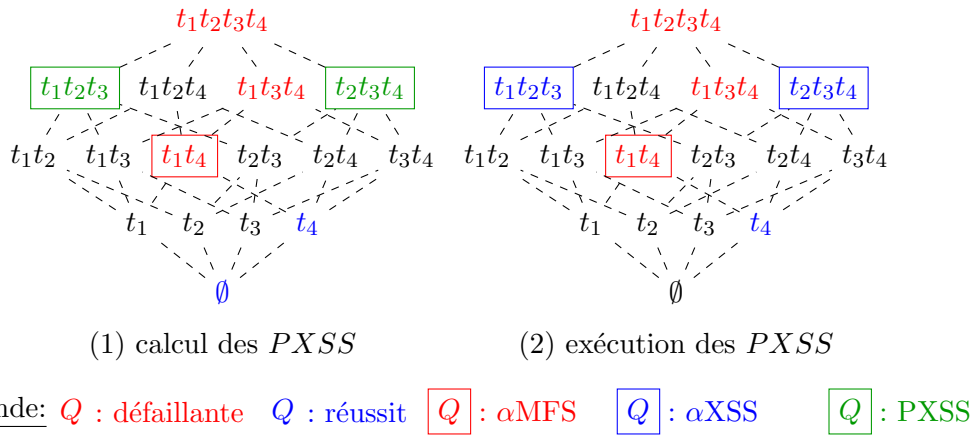


FIGURE 3.3 – Illustration de la découverte des αXSS

t_4 est un élément de Q^*. L'algorithme s'arrête et retourne le résultat $Q^* = t_1t_4$.

3.3.2.2 Calcul des αXSS potentielles

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

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

Dans notre exemple précédent, $pxss(Q, Q^*) = \{t_2t_3t_4, t_1t_2t_3\}$ (voir figure 3.3.1).

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 réussisse ne contienne pas une requête qui échoue pour le α donné. Ce pré-requis est discuté dans la section 3.3.3

3.3.2.3 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 3. Il est important de noter que cet algorithme évite de redécouvrir une αMFS plusieurs fois (lignes 13-15). Dans notre exemple illustratif, les deux αXSS potentielles $t_2t_3t_4$ et $t_1t_2t_3$ réussissent (voir figure 3.3.2), ce sont donc les αXSS de Q.

3.3.3 Condition nécessaire pour l'utilisation de l'algorithme αLBA

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 de monotonie décroissante 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

Algorithm 3: 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` α`XSS`
8 $xss^\alpha(Q) \leftarrow xss^\alpha(Q) \cup \{Q'\}$;
9 $pxss \leftarrow pxss - \{Q'\}$;
10 **else** `//` Q' `contient une` α`MFS`
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** $\{mfs^\alpha(Q), xss^\alpha(Q)\}$;

un seuil de confiance minimum spécifié, 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 3.4, 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, car elles ne sont pas monotones décroissantes. 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 monotone décroissante par rapport à l'inclusion ensembliste.

Définition 3 *soit* $aggreg : [0,1]^n \rightarrow [0,1]$ *une fonction d'agrégation, aggreg est monotone décroissante par rapport à l'inclusion ensembliste*[2] *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 1 *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* 1** *Nous considérons* $Q = t_1 \wedge \cdots \wedge t_n$ *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* $[[Q]]^\alpha_D \neq \emptyset$.

2. 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 3.4 – Illustration des différentes fonctions d'agrégation

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* μ *aux variables de* Q'. *Par définition,* $tv(\mu, Q) = aggreg(tv(\mu(t_1)), \cdots, tv(\mu(t_n))) \geq \alpha$ *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 spécifié dans sa requête. Dans la suite, nous présentons le cache utilisé par l'algorithme αLBA qui permet de réduire le nombre de requêtes exécutées par cet algorithme.

3.3.4 Optimisation de l'approche αLBA (Extended Cache)

Nous avons mis en place une technique pour optimiser αLBA. Elle se base sur un cache qui permet la réduction du nombre de requêtes exécutées par l'algorithme αLBA dans le but de de trouver l'ensemble des αMFS et αXSS. Dans sa version originale, l'algorithme LBA maintient un cache des requêtes exécutées annotées avec leurs résultats : réussite ou échec [FJHB17]. Avant d'exécuter une sous-requête, cet algorithme vérifie d'abord s'il s'agit d'une sous-requête (resp., 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.

3.4 Évaluation expérimentale

Afin de montrer l'intérêt de notre approche (αLBA) présentée précédemment, nous présentons, dans cette section, des expérimentations comparant les performances de notre approche avec celle de l'approche de base DFS (DFS exécute toutes les requêtes du treillis).

3.4.1 Environnement expérimental

Nous avons développé nos deux algorithmes DFS et αLBA avec JAVA 1.8 64 bits. Ces algorithmes prennent en entrées une requête qui échoue et un seuil α, et renvoient les ensembles de αMFS et αXSS de Q pour le seuil donné. 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 en effet 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.

3.4.2 Données et requêtes utilisées

Nous avons utilisé un jeu de données de 20 millions de triplets générés par le banc d'essai WatDiv [AHÖD14]. 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 3.1. Ces requêtes contiennent entre 1 et 15 patrons de triplets 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.

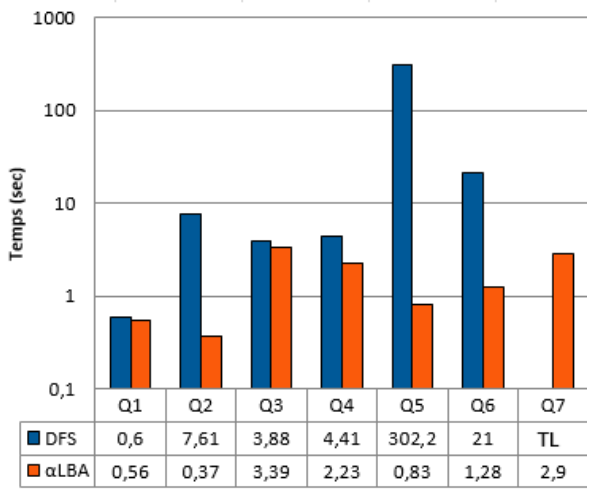
— *Requêtes en chaîne* : elles sont caractérisées par une jointure objet-sujet. La variable de jointure est présente comme objet dans un patron de triplet de la requête et comme sujet dans un autre.

— *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.

Cette expérimentation évalue le temps d'exécution de DFS et αLBA pour un seuil arbitrairement fixé à $0,2$ et un jeu de données de 20M triplets.

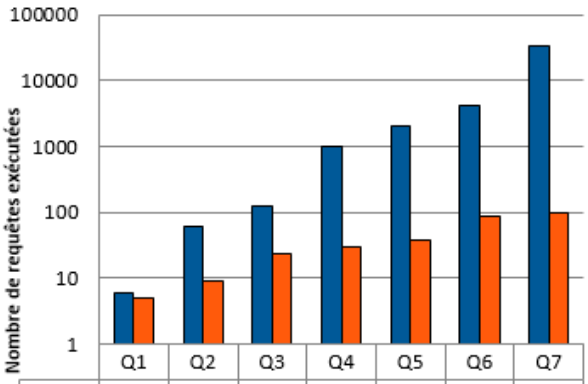
Q1 (3TP*)	SELECT * WHERE { ?p friendOf ?f . ?f likes ?p . ?p type ProductCategory }
Q2 (6TP)	SELECT * WHERE { User666524 likes ?v0 . ?v0 hasGenre ?v1 . ?v1 tag Topic129 . ?v0 friendOf ?v2 . ?v2 Location ?v3 . ?v3 parentCountry Country17 }
Q3 (7TP)	SELECT * WHERE { ?v0 follows ?v1 . ?v1 follows ?v0 . ?v1 subscribes ?v2 . ?v0 subscribes ?v2 . ?v1 likes Product16770 . ?v0 nationality Country20 . ?v0 makesPurchase ?v3 }
Q4 (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' }
Q5 (11TP)	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 }
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 }

TABLEAU 3.1 – Requêtes utilisées pour l'évaluation expérimentale (en format simplifié)
* nombre de patrons de triplets



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
DFS	0,6	7,61	3,88	4,41	302,2	21	TL
αLBA	0,56	0,37	3,39	2,23	0,83	1,28	2,9

FIGURE 3.5 – Temps d'exécution (20M triplets, implémentation avec Jena TDB Quad Filter)



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
DFS	6	62	126	1022	2046	4094	32766
αLBA	5	9	24	30	38	88	99

FIGURE 3.6 – # Requêtes exécutées (20M triplets, implémentation avec Jena TDB Quad Filter)

3.4.3 Résultats expérimentaux

Les figures 3.5 et 3.6 montrent respectivement le temps d'exécution et le nombre de sous-requêtes exécutées par chaque algorithme pour chaque requête du banc d'essai. Ces graphiques sont affichés en échelle algorithmique pour plus de lisibilité.

Cette expérimentation montre les gains apportés par notre algorithme αLBA par rapport à la méthode de base DFS. Notre algorithme exécute moins de requêtes pour trouver les αMFS et les αXSS. Par conséquent, il présente des temps d'exécutions inférieurs, ce qui correspond à une diminution conséquente par rapport à DFS lorsque la taille de la requête augmente. Par exemple, DFS a besoin de 4 094 requêtes pour trouver les αMFS et les αXSS de la requête Q6, alors que notre algorithme effectue ce calcul en exécutant 88 requêtes.

Un algorithme qui évalue toutes les sous-requêtes tel que DFS, peut être utilisé pour les requêtes contenant seulement quelques partons de triplets. Pour des requêtes plus grandes, le nombre de sous-requêtes augmente de manière exponentielle et, par conséquent, les performances de DFS diminuent rapidement et le temps d'exécution devient rédhibitoire. Dans ce cas, l'exploration intelligente des sous-requêtes fournie par notre algorithme αLBA est nettement plus efficace. Par exemple le temps d'exécution de DFS pour la requête Q7 dépasse le temps d'exécution maximal [3], car DFS est amené à exécuter les 32 766 sous-requêtes de Q7.

Nous remarquons que les temps d'exécution de LBA ne sont pas proportionnels à la taille des requêtes, de même pour DFS. Ceci s'explique par la nature des sous-requêtes exécutées par ces algorithmes. Dans le cas des sous-requête en forme de chaîne, la suppression de patrons de triplets peut mener à des produits cartésiens. En conséquence, les temps d'exécution sont plus élevée que ceux d'une requête plus grande mais uniquement basée sur des jointures.

3.5 Conclusion

Dans ce chapitre nous avons présenté notre première contribution, à savoir, l'introduction des αMFS et αXSS. Nous avons défini les conditions pour lesquelles l'algorithme LBA de Fokou et al. [FJHB17] peut être directement adapté pour trouver les αMFS et αXSS d'une requête

3. noté TL (time limitation), c'est à dire que l'algorithme a dépassé le temps d'exécution maximal que nous avons fixé, à savoir 12 heures.

RDF incertaine défaillante (αLBA). Grâce à nos expérimentations, nous avons montré que notre approche αLBA est meilleure que l'approche de base DFS. 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.

Ainsi, nous nous intéressons dans le chapitre suivant 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 triplets, les retours sur des seuils multiples peuvent permettre à l'utilisateur de réévaluer le seuil de confiance associé à sa requête.

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. Ainsi, en fonction de l'ordre dans lequel les valeurs α sont considérées, trois approches sont proposées : *ascendante*, *descendante* et *hybride*.

Chapitre 4

Découverte de αMFS et αXSS multi-seuils pour le problème des réponses vides

Sommaire

4.1 Introduction

Lorsqu'un utilisateur interroge une base de connaissances incertaine, celui-ci s'attend à obtenir des résultats de qualité, c'est à dire des résultats dont le degré de confiance est supérieur à un seuil donné, noté α. Cependant, la structure et le contenu d'une base de connaissances ne sont que rarement connus au préalable, l'utilisateur peut donc être confronté à un problème de réponse vide, où aucun résultat n'est retourné, ou bien seuls des résultats avec un degré de confiance inférieur au seuil α renseigné. Au lieu de renvoyer uniquement l'ensemble vide de résultats, nous avons proposé l'algorithme αLBA dans le chapitre précédent. Le système peut ainsi aider l'utilisateur à comprendre les raisons de cet échec, en lui fournissant un ensemble de sous-requêtes minimales causant l'échec pour un degré α (αMFS). En plus des informations basées sur les αMFS, d'autres requêtes, appelées MaXimal Succeeding Subqueries (αXSS), sont suggérées à l'utilisateur par le système. Ce sont les sous-requêtes qui réussissent et qui sont constituées d'un nombre maximal de patrons de triplets par rapport à la requête initiale. L'utilisateur peut exécuter ces XSS pour trouver des réponses alternatives à sa requête initiale en échec.

Dans l'approche αLBA proposée dans le chapitre précédent, l'utilisateur doit fournir le paramètre α. Cependant, comme celui-ci peut ne pas avoir une idée précise du degré de confiance des informations présentes dans la BC incertaine cible, nous souhaitons lui suggérer des requêtes assouplies, avec des seuils α inférieurs au degré initial. Ce type de relaxation nécessite le calcul de αMFS et de αXSS pour un ensemble de seuils α. Pour optimiser le temps de calcul, certaines propriétés entre les αMFS et les αXSS correspondant à différents seuils sont établies et exploitées.

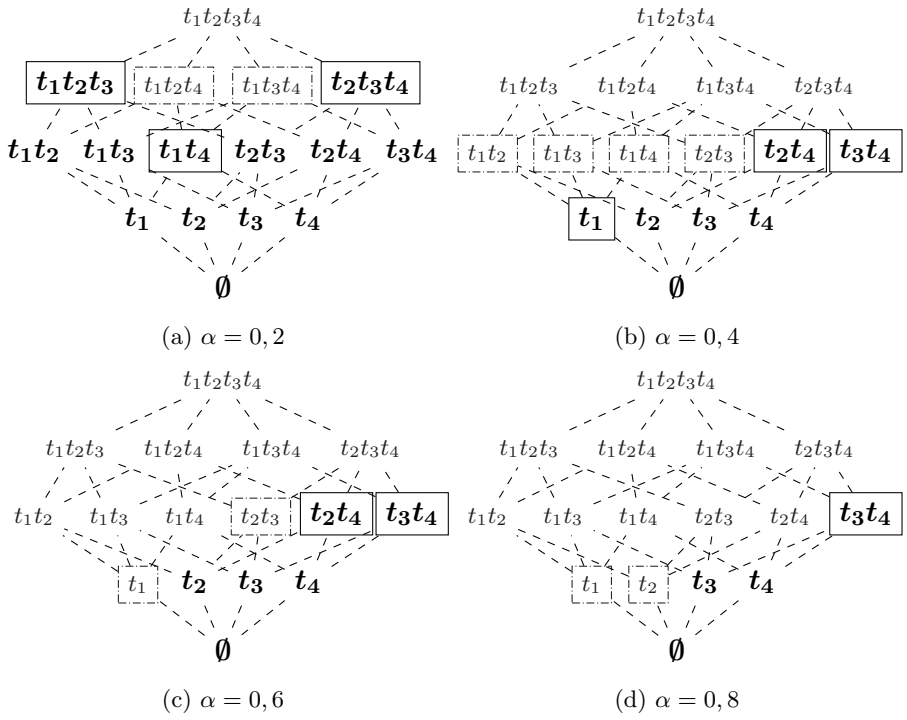
4.2 Motivation de la problématique

Considérons la requête Q du chapitre précédent en substituant l'auteur. L'utilisateur recherche cette fois les `Livres` édités par `Springer` et écrits par `Abraham Lincoln` avec leurs nombres de pages. Nous notons cette requête $t_1t_2t_3t_4$.

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

La figure 4.1 décrit les αMFS et αXSS de la requête Q pour quatre seuils de confiance sur le treillis des sous-requêtes de Q. Nous considérons d'abord que l'utilisateur attend des résultats avec un degré de confiance d'au moins 0,8. Dans notre exemple, cette requête échoue, mais, grâce aux $0,8$MFS ($mfs^{0,8}(Q) = \{t_1, t_2\}$) et $0,8$XSS ($xss^{0,8}(Q) = \{t_3t_4\}$), nous fournissons à l'utilisateur les explications suivantes : (1) pour un degré de confiance égal à 0,8, il n'existe aucun élément écrit par `Abraham Lincoln` (t_1), ni d'élément édité par `Springer` (t_2), ce qui signifie que toute requête incluant un de ces deux patrons de triplets échouera pour ce seuil, (2) mais il existe des `Livres` avec leurs nombres de pages (t_3t_4), ce qui constitue la seule requête alternative renvoyant des résultats tout en conservant un maximum de patrons de triplets de la requête initiale.

Si nous calculons les αMFS et αXSS, pour des degrés de confiance inférieurs (par exemple, 0,6, 0,4 et 0,2), nous pouvons trouver des explications supplémentaires et significatives pour l'utilisateur. Par exemple, les αXSS peuvent être utilisés pour fournir les explications suivantes :



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

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

FIGURE 4.1 – Treillis de sous-requêtes de Q pour différents α

— $xss^{0,6}(Q) = \{t_2t_4, t_3t_4\}$, l'utilisateur peut trouver dans la BC incertaine des éléments édités par `Springer` avec leurs nombres de pages (t_2t_4) s'il accepte de réduire son seuil de confiance à 0,6, la αXSS (t_3t_4) précédemment identifié pour le degré 0,8 reste une αXSS pour 0,6 ;

— $xss^{0,4}(Q) = \{t_1, t_2t_4, t_3t_4\}$, en plus des αXSS précédentes, il existe des objets écrits par `Abraham Lincoln` (t_1) avec un degré de confiance égal à 0,4 ;

— $xss^{0,2}(Q) = \{t_1t_2t_3, t_1t_4, t_2t_3t_4\}$, ce qui signifie qu'il existe des `livres` édités par `Springer` et écrits par `Abraham Lincoln` ($t_1t_2t_3$), des éléments écrits par `Abraham Lincoln` avec leur `nombre de pages` (t_1t_4), et des `livres` édités par `Springer` avec leur `nombre de pages` ($t_2t_3t_4$) pour un degré de confiance égal à 0,2.

En ce qui concerne les $0,6$MFS, $mfs^{0,6}(Q) = \{t_1, t_2t_3\}$, signifie par exemple que, pour ce seuil de 0,6, il n'existe aucun élément écrit par `Abraham Lincoln`, et bien qu'il existe des éléments édités par `Springer` (t_2 n'est pas une αMFS pour le degré 0,6), il n'y a pas de `Livres` édités par `Springer` (t_2t_3).

Ces informations peuvent aider l'utilisateur à mieux comprendre le contenu de la base de connaissances, à reformuler sa requête ou à ajuster ses attentes. Pour fournir ce retour, une approche efficace permettant de calculer les αMFS et les αXSS pour un ensemble de seuils est nécessaire.

Énoncé du problème

Nous nous intéressons dans ce chapitre au calcul efficace des $mfs^{\alpha_i}(Q)$ et $xss^{\alpha_i}(Q)$, d'une requête RDF incertaine défaillante Q, pour un ensemble de seuils $\alpha_i \in \{\alpha_1, \cdots, \alpha_n\}$.

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 les prochaines sections, 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, pour laquelle les seuils sont considérés par ordre croissant.

4.3 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. Par exemple, si l'utilisateur n'obtient aucun résultat pour une recherche basée sur un degré minimal des réponse à $\alpha_i = 0,5$, toutes les recherche pour un degré supérieur à 0,5 ne retourneront également aucune réponse. En effet, les réponses pour le degré α_j sont un sous-ensemble des réponses pour le degré α_i. Cependant, cette sous-requête Q^* n'est pas nécessairement minimale pour α_j, et peut donc ne pas être une α_jMFS. La proposition suivante énonce une condition nécessaire pour qu'une α_iMFS soit aussi une α_jMFS.

Propriété 1 *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 2 *Si Q^* est une α_iMFS de Q sur un ensemble de données D, alors $[[Q^*]]_D^{\alpha_i} = \emptyset$. Puisque $\alpha_i < \alpha_j$, le résultat de la requête Q^* pour α_j est un sous-ensemble du résultat de Q^* pour α_i, et comme Q^* retourne un ensemble vise de résultat pour α_i, nous avons aussi $[[Q^*]]_D^{\alpha_j} = \emptyset$. Q^* est minimale ($|Q^*| = 1$) car, par convention, la requête vide réussit, et Q^* échoue pour α_j, ainsi Q^* est une α_jMFS de Q.*

Ainsi, une α_iMFS Q^* de Q, telle que $|Q^*| = 1$ est une α_jMFS de Q. Comme indiqué précédemment, pour qu'une sous-requête Q^* soit une α_jMFS d'une requête Q, toutes ses sous-requêtes doivent réussir. D'après la proposition 1, cette propriété est toujours vraie, si la requête contient un seul patron de triplet. La vérification de cette condition, c'est-à-dire de tester si une requête ne possède qu'un seul patron de triplet, ne nécessite aucun accès à la base de connaissances. Ainsi, nous vérifions d'abord ce cas et mettons toutes les α_jMFS découvertes de Q dans un ensemble de *αMFS découvertes* désignées par $dmfs^{\alpha_j}(Q)$ (*discovered αMFS*).

Nous considérons maintenant le cas général, où $|Q^*| > 1$. 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 2 du chapitre précédent) 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 4. Cette approche évite de recommencer à partir de la requête initiale pour trouver de nouvelles α_jMFS car les super-requêtes des α_iMFS ne sont jamais évaluées. Par conséquent, cette approche exécute moins de sous-requêtes par rapport à la méthode de base NLBA.

Nous avons vu les propriétés qui peuvent être exploitées pour déduire certaines α_jMFS de Q à partir de ses α_iMFS. 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 pour le seuil α_j, elle est aussi une α_jXSS de Q.

Propriété 2 *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* 3** *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 $dxss^{\alpha_j}(Q)$. Si la α_iXSS échoue pour α_j, nous pouvons toujours l'utiliser comme point de départ pour trouver une α_jMFS grâce à l'algorithme *TrouverUneαMFS*. Ainsi, l'exécution d'une α_iXSS pour le nouveau seuil α_j aboutit toujours à la découverte d'une α_jXSS ou d'une α_jMFS : si elle réussit, c'est une α_jXSS ; si elle échoue, nous l'utilisons pour trouver une α_jMFS.

4.3.1 Algorithme

L'algorithme 4 présente notre approche complète pour rechercher des α_jMFS et des α_jXSS à partir de l'ensemble des α_iMFS et α_iXSS. Toutes les α_iMFS qui ont un seul patron de triplet (dites *Atomiques*) sont insérés dans $dmfs^{\alpha_j}(Q)$ (ligne 1). Ensuite, l'algorithme parcourt les α_iMFS avec au moins deux patrons de triplets (l'ensemble *pmfs*) et recherche une α_jMFS Q^* dans une requête Q' de *pmfs* avec l'algorithme *TrouverUneαMFS* (ligne 6). Ensuite, il supprime toutes les requêtes en échec de *pmfs* contenant Q^* (lignes 8-9), car elles ne peuvent pas être minimales. Ce processus s'arrête lorsque toutes les requêtes de *pmfs* ont été traitées (elles ont été utilisées pour trouver une α_jMFS ou supprimées car elles contiennent une α_jMFS).

Nous considérons ensuite toutes les α_iXSS qui ne contiennent pas une α_jMFS découverte (l'ensemble *pxss*). Si une requête Q' de cet ensemble réussit, il s'agit d'une α_jXSS (propriété 2). Sinon, nous l'utilisons pour rechercher une α_jMFS avec *TrouverUneαMFS* (lignes 16-17) et supprimer les requêtes de l'ensemble *pxss* contenant cette α_jMFS découverte (lignes 18-19).

Après avoir découvert des 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 5). Cet algorithme calcule les α_jXSS potentielles (*pxss*) qui ne contiennent aucune α_jMFS (lignes 4-8), supprime de cet ensemble les α_jXSS (ligne 9), puis itère sur l'ensemble *pxss* comme pour la version originale de αLBA (voir l'algorithme 3 du chapitre précédent).

Par ailleurs, les algorithmes présentés dans ce chapitre utilisent également le cache étendu présenté dans la section 3.3.4.

4.3.2 Illustration de l'approche ascendante

Pour illustrer l'approche ascendante, nous considérons l'exemple de la figure 4.1. À partir des $0,6$MFS et $0,6$XSS, nous montrons comment l'algorithme ascendant calcule les $0,8$MFS et $0,8$XSS. Comme t_1 est une $0,6$MFS et ne contient qu'un seul patron de triplet, cette dernière est une $0,8$MFS (propriété 1). La seconde $0,6$MFS est t_2t_3. Comme cette requête échoue nécessairement pour 0,8, nous utilisons l'algorithme $TrouverUne\alpha MFS$ pour identifier la $0,8$MFS t_2. Suite à la découverte de la $0,8$MFS t_2, nous ne considérons plus les $0,6$XSS qui contiennent cette $0,8$MFS t_2. Donc, la $0,6$XSS t_2t_4 n'est pas considéré et seule la $0,6$XSS t_3t_4 est exécutée, et réussit pour 0,8. Ainsi, t_3t_4 est une $0,8$XSS (propriété 2). Les αMFS et αXSS découvertes, données en entrée de l'algorithme OptimizedαLBA, sont respectivement : $dmfs^{\alpha}(Q) = \{t_1, t_2\}$

Algorithm 4: Découvrir des α_jMFS et α_jXSS pour 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 $requêteAtomique \leftarrow \{Q_a \in mfs^{\alpha_i}(Q) \mid |Q_a| = 1\}$;
2 $dmfs^{\alpha_j}(Q) \leftarrow requêteAtomique$;
3 $pmfs \leftarrow mfs^{\alpha_i}(Q) - requêteAtomique$;
4 **while** $pmfs \neq \emptyset$ **do**
5 $Q' \leftarrow pmfs.dequeue()$;
6 $Q^* \leftarrow TrouverUne\alpha MFS(Q', D, \alpha_j)$;
7 $dmfs^{\alpha_j}(Q) \leftarrow dmfs^{\alpha_j}(Q) \cup \{Q^*\}$;
8 **foreach** $Q'' \in pmfs$ *tel que* $Q^* \subseteq Q''$ **do**
9 $pmfs \leftarrow pmfs - \{Q''\}$;
10 $pxss \leftarrow \{Q_a \in xss^{\alpha_i}(Q) \mid \nexists\, Q^* \in dmfs^{\alpha_j}(Q)$ *tel que* $Q^* \subset Q_a\}$;
11 **while** $pxss \neq \emptyset$ **do**
12 $Q' \leftarrow pxss.dequeue()$;
13 **if** $[[Q']]_D^{\alpha_j} \neq \emptyset$ **then** `// Q' est une` α_j`XSS`
14 $dxss^{\alpha_j}(Q) \leftarrow dxss^{\alpha_j}(Q) \cup \{Q'\}$;
15 **else** `// Q' contient une` α_j`MFS`
16 $Q^* \leftarrow TrouverUne\alpha MFS(Q', D, \alpha_j)$;
17 $dmfs^{\alpha_j}(Q) \leftarrow dmfs^{\alpha_j}(Q) \cup \{Q^*\}$;
18 **foreach** $Q'' \in pxss$ *tel que* $Q^* \subseteq Q''$ **do**
19 $pxss \leftarrow pxss - \{Q''\}$;
20 **return** $\{dmfs^{\alpha_j}(Q), dxss^{\alpha_j}(Q)\}$;

Algorithm 5: 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` α`LBA`
11 **return** $\{mfs^{\alpha}(Q), xss^{\alpha}(Q)\}$;

et $dxss^{\alpha}(Q) = \{t_3t_4\}$. À partir de ces ensembles, l'algorithme OptimizedαLBA détermine qu'il n'existe pas de αXSS potentielles (lignes 1-8 du présent algorithme) et ainsi, que toutes les $0,8$MFS et $0,8$XSS ont été trouvées. La figure 4.2 donne un aperçu de cette séquence d'exécution des algorithmes.

L'approche ascendante découvre des αMFS et αXSS (et améliore donc l'exécution de αLBA pour chaque seuil α_i), dans le cas où certaines αXSS restent les mêmes pour différentes valeurs de seuil, ou dans le cas où les αMFS sont atomiques. Dans le cas contraire, l'algorithme utilisera néanmoins les αMFS précédemment découvertes comme points de départ, au lieu de recommencer à partir de zéro, ce qui améliore tout de même le temps d'exécution.

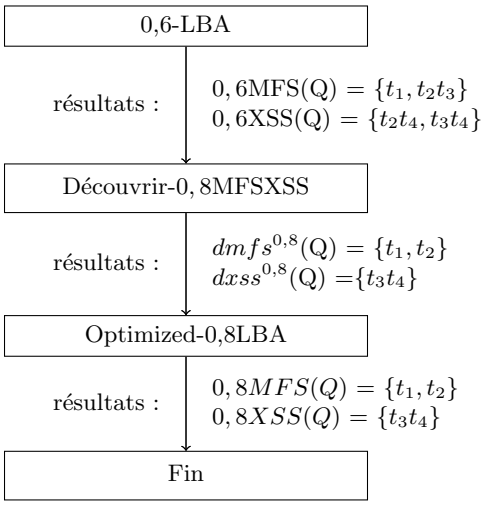


FIGURE 4.2 – Illustration de l'approche ascendante pour deux seuils (0,6 et 0,8)

4.4 Approche descendante

Nous considérons maintenant une approche descendante qui calcule les αMFS et les αXSS avec des valeurs de seuil dans un ordre décroissant. Grâce à la relation de dualité qui existe entre les αMFS et les αXSS, les propriétés utilisées dans cette approche sont similaires à celles utilisées dans l'approche ascendante. Ainsi, nous ne les présentons que de manière informelle. Soit α_i et α_j deux seuils tels que $\alpha_i > \alpha_j$, nous avons les propriétés suivantes :

Propriété 3 *Les α_iMFS qui échouent pour α_j sont des α_jMFS.*

Propriété 4 *Les α_iXSS d'une taille $|Q| - 1$ sont des α_jXSS où Q est la requête initiale, qui échoue par convention.*

Propriété 5 *Les α_iXSS d'une taille $< |Q| - 1$ réussissent aussi pour α_j et sont donc des sous-requêtes d'α_jXSS. Les α_jXSS sont découvertes en utilisant un algorithme appelé TrouverUneαXSS (voir algorithme 6) qui est le dual de l'algorithme TrouverUneαMFS (algorithme 2 du chapitre précédent).*

4.4.1 Algorithme

Algorithm 6: Découverte d'une αXSS pour une requête RDF Q qui réussit

***TrouverUneαXSS*(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' ← Q;
2  foreach patron de triplet t_i ∈ (Q_i − Q) do
3      Q' ← Q' ∧ t_i;
4      if [[Q']]^α_D = ∅ then
5          Q' ← Q' − t_i;
6  return Q';
```

Une fois un ensemble de α_jMFS et de α_jXSS trouvé sur la base des propriétés susmentionnées, l'algorithme Optimized-αLBA est exécuté pour rechercher l'ensemble complet des α_j MFS et des α_jXSS. Cette approche améliorera l'exécution de αLBA pour chaque seuil α, si certaines αMFS restent les mêmes pour des valeurs de seuil différentes, ou si certaines αXSS ont une taille de $|Q| - 1$. Sinon, il utilisera néanmoins les αXSS précédemment découvertes comme point de départ, au lieu de recommencer à partir de la requête d'origine.

4.4.2 Illustration de l'approche descendante

Pour illustrer l'approche descendante, nous considérons encore une fois l'exemple de la figure 4.1. À partir des $0,8$MFS et $0,8$XSS, nous montrons comment l'algorithme ascendant calcule les $0,6$MFS et $0,6$XSS. La $0,8$MFS t_1 échoue aussi pour 0,6, c'est donc une $0,6$MFS. La $0,8$XSS t_3t_4 réussit nécessairement pour 0,6, cette dernière est utilisée comme paramètre de l'algorithme

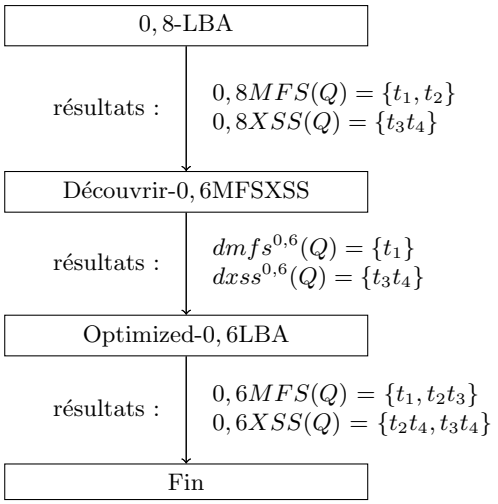


FIGURE 4.3 – Illustration de l'approche descendante pour deux seuils (0,8 et 0,6)

TrouverUneαXSS afin de trouver la $0,6$XSS t_3t_4 . Ainsi, les $0,6$MFS et $0,6$XSS 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 OptimizedαLBA) qui identifie alors les autres $0,6$MFS (t_2t_3) et $0,6$XSS (t_2t_4). La figure 4.3 donne un aperçu de cette séquence d'algorithmes.

Ainsi, l'approche descendante peut être considérée comme le dual de l'approche ascendante. Nous proposons maintenant une approche hybride qui exploite les propriétés de ces deux approches.

4.5 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.

4.5.1 Algorithme

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 inférieurs et supérieurs, et peut aussi bénéficier des propriétés utilisées dans les approches ascendantes et descendantes afin de découvrir des αMFS et αXSS. Nous utilisons la propriété additionnelle suivante pour trouver des αMFS et αXSS supplémentaires.

Propriété 6 *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.*

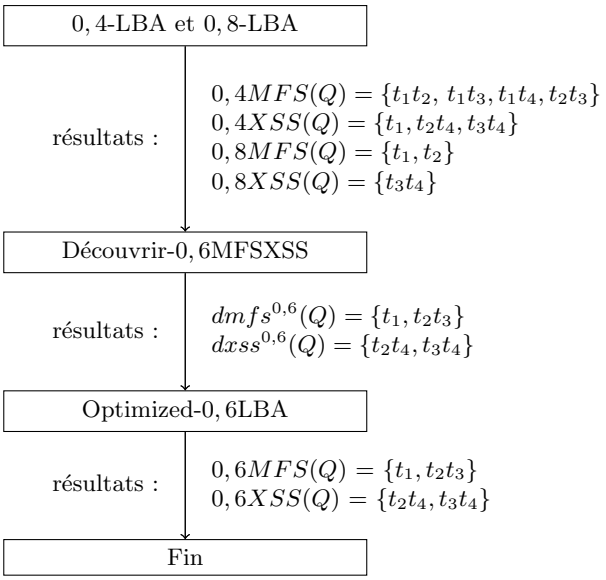


FIGURE 4.4 – Illustration de l'approche hybride pour trois seuils (0,4, 0,8 et 0,6)

***Preuve* 4** *Si Q^* est une α_iMFS de Q, alors Q^* échoue nécessairement pour α_j. De plus, si Q^* est une α_kMFS de Q, alors toutes ses sous-requêtes réussissent pour α_k et donc également pour α_j. Ainsi, nous avons prouvé que Q^* est à la fois une sous-requête qui échoue et qu'elle est minimale pour le degré α_j. Par conséquence Q^* est une α_jMFS. La propriété correspondante aux αXSS est prouvée de manière similaire.*

Ainsi, l'approche hybride permet de découvrir un ensemble de αMFS et de αXSS, en utilisant les propriétés de l'approche ascendante et de l'approche descendante ainsi que la propriété 6. Comme dans les approches précédentes, l'algorithme Optimized-αLBA est exécuté à l'aide des αMFS et αXSS découvertes, pour trouver l'ensemble complet des αMFS et αXSS pour le seuil considéré.

4.5.2 Illustration de l'approche hybride

Nous illustrons le fonctionnement de l'algorithme hybride sur le même exemple (figure 4.1), en calculant les $0,6$MFS et $0,6$XSS, 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 $0,6$XSS (proposition 6). Ensuite, l'algorithme recherche des $0,4$MFS contenant un seul patron de triplet et des $0,8$XSS, contenant trois patrons de triplets ($|Q| - 1$). Comme cet exemple n'en possède pas, l'approche hybride continue en recherchant les $0,8$MFS qui échouent pour 0,6 (propriété de l'approche descendante). C'est le cas pour t_1, qui est une $0,6$MFS. De même, il recherche les $0,4$XSS qui réussissent pour 0,6 (propriété de l'approche ascendante) et trouve la $0,6$XSS t_2t_4. L'algorithme *TrouverUneαMFS* est ensuite utilisé avec les $0,4$MFS. Dans cet exemple, l'algorithme est seulement appliqué à t_2t_3, car les autres contiennent t_1 (la $0,6$MFS). Il trouve que t_2t_3 est une $0,6$MFS. Inversement, il utilise l'algorithme *TrouverUneαXSS* avec la $0,8$XSS. Dans cet exemple, toutes les $0,8$XSS ont déjà été utilisées. Grâce aux propriétés des approches ascendantes et descendantes, l'approche hybride a découvert toutes les $0,6$MFS et $0,6$XSS, sans avoir eu besoin d'exécuter l'algorithme $0,6$LBA. La figure 4.4 donne un aperçu de cet enchaînement.

4.6 Analyse de la complexité des algorithmes proposés

Dans cette section, nous considérons une complexité temporelle de nos algorithmes proportionelle au nombre de requêtes exécutées. Cette analyse ne prend pas en compte les temps de réponses individuels de chaque requête (nous supposons que chaque exécution d'une requête coûte une unité de temps). La complexité de la recherche de toutes les αMFS et αXSS étant exponentielle par rapport au nombre de patrons de triplets de la requête initiale [God97], nous évaluons cette complexité par rapport à la taille de la sortie. Il a été démontré que tout algorithme calculant les αMFS et αXSS en se basant sur la valeur booléenne (réussite ou échec) du résultat de leur exécution doit exécuter au minimum $|mfs^{\alpha}(Q)| + |xss^{\alpha}(Q)|$ requêtes [MT97].

4.6.1 Approche de base NLBA

Nous analysons d'abord la complexité de la méthode de base.

Théorème 1 *Dans le pire cas, αLBA exécute $|xss^{\alpha}(Q)| + |Q| * |mfs^{\alpha}(Q)|$ requêtes.*

***Preuve* 5** *TrouverUneαMFS(algorithme 2) découvre une nouvelle αMFS lors de chaque exécution. TrouverUneαMFS exécute exactement $|Q|$ requêtes (lignes 2 à 4). A chaque itération (ligne 4) de αLBA, cet algorithme recherche (i) une nouvelle αXSS Q' (ligne 7) ou (ii) une nouvelle αMFS Q^{**} (ligne 9). Dans le premier cas, une requête unique est exécutée (ligne 6), alors que dans le second cas, TrouverUneαMFS est appelée avec la requête Q' en entrée. Donc $|Q'| + 1$ requêtes sont exécutées pour trouver une αMFS, avec $|Q'| < |Q|$. Au total, αLBA exécute donc au plus $|xss^{\alpha}(Q)| + |Q| * |mfs^{\alpha}(Q)|$ requêtes.*

Corolaire 1 *Pour un nombre n de seuils α_i, NLBA exécute au plus $\sum_{i=1}^{n} |xss^{\alpha_i}(Q)| + |Q| * |mfs^{\alpha_i}(Q)|$ requêtes.*

4.6.2 Approche ascendante

Dans l'algorithme Optimized-αLBA (algorithme 5), l'exécution de requêtes est effectuée uniquement lors des instructions communes avec l'algorithme initial αLBA (lignes 5 à 13). Par conséquent, la complexité de Optimized-αLBA peut être déduite directement du théorème 1.

Corolaire 2 *Dans le pire cas, l'algorithme Optimized-αLBA exécute $|xss^{\alpha}(Q) - dxss^{\alpha}(Q)| + |Q| * |mfs^{\alpha}(Q) - dmfs^{\alpha}(Q)|$ requêtes.*

Nous montrons tout d'abord que, dans le pire des cas, cet algorithme n'exécute pas plus de requêtes que NLBA.

Lemme 1 *DécouvrirαMFSXSS (algorithme 4) exécute au plus $|dxss^{\alpha_i}(Q)| + |Q| * |dmfs^{\alpha_i}(Q)|$ requêtes.*

***Preuve* 6** *La découverte d'une α_iMFS atomique (lignes 1 à 3) ne nécessite aucune requête sur la base de connaissances. Ensuite, l'algorithme peut être décomposée en deux parties : (i) la première partie concerne les α_{i-1}MFS précédentes (lignes 4 à 9) et (ii) la deuxième partie concerne le test d'échec des α_{i-1}XSS (lignes 15 à 19). Pour la première partie, $|\alpha_{i-1}MFS|$ requêtes sont*

exécutées lors de chaque appel de l'algorithme TrouverUneαMFS (ligne 6). Chacun de ces appels aboutit à la découverte d'une α_iMFS. Pour la deuxième partie, $|\alpha_{i-1}XSS| + 1$ requêtes sont exécutées pour découvrir si une α_{i-1} XSS est en échec (ligne 13), dans ce cas elle fait l'objet de l'entrée de l'algorithme TrouverUneαMFS (ligne 16), qui exécute alors $|\alpha_{i-1}XSS|$ requêtes pour découvrir une nouvelle $\alpha_i MFS$; et nous savons que $|\alpha_{i-1}MFS| < |Q|$ et $|\alpha_{i-1}XSS| + 1 \leq |Q|$. Chaque découverte d'une α_iXSS (lignes 13 à 14) nécessite l'exécution d'une seule requête. Au total, DécouvrirαMFSXSS exécute au plus $|dxss^{\alpha_i}(Q)| + |Q| * |dmfs^{\alpha_i}(Q)|$ requêtes.

L'approche ascendante s'appuie sur l'algorithme DécouvrirαMFSXSS suivi de l'algorithme Optimized-αLBA. La synthèse des résultats du lemme 1 et du corollaire 2 donne directement le théorème suivant.

Théorème 2 *Pour un nombre n de seuils α_i, l'approche ascendante exécute au maximum $\sum_{i=1}^{n} |xss^{\alpha_i}(Q)| + |Q| * |mfs^{\alpha_i}(Q)|$ requêtes.*

Dans le pire des cas, NLBA et l'approche ascendante ont la même complexité. L'utilisation de l'approche ascendante devient utile lorsque certaines αMFS et αXSS peuvent être déduites entre deux seuils successifs α_i. Dans le meilleur des cas, les αMFS et les αXSS restent identiques pour chaque seuil.

Lemme 2 *Dans le meilleur des cas, où les αMFS et les αXSS restent identiques, l'approche ascendante exécute pour chaque seuil α_i après le premier, au plus*

$$\left(\sum_{Q^* \in\, mfs^{\alpha_i}(Q) \wedge |Q^*|>1} |Q^*| \right) + |xss^{\alpha_i}(Q)| \text{ requêtes.}$$

***Preuve* 7** *La découverte de toutes les α_iXSS nécessite l'exécution de chacune d'entre elles, soit $|xss^{\alpha_i}(Q)|$ requêtes. Pour la découverte des α_iMFS ayant un seul patron de triplet (atomiques), aucune requête n'est requise. Pour les autres α_iMFS Q^*, TrouverUneαMFS est appelé avec l'exécution de $|Q^*|$ requêtes, où $|Q^*|$ est la taille de la α_{i-1}MFS considérée, qui est également, dans ce meilleur cas, une α_iMFS.*

Pour un nombre de seuils n, la complexité de l'approche ascendante est directement déduite du lemme précédent.

Proposition 2 *Dans le meilleur des cas, pour un nombre n de seuils α_i où $\forall\, i \in \{2, 3, ..., n\}$, $mfs^{\alpha_i}(Q) = mfs^{\alpha_1}(Q)$, l'approche ascendante exécute, au plus, le nombre de requêtes suivant :*

$$n * |xss^{\alpha_1}(Q)| + |Q| * |mfs^{\alpha_1}(Q)| + (n-1) \left(\sum_{Q^* \in\, mfs^{\alpha_1}(Q) \wedge |Q^*|>1} |Q^*| \right) \text{ requêtes.} \qquad (4.1)$$

Par conséquent, l'approche ascendante devient plus efficace lorsque les αMFS ont un petit nombre de patrons de triplets, dans le but de minimiser le coût associé au terme le plus à droite de l'équation 4.1. Intuitivement, puisqu'une α_{i-1}MFS est la super-requête d'une α_iMFS à découvrir, l'espace de recherche devient plus petit lorsque les α_{i-1}MFS ont un petit nombre de patrons de triplets.

4.6.3 Approche descendante

Nous examinons maintenant la complexité de l'approche descendante. Les deux lemmes suivants sont adaptés directement des résultats obtenus pour l'approche ascendante.

Lemme 3 *L'algorithme TrouverUneαXSS exécute $|Q| - |Q^*|$ requêtes.*

Lemme 4 *Au cours de l'approche descendante, l'algorithme dual de DécouvrirαMFSXSS exécute au maximum $|dmfs^{\alpha_i}(Q)| + |Q| * |dxss^{\alpha_i}(Q)|$ requêtes.*

L'approche descendante s'appuie sur l'algorithme dual de DécouvrirαMFSXSS, suivi de Optimized-αLBA. En utilisant les résultats du lemme 4 et du corollaire 2 le théorème suivant est directement déduit.

Théorème 3 *Pour chaque seuil α_i, l'approche descendante exécute au plus $(|Q| - 1)\,(|dxss^{\alpha_i}(Q)| - |dmfs^{\alpha_i}(Q)|) + |xss^{\alpha_i}(Q)| + |Q| * |mfs^{\alpha_i}(Q)|$ requêtes.*

Notons que l'augmentation du nombre des αMFS découvertes lors de la procédure descendante ($|dmfs^{\alpha}(Q)|$) réduit cette complexité. En particulier, si toutes les αMFS et αXSS sont découvertes ($dmfs^{\alpha}(Q) = mfs^{\alpha}(Q)$ et $dxss^{\alpha}(Q) = xss^{\alpha}(Q)$), la complexité totale devient $|Q| * |xss^{\alpha}(Q)| + |mfs^{\alpha}(Q)|$, qui est le dual de l'approche ascendante. Dans le cas contraire, si les αMFS et αXSS ne sont pas découvertes ($dmfs^{\alpha}(Q) = \emptyset$ et $dxss^{\alpha}(Q) = \emptyset$), la complexité totale devient $|xss^{\alpha}(Q)| + |Q| * |mfs^{\alpha}(Q)|$, c'est-à-dire identique à celle de l'algorithme αLBA.

Lemme 5 *Dans le meilleur des cas, où les αMFS et αXSS restent les mêmes pour chaque seuil, pour chaque seuil α_i sauf pour le premier, l'approche descendante exécute au plus*

$$\left(\sum_{\substack{Q^* \in xss^{\alpha_1}(Q) \\ \wedge |Q^*| < |Q| - 1}} |Q| - |Q^*| \right) + |mfs^{\alpha_i}(Q)| \text{ requêtes.}$$

***Preuve* 8** *La preuve est le dual du lemme 2 de l'approche ascendante.*

Pour un nombre n de seuils, la complexité de l'approche descendante est directement déduite du lemme précédent.

Proposition 3 *Dans le meilleur des cas, pour n seuils α_i où $\forall\, i \in \{2, 3, ..., n\}$, $xss^{\alpha_i}(Q) = xss^{\alpha_1}(Q)$, l'approche descendante exécute au maximum le nombre de requêtes suivant :*

$$|xss^{\alpha_1}(Q)| + (|Q| + n - 1) * |mfs^{\alpha_1}(Q)| + (n - 1) \left(\sum_{\substack{Q^* \in xss^{\alpha_1}(Q) \\ \wedge |Q^*| < |Q| - 1}} |Q| - |Q^*| \right) \tag{4.2}$$

Par conséquent, l'approche descendante devient plus efficace lorsque les αXSS ont un grand nombre de patrons de triplets, dans le but de minimiser le coût associé au terme le plus à droite de l'équation 4.2. Intuitivement, puisqu'une α_{i-1}XSS est une sous-requête d'une α_i XSS à découvrir, l'espace de recherche devient plus petit lorsque les α_{i-1}XSS ont un grand nombre de patrons de triplets.

4.6.4 Approche hybride

Nous considérons maintenant la complexité de l'approche hybride. Étant donné que les αMFS et les αXSS peuvent être découvertes à partir de seuils inférieurs et supérieurs, avec un nombre distinct de requêtes exécutées, le lemme suivant identifie $dmfs^{\alpha}_{inf}(Q)$ comme des αMFS découvertes à partir d'un seuil inférieur d'une manière similaire à l'approche ascendante. $dxss^{\alpha}_{inf}(Q)$,$dmfs^{\alpha}_{sup}(Q)$, $dxss^{\alpha}_{sup}(Q)$ sont définis de manière similaire en fonction de leur type (αMFS, αXSS) et de leur origine (seuil inférieur : inf, seuil supérieur : sup). Le lemme suivant est issu des résultats des lemmes 1 et 4.

Lemme 6 *Le processus de découverte des αMFS et des αXSS au cours de l'approche hybride exécute au plus* $|Q| * |dmfs^{\alpha_i}_{inf}(Q)| + |dxss^{\alpha_i}_{inf}(Q)| + |dmfs^{\alpha_i}_{sup}(Q)| + |Q| * |dxss^{\alpha_i}_{sup}(Q)|$ *requêtes.*

Notons que $dmfs^{\alpha_i}_{inf}(Q) \cap dmfs^{\alpha_i}_{sup}(Q) = \emptyset$ et $dxss^{\alpha_i}_{inf}(Q) \cap dxss^{\alpha_i}_{sup}(Q) = \emptyset$.

En additionnant les résultats du lemme 6 et du corollaire 2, nous obtenons directement le théorème suivant.

Théorème 4 *Pour chaque seuil α_i, l'approche hybride exécute au plus*

$$(|Q| - 1)\left(|dxss^{\alpha_i}_{sup}(Q)| - |dmfs^{\alpha_i}_{sup}(Q)|\right) + |xss^{\alpha_i}(Q)| + |Q| * |mfs^{\alpha_i}(Q)| \text{ requêtes.}$$

Dans le cas le plus défavorable, cette complexité est la même que celle utilisée dans l'approche descendante (théorème 3). Intuitivement, la partie de l'approche ascendante est couverte par Optimized-LBA, comme le souligne le théorème 2. En ce qui concerne l'approche descendante, cette complexité devient la même que celle de αLBA, lorsque aucune αMFS ou αXSS n'est découverte.

Dans le meilleur des cas, la complexité de l'approche hybride est identique à celle de l'approche ascendante (théorème 2) pour ses deux premiers seuils (le seuil le plus bas est suivi du plus élevé lors de cette évaluation). Une fois que ces mfs^{α_1}, xss^{α_1}, mfs^{α_n} et xss^{α_n} sont découvertes, l'algorithme n'exécute plus de requêtes pour déterminer les αMFS et les αXSS de tous les autres seuils (proposition 6).

Corolaire 3 *Dans le meilleur cas, pour un nombre n de seuils α_i où $\forall\, i \in \{2, 3, ..., n\}$, $mfs^{\alpha_i}(Q) = mfs^{\alpha_1}(Q)$, l'approche hybride exécute au plus, le nombre de requêtes suivant :*

$$2 * |xss^{\alpha_1}(Q)| + |Q| * |mfs^{\alpha_1}(Q)| + \left(\sum_{Q^* \in mfs^{\alpha_1}(Q) \wedge |Q^*|>1} |Q^*|\right) \text{ requêtes.}$$

Dans le meilleur cas, l'approche hybride est la seule approche dont la complexité est indépendante du nombre de seuils, ce qui peut être particulièrement utile si de nombreux seuils doivent être considérés.

4.7 Évaluation expérimentale

Dans cette section, nous présentons les implémentations des algorithmes définis dans ce chapitre, que nous comparons ensuite expérimentalement à la méthode de base NLBA. Nous étudions également les capacités de passage à l'échelle de nos propositions.

Algorithmes

Nous avons implémenté nos algorithmes descendants, ascendants et hybrides, ainsi que la méthode de base NLBA, avec Oracle Java 1.8 64 bits. Ces algorithmes prennent en entrée une requête RDF défaillante et un ensemble de seuils. Ils renvoient les ensembles des αMFS et αXSS de cette requête pour chaque seuil. Dans l'implémentation actuelle, ces algorithmes peuvent être exécutés sur Jena TDB et Virtuoso. Notre implémentation est disponible sur `https://forge.lias-lab.fr/projects/qars4ukb` avec un tutoriel pour reproduire nos expérimentations.

Configuration expérimentale

Nos expérimentations ont été menées sur un système Ubuntu Server 16.04 LTS avec un processeur Intel XEON E5-2630 v3 @ 2,4 GHz, équipé de 16 Go de RAM. Pour nos expérimentations, nous utilisons arbitrairement la fonction d'agrégation (aggreg) *min*, comme nous l'avons précisé dans le chapitre précédant (section 3.3.3). Les résultats présentés sont la moyenne de cinq exécutions consécutives des algorithmes. Pour éviter un effet de démarrage à froid, une exécution préliminaire est effectuée mais n'est pas incluse dans les résultats.

Jeu de données et requêtes

Nous avons utilisé six jeux de données RDF incertains avec un nombre de triplets de 20K, 100K, 20M, 40M, 60M et 80M générés avec le banc d'essai Waterloo SPARQL Diversity (WatDiv) [AHÖD14] (le jeu de données 20K est un sous-ensemble du 100K, etc.). Le degré de confiance de chaque triplet RDF a été généré aléatoirement avec une distribution uniforme sur $[0, 1]$. Nous considérons les mêmes 7 requêtes du banc d'essais WatDiv utilisées dans le chapitre précédent (voir chapitre 3, tableau 3.1).

Implémentation Quadstore

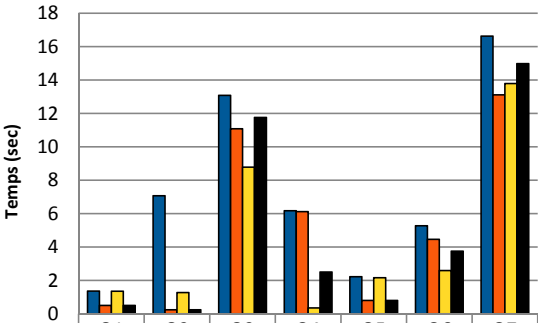
Le stockage et l'interrogation des triplets RDF sont généralement délégués à un TriplesStore tel que Jena TDB ou Virtuoso. Dans ce chapitre nous avons envisagé différentes implémentations, au-delà du simple choix du TriplesStore sous-jacent, pour gérer des triplets RDF incertains et des requêtes avec seuil de confiance :

— *implémentation avec Quad Filter*. Cette implémentation est spécifique à Jena TDB. Nous utilisons la *technique du graphe nommé* [GC14] (également appelée *technique des N-quads*) pour représenter les triplets avec leur degré de confiance (Quads), ainsi Jena TDB fournit un filtre de bas niveau que nous utilisons, pour ne considérer que les résultats satisfaisant le seuil fourni [1];

— *implémentation avec les graphes nommés*. Comme dans l'implémentation précédente, nous utilisons la technique des graphes nommés, pour gérer les triplets RDF incertains, mais nous récupérons les résultats satisfaisants le seuil fourni en interrogeant un ensemble de graphes nommés (l'implémentation de cette technique est détaillée dans la section 4.7.4);

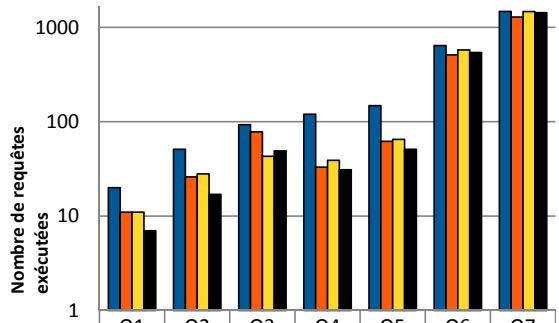
— *implémentation avec la réification*. La réification est une méthode standard pour inclure la confiance, en décomposant le quadruple obtenu en quatre triplets RDF. Les requêtes de seuil doivent ensuite être réécrites pour permettre la réification. L'implémentation de cette technique est détaillée dans la section 4.7.5.

1. `http://jena.apache.org/documentation/tdb/quadfilter.html`



	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

FIGURE 4.5 – Temps d'exécution (20M triplets, implémentation avec Jena TDB Quad Filter)



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
NLBA	20	51	93	120	148	641	1475
Ascendant	11	26	78	33	62	510	1286
Descendant	11	28	43	39	65	576	1470
Hybride	7	17	49	31	51	541	1431

FIGURE 4.6 – # Requêtes exécutées (20M triplets, implémentation avec Jena TDB Quad Filter)

Tandis que l'implémentation avec Quad Filter est spécifique à Jena TDB, les implémentations avec graphe nommé et réification peuvent être définies sur n'importe quel TriplesStore. Cependant, nos résultats expérimentaux montrent que les temps d'exécution des requêtes sont nettement plus longs sur les implémentations avec graphes nommées et réification par rapport à l'implémentation avec Quad Filter. Ainsi, nos expérimentations sur de grands ensembles de données (Sections 4.7.1, 4.7.2 et 4.7.3) sont exécutées à l'aide de Jena TDB Quad Filter. Dans les sections 4.7.4 et 4.7.5, nous verrons, sur de plus petits ensembles de données, les performances des implémentations avec graphe nommé et réification.

4.7.1 Comparaison de performance des algorithmes

Description de l'expérimentation

Dans cette expérimentation, nous avons évalué les performances de nos algorithmes ascendants, descendants et hybrides par rapport à la méthode de base NLBA. Cette expérimentation a été exécutée avec les seuils de confiance définis arbitrairement à $\{0,2, 0,4, 0,6, 0,8\}$ sur Jena TDB Quad Filter avec le jeu de données de 20 millions de triplets.

Résultats et discussion

La figure 4.5 indique le temps d'exécution de chaque algorithme pour chaque requête de charge de travail. La figure 4.6 donne le nombre de requêtes exécutées par chaque algorithme. Cette expérimentation montre l'amélioration de nos algorithmes par rapport à la méthode de base NLBA. En comparaison avec NLBA, nos algorithmes exécutent moins de requêtes pour trouver les αMFS et les αXSS de chaque requête. Globalement, les processus ascendant, descendant et hybride exécutent respectivement 39%, 40% et 39% moins de requêtes que NLBA. En conséquence, ces algorithmes ont des temps d'exécution plus courts (une diminution des temps d'exécution respectifs de 30%, 42% et 33%). Pour certaines requêtes, cette amélioration est importante. Par exemple, NLBA a besoin de 7 secondes pour rechercher les αMFS et les αXSS de la requête Q2, alors que nos algorithmes réalisent ce traitement en environ 1 seconde.

La différence de temps d'exécution dépend fortement des requêtes que nos algorithmes évitent d'exécuter. Par exemple, nos algorithmes exécutent entre 30 et 40 requêtes pour la requête Q4

alors que NLBA nécessite 120 requêtes. Pour les algorithme descendant et hybride, cela se traduit par 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 uniquement l'exécution de requêtes dont les temps d'exécution sont courts, et exécute toujours les requêtes les plus coûteuses. Ainsi, le temps d'exécution total reste pratiquement inchangé.

Cette expérimentation montre également que, bien que nos trois approches soient toujours avantageuses par rapport à NLBA, aucun des trois algorithmes n'est systématiquement meilleur. Les algorithmes ascendant et hybride ont les meilleurs temps d'exécution pour Q1, Q2 et Q5, tandis que l'algorithme descendant est le meilleur pour Q3, Q4 et Q6. Malgré l'exécution du plus petit nombre de requêtes, l'algorithme ascendant a le pire temps d'exécution pour Q6. Inversement, l'algorithme descendant exécute le plus grand nombre de requêtes mais a le meilleur temps d'exécution pour Q4 et Q6. Cela est dû au fait que nos algorithmes exécutent différentes requêtes ayant des temps d'exécution distincts. En particulier, l'algorithme descendant (de haut en bas) commence par rechercher les αMFS et les αXSS pour les seuils les plus élevés. Les requêtes exécutées ont tendance à être sélectives car leur seuil est élevé ; elles ont donc des temps d'exécution courts. Une fois que les αMFS et les αXSS des seuils les plus élevés ont été identifiées, elles permettent d'éviter l'exécution de requêtes avec un seuil inférieur, susceptibles d'être plus coûteuses. Comme l'algorithme ascendant suit l'approche duale, il a tendance à exécuter des requêtes non sélectives et présente la pire performance globale. Dans cette expérimentation, l'algorithme hybride n'est jamais aussi performant que l'ascendant et le descendant. Étant donné que les degrés de confiance ont été générés de manière aléatoire et que les quatre seuils considérés sont séparés par une marge importante, les requêtes partagent peu de αMFS et de αXSS entre les différents seuils. Ainsi, la propriété spécifique utilisée par hybride (voir la section 4.5) est rarement exploitée.

4.7.2 Passage à l'échelle

Description de l'expérimentation

La deuxième expérimentation consiste à évaluer l'évolution des performances des algorithmes lorsque la taille du jeu de données augmente. Cette expérimentation a été exécutée en utilisant la requête Q2 avec les mêmes paramètres que l'expérimentation précédente et les jeux de données 20M, 40M, 60M et 80M.

Résultats et discussion

La figure 4.7 et le tableau 4.1 présentent le temps d'exécution des algorithmes pour cette deuxième expérimentation. Les temps d'exécution de nos algorithmes n'augmentent pas de manière significative entre les jeux de données 40M et 80M. Sur ces jeux de données, nous avons observé que les αMFS et les αXSS de Q2 restent les mêmes et que les mêmes requêtes sont exécutées (environ 25 requêtes pour nos algorithmes et 46 pour NLBA). En conséquence, dans ce cas, le passage à l'échelle des algorithmes ne dépend que du temps d'exécution de ces requêtes.

Pour le jeu de données 20M, Q2 a une αMFS supplémentaire pour tous les seuils. Le nombre de αXSS est le même, mais elles ont moins de patrons de triplets. En conséquence, les algorithmes exécutent des requêtes différentes. Cela a un impact direct sur les performances des algorithmes. 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 davantage de requêtes (28 requêtes sur 20M et 24 requêtes sur les autres). Ainsi, lorsque les αMFS et les

	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

TABLEAU 4.1 – Temps d'exécution vs Taille de jeu de données

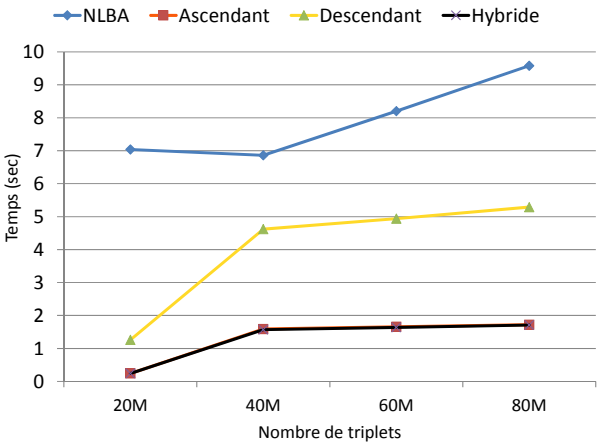


FIGURE 4.7 – Temps d'exécution vs Taille des données (Seuils $\{0, 2, 0, 4, 0, 6, 0, 8\}$, implémentation avec Jena TDB Quad Filter)

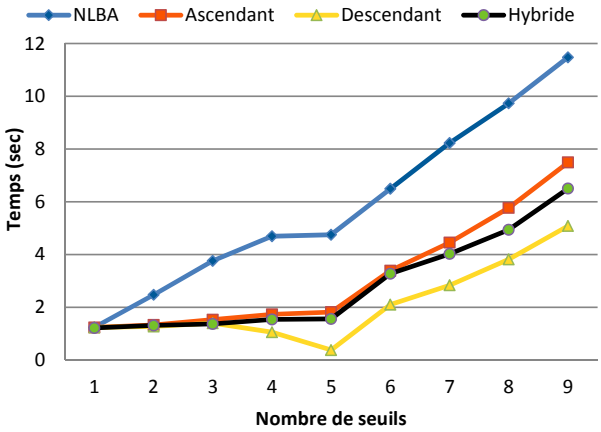


FIGURE 4.8 – Temps d'exécution vs Nombre de seuils α (20M triplets, implémentation avec Jena TDB Quad Filter)

αXSS changent, le passage à l'échelle des algorithmes dépend également des requêtes exécutées et de leurs temps de réponse respectifs.

4.7.3 Évolution du nombre de seuils

Description de l'expérimentation

Dans cette expérimentation, nous évaluons l'évolution des performances des algorithmes lorsque le nombre de seuils augmente. Cette expérimentation a été menée avec la requête Q6 avec 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 jusqu'à neuf seuils.

Résultats et discussion

Les résultats de cette expérimentation sont décrits dans la figure 4.8. Comme le montre cette expérimentation, nos algorithmes s'exécutent plus rapidement que NLBA dès que deux seuils sont pris en compte. Cela est dû au fait que, pour chaque nouveau seuil, NLBA exécute la version originale de l'algorithme αLBA alors que nos algorithmes s'appuient sur les αMFS et αXSS découvertes. En conséquence, NLBA évolue presque linéairement avec le nombre de seuils. En comparaison, la performance de nos algorithmes dépend du nombre de αMFS et de αXSS découvertes. Si les αMFS et les αXSS restent similaires, nos algorithmes ne nécessitent que quelques millisecondes pour rechercher les αMFS et les αXSS pour chaque nouveau seuil. C'est le cas dans nos expérimentations pour les seuils compris entre $0,1$ et $0,5$. Dans le cas contraire, si les αMFS et les αXSS changent entre des seuils différents (c'est le cas ici entre 0,5 et 1), nos algorithmes évoluent de manière analogue à NLBA, car la version optimisée 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, ce qui peut être surprenant. Ce comportement s'explique par le temps d'exécution de αLBA pour le seuil 0,5. Comme le montrent les résultats de NLBA sur la figure 4.8, αLBA pour le seuil 0,5 s'exécute en quelques millisecondes. Pour rappel, NLBA pour 5 seuils exécute, dans l'ordre, αLBA pour 0,1, puis pour 0,2, ..., puis pour 0,5. Une fois les αMFS et les αXSS trouvées pour 0,5 (elles restent similaires entre 0,1 et 0,5), l'algorithme descendant ne nécessite que quelques millisecondes pour les autres seuils. En comparaison, l'exécution de αLBA pour 0,1 prend plus de temps car les requêtes exécutées sont moins sélectives qu'avec 0,5.

4.7.4 Performances des algorithmes avec la technique des graphes nommés

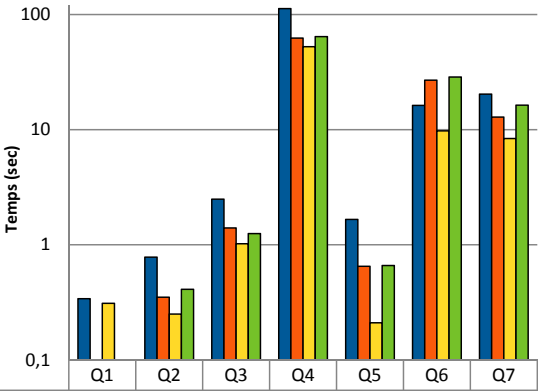
Les expérimentations précédentes ont toutes été exécutées avec l'implémentation basée sur l'implémentation Quad Filter disponible uniquement dans Jena TDB. Comme d'autres TriplesStores, tels que Virtuoso, peuvent être utilisés pour stocker et interroger des bases de données incertaines, nous avons envisagé d'autres manières de prendre en compte l'incertitude. Nous visons deux objectifs : d'une part, montrer que nos approches pouvaient être utilisées sur plusieurs TriplesStores et, d'autre part, vérifier si ces implémentations génériques avaient un impact sur l'intérêt de nos approches. Dans cette section, nous considérons l'implémentation avec la technique des graphes nommés.

Description de l'expérimentation

Dans l'implémentation basée sur les graphes nommés, chaque instance RDF est représentée sous la forme d'un quadruplet *(sujet, prédicat, objet, nom de graphe)*. Le quatrième attribut (nom de graphe) est utilisé pour représenter le degré de confiance pour chaque triplet RDF. Ensuite, les requêtes de seuil sont réécrites pour prendre en compte les graphes nommés. Par exemple, de manière simplifiée, la requête Q introduite dans la section 4.2, est réécrite sous la forme suivante pour $\alpha = 0,8$:

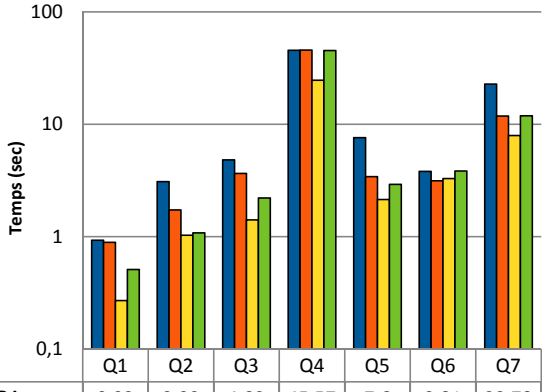
```
SELECT ?b ?p WHERE {
  GRAPH ?g {
    ?b authors "Abraham Lincoln". (t1)
    ?b editor "Springer" .        (t2)
    ?b type Book .                (t3)
    ?b nbPages ?p }               (t4)
  FILTER (?g > 0,8) }
```

Cette expérimentation consiste à évaluer les performances de nos algorithmes ascendants, descendants et hybrides par rapport à l'approche de base NLBA, sur la mise en œuvre du graphe nommé effectuée à la fois sur Jena TDB et Virtuoso. Comme indiqué précédemment, cette implémentation est significativement plus lente que celle du filtre quadratique (Quad Filter). Ainsi, nous utilisons un jeu de données plus petit, composé de 100 000 quadruplets, pour tenir compte de la performance réduite. Comme dans les expérimentations précédentes, nous avons utilisé les seuils $\{0,2, 0,4, 0,6, 0,8\}$.



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
NLBA	0,34	0,78	2,49	112,67	1,66	16,24	20,36
Ascendant	0,1	0,35	1,4	62,4	0,65	26,91	12,85
Descendant	0,31	0,25	1,02	52,57	0,21	9,76	8,37
Hybride	0,1	0,41	1,25	64,26	0,66	28,64	16,31

FIGURE 4.9 – Temps d'exécution (100K triplets, implémentation avec Jena TDB Graphes Nommés)



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
NLBA	0,93	3,09	4,82	45,57	7,6	3,81	22,78
Ascendant	0,89	1,73	3,65	45,72	3,42	3,14	11,83
Descendant	0,27	1,03	1,41	24,65	2,14	3,29	7,94
Hybride	0,51	1,08	2,21	45,3	2,92	3,84	11,9

FIGURE 4.10 – Temps d'exécution (100K triplets, implémentation avec Virtuoso Graphes Nommés)

Résultats et discussion

Les figures 4.9 et 4.10 présentent le temps d'exécution des différents algorithmes pour chaque requête, sur les TriplesStore Jena TDB et Virtuoso. La figure 4.11 donne le nombre de requêtes exécutées par chaque algorithme. Le nombre de requêtes exécutées dépend uniquement de l'algorithme, du jeu de données et de la requête ; l'implémentation choisie (TriplesStore et la représentation des valeur de confiance : graphe nommé, réification) n'a aucun impact sur ce résultat.

Cette expérimentation montre que nos approches surpassent encore la méthode de base NLBA sur l'implémentation basé sur les graphes nommés. En comparaison avec NLBA, nos algorithmes exécutent moins de requêtes pour trouver les αMFS et les αXSS de chaque requête. Globalement, les approches ascendante, descendante et hybride exécutent respectivement 32%, 25% et 28% moins de requêtes que l'approche NLBA. En conséquence, ces algorithmes ont des temps d'exécution plus courts sur Jena TDB (diminution respectivement de 32%, 53% et 27% des temps d'exécution pour les processus ascendant, descendant et hybride) et Virtuoso (diminution respective de temps d'exécution de 20%, 54 % et 23% pour les processus ascendants, descendants et hybrides).

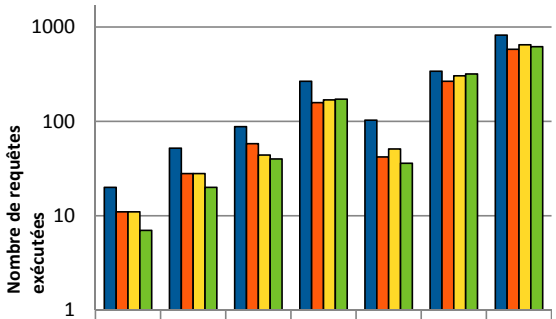
4.7.5 Performances des algorithmes avec la la technique de réification

Une autre manière de représenter des quadruples dans un TriplesStore consiste à utiliser la réification. L'avantage de cette technique est qu'il s'agit d'un standard du W3C [2] et qu'elle peut être utilisée sur n'importe quel Triplestore [GS14]. Cette technique est utilisée dans plusieurs projets [HSBW13, SNB+11, SSST08, SLLP10] pour représenter et interroger les propriétés ajoutées aux triplets RDF telles que la provenance, la confiance, l'heure, l'emplacement, etc.

Résultats et discussion

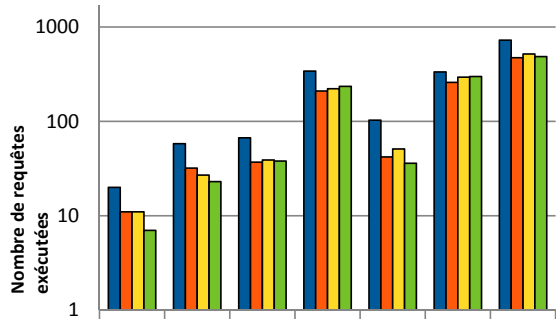
Nous décrivons d'abord brièvement le principe de la mise en oeuvre de la réification. Considérons le triplet $t = (b1, type, livre)$. Pour associer un degré de confiance de 0,4 à ce triplet RDF,

2. https://www.w3.org/wiki/PropertyReificationVocabulary



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
NLBA	20	52	88	266	103	340	820
Ascendant	11	28	58	158	42	266	581
Descendant	11	28	44	169	51	304	648
Hybride	7	20	40	172	36	318	619

FIGURE 4.11 – Requêtes exécutées (100K Triplets, implémentation avec Jena TDB et Virtuoso, Graphes Nommés)



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
NLBA	20	58	67	341	103	334	726
Ascendant	11	32	37	210	42	259	473
Descendant	11	27	39	222	51	294	518
Hybride	7	23	38	235	36	299	486

FIGURE 4.12 – # Requêtes exécutées (20K triplets, implémentation avec Jena TDB et Virtuoso, réification)

nous décrivons cette déclaration (réification) avec les triplets suivants:

```
t rdf:type rdf:statement .
t rdf:subject b1 .
t rdf:predicate type .
t rdf:object book .
t trust "0,4"^^xsd:float
```

Pour interroger l'ensemble de données réifié pour rechercher des résultats avec leur degré de confiance, les requêtes sont réécrites comme suit :

```
SELECT ?book WHERE {
    ?book type book }
threshold = 0,4
```

⇒

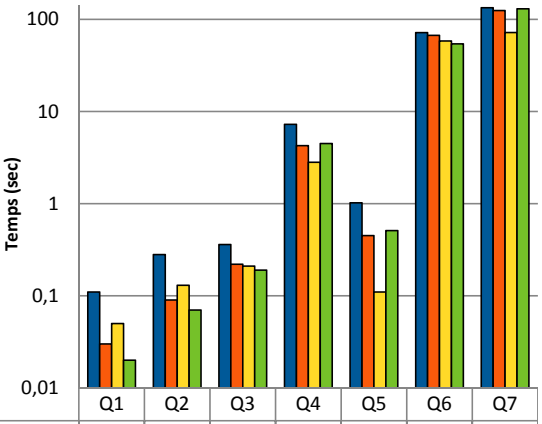
```
SELECT ?book WHERE {
  ?t rdf:subject ?book .
  ?t rdf:predicate type .
  ?t rdf:object book .
  ?t rdf:trust ?trust
  FILTER (?trust > 0,4) }
```

Par rapport à l'implémentation utilisant les graphes nommées, la méthode de réification multiplie par cinq la taille du jeu de données, ainsi que par quatre le nombre de patrons de triplets de la requête. En conséquence, nous considérons un jeu de données de 20 000 triplets pour cette expérimentation afin de conserver des temps d'exécution raisonnables. Cette expérimentation consiste à évaluer les performances de nos algorithmes par rapport à la méthode de base NLBA en utilisant l'implémentation de réification sur Jena TDB et Virtuoso. Comme dans les expérimentations précédentes, nous avons utilisé les seuils $\{0,2, 0,4, 0,6, 0,8\}$.

Résultats et discussion

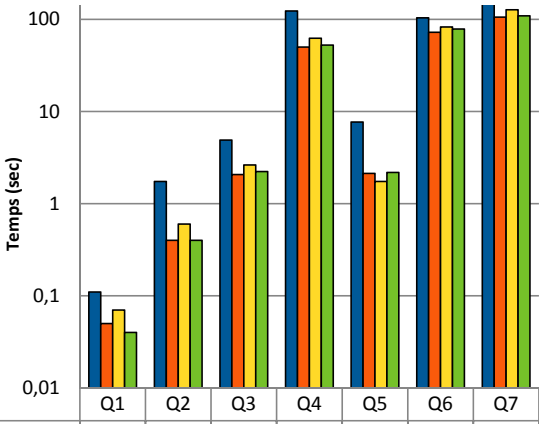
Les chiffres présentés dans les figures 4.13 et 4.14 indiquent le temps d'exécution de chaque algorithme pour chaque requête, respectivement sur Jena TDB et Virtuoso. La figure 4.12 donne le nombre de requêtes exécutées par chaque algorithme.

En ce qui concerne l'implémentation basée sur la réification, nos algorithmes sont encore plus performants que la méthode de base NLBA. Ils exécutent systématiquement moins de requêtes



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
NLBA	0,11	0,28	0,36	7,25	1,02	71,86	133,84
Ascendant	0,03	0,09	0,22	4,26	0,45	67,09	124,61
Descendant	0,05	0,13	0,21	2,81	0,11	58,29	71,97
Hybride	0,02	0,07	0,19	4,49	0,51	54,19	130,21

FIGURE 4.13 – Temps d'exécution (20K triplets, Jena TDB, réification)



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
NLBA	0,11	1,74	4,9	123,27	7,68	103,83	182,37
Ascendanrt	0,05	0,4	2,07	50,07	2,13	72,43	105,69
Descendant	0,07	0,6	2,63	62,39	1,74	82,65	127,11
Hybride	0,04	0,4	2,23	52,58	2,18	78,53	109,13

FIGURE 4.14 – Temps d'exécution (20K triplets, Virtuoso, réification)

pour trouver les αMFS et les αXSS. Globalement, les processus ascendant, descendant et hybride exécutent respectivement 37%, 31% et 33% moins de requêtes que NLBA. En conséquence, ces algorithmes ont des temps d'exécution plus courts sur Jena TDB (avec une diminution de respectivement 11%, 37% et 8% des temps d'exécution pour les processus ascendant, descendant et hybride) et Virtuoso (avec une diminution de respectivement 42%, 34% et 45% de Temps d'exécution pour les processus ascendants, descendants et hybrides).

De par leur conception, nos trois algorithmes exécutent toujours moins de requêtes que la méthode de base pour calculer αMFS et αXSS. Nos expérimentations ont montré que, quelle que soit l'implémentation choisie, ceci entraînait toujours une amélioration significative des performances, avec des temps totaux d'exécution réduits de 8% à 54%.

4.8 Conclusion

Dans ce chapitre, 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 exprimé. Comme l'utilisateur doit définir un degré de confiance attendu, celui-ci 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 de confiance. 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 appelées ascendante, descendante et hybride, qui considèrent des seuils α dans des ordres différents. Nous avons effectué une mise en œuvre complète de ces algorithmes, et montré expérimentalement sur différents jeux de données du banc d'essais WatDiv et différents TriplesStores, que nos approches sont plus efficaces qu'une implémentation naïve.

Dans le prochain chapitre, nous nous intéressons au problème dual à celui traité dans les deux derniers chapitres. Nous traitons le problème des réponses RDF pléthoriques, c'est à dire, le problème survenant lorsqu'une requête RDF retourne un nombre de réponses qui dépasse un seuil renseigné par l'utilisateur.

Chapitre 5

Traitement de requêtes RDF aux réponses pléthoriques

Sommaire

5.1 Introduction

Dans les deux chapitres précédents, nous avons traité le problème des réponses vides en identifiant les causes de l'échec de la requête initiale et les plus grandes sous-requêtes qui réussissent. Dans ce chapitre, nous nous intéressons au problème inverse, où un utilisateur qui interroge une base de données volumineuse risque d'obtenir trop de réponses. Ce problème est connu sous le nom de problème des réponses pléthoriques (PRP).

Dans le contexte des données RDF, nous considérons qu'une requête échoue si elle retourne plus de K réponses, K étant un paramètre fourni par l'utilisateur. Dans la littérature, comme nous l'avons vu au chapitre 2, plusieurs solutions ont été proposées pour résoudre ce problème. La plupart de ces travaux se basent sur une sélection des K meilleurs réponses (Top K) [IFIS08]. Plus concrètement, ces travaux introduisent un ordre total entre les éléments de la réponse en leur associant un degré de satisfaction (pertinence) [SM86, Rob97, Kle99].

Dans nos travaux, et dans la continuité des chapitres précédents, nous considérons que le manque de connaissance sur la BC interrogée amène l'utilisateur à formuler une requête peu sélective retournant un nombre important de réponses, c'est-à-dire plus de K réponses. Il est à noter que pour traiter ce problème, les approches top K agissent sur les réponses et ne fournissent aucune explication sur les causes du problème. Notre solution s'inscrit dans le même esprit que les méthodes proposées dans les chapitres précédents : nous nous efforcons d'identifier les causes de l'échec, puis nous nous intéressons au calcul des plus grandes sous-requêtes de la requête qui réussissent. Contrairement aux approches Top K, notre solution ne nécessite pas l'intervention de l'utilisateur, et ne fait pas appel à ses préférences, ni à son historique de recherche.

Dans ce chapitre, nous présentons ainsi notre dernière contribution liée au traitement de requêtes problématiques dans le contexte des données RDF. En particulier, nous nous intéressons, dans la section 5.2, à la définition des contre-parties des MFS et des XSS pour le problème PRP. Nous illustrons ces notions via un exemple. Les sections 5.3 et 5.4 décrivent nos propositions pour le calcul des MFIS et XSS. La section 5.5 présente l'étude expérimentale menée sur le banc d'essai LUBM. Enfin, nous concluons ce chapitre dans la section 5.6.

5.2 Notions de MFIS et XSS

Dans le contexte des réponses vides, une MFS (une plus petite sous-requête qui échoue) (1) est une sous-requête qui échoue et (2) dont toutes les sous-requêtes réussissent (voir la section 3.2.1 du chapitre 3). Dans cette section, nous présentons en quoi les MFS ne sont pas applicables au problème PRP, et proposons une notion alternative.

5.2.1 Des MFS aux MFIS

Grâce à la propriété de la monotonie vue dans le chapitre 3 (définition 3 et proposition 1), nous avions la garantie que les super-requêtes d'une MFS échouent. C'est à dire, d'une manière générale, il suffit qu'une requête échoue pour savoir que toutes ses super-requêtes échouent. Dans le nouveau contexte concernant le problème PRP la monotonie n'est pas garantie.

Définition 4 *Pour le problème PRP, une requête réussit si et seulement si le nombre de réponses est inférieur ou égal à un seuil K prédéfini. Cet état correspond à une propriété booléenne notée $P(Q)$.*

Q_1 :
```
SELECT * WHERE {
?fp type FullProfessor . (t1)
?fp age ?a .             (t2)
?fp nationality ?n .     (t3)
?fp teacherOf ?c }       (t4)
```

(a) Requête $Q = t_1t_2t_3t_4$

Réponses de Q			
?fp	**?a**	**?n**	**?c**
fp1	45	US	SW
fp1	45	US	DB
fp1	45	US	ES
fp2	52	FR	PR
fp2	52	FR	CA
fp2	52	FR	IA
fp3	49	US	ML

(b) Résultats de Q

FIGURE 5.1 – Une requête SPARQL avec ses réponses

$$P(Q) = (|[[Q]]_D| \leq K) \quad i.e.,$$
$$P(Q) = \begin{cases} 1 & si\ Q\ réussit \\ 0 & si\ Q\ échoue \end{cases}$$

Considérons la requête Q_1 de la figure 5.1.a, qui recherche des `professeurs` avec leurs `âges`, `nationalités` et les `cours qu'ils enseignent`. Nous notons cette requête par $t_1t_2t_3t_4$.

La figure 5.2, illustre les sous-requête de Q avec ses MFS et XSS.

Avec un seuil $K = 3$, Q_1 échoue dans le contexte du problème PRP car elle retourne sept réponses (figure 5.1.b).

Propriété 7 *La propriété booléenne de réussite d'une requête dans le cadre du problème PRP n'est pas monotone par rapport à l'inclusion ensembliste des patrons de triplets.*

***Preuve* 9** *Par contre-exemple : soit dans le treillis deux requêtes $Q_2 = t3$ et $Q_3 = t2 \wedge t3$ Nous avons bien $Q \subseteq Q'$*

- *$P(Q_2) = 0$ car $|[[Q_2]]_D| = 4$ un nombre de réponses qui dépasse $K = 3$*
- *$P(Q_3) = 1$ car $|[[Q_3]]_D| = 3$ un nombre de réponses inférieur ou égal à K*

La monotonie n'est donc pas vérifiée car $Q_2 \subseteq Q_3 \wedge P(Q_2) < P(Q_3)$

La requête t_3 est une MFS, pourtant sa super-requête t_2t_3 réussit. Dans le cadre d'un retour à l'utilisateur, t_3 ne peut donc plus être considérée comme une cause d'échec. Nous souhaitons garder la même idée du retour fourni à l'utilisateur, c'est à dire bien identifier la cause de l'échec.

Avant d'introduire les MFIS, nous définissons d'abord les FIS (Failure-Inducing Subquery).

Définition 5 *Une FIS est une sous-requête de Q dont toutes les super-requêtes échouent pour un seuil K. L'ensemble des FIS d'une requête Q est noté $fis(Q)$.*

$$fis(Q) = \{Q^* \mid Q^* \subseteq Q \ \wedge\ |[[Q^*]]_D| > K \ \wedge\ \nexists\ Q' \text{ telle que } Q^* \subset Q' \ \wedge |[[Q']]_D| \leq K\}.$$

Définition 6 *Une MFIS (Minimal Failure-Inducing Subquery) est une FIS minimale. L'ensemble de toutes les MFIS d'une requête Q pour un seuil K est noté mfis(Q).*

$$mfis(Q) = \{Q^* \mid Q^* \in fis(Q) \ \wedge\ \nexists\ Q' \in fis(Q) \text{ telle que } Q' \subset Q^*\}.$$

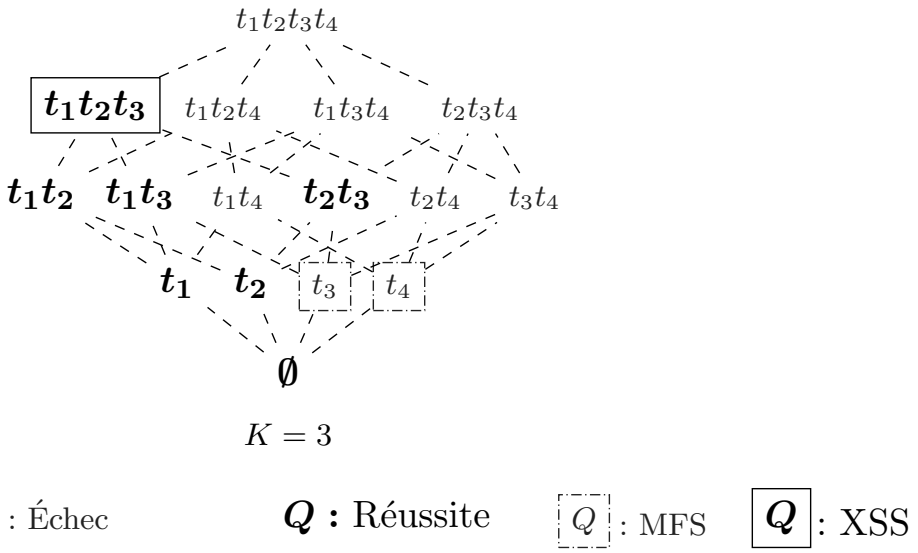


FIGURE 5.2 – Exemple de l'incompatibilité des MFS pour PRP

Dans le contexte des réponses vides, la définition des MFIS correspond à la définition des MFS. Il s'agit donc d'une définition plus générique.

Concernant la pertinence et l'utilisation des XSS, avec leur définition originale comme retour à l'utilisateur suite à la perte de la monotonie dans le contexte du problème PRP, la différence par rapport au problème des réponses vides est que l'utilisateur ne peut plus exécuter une sous-requête de la XSS avec la garantie qu'elle réussisse. Dans l'exemple de la figure 5.2, t_3 échoue alors que $t_1t_2t_3$ est une XSS. Mais les XSS fournies représentent toujours des requêtes qui réussissent et qui sont les plus proches de la requête initiale. Nous considérons que l'utilisateur a donc peu d'intérêt à exécuter une sous-requête d'une XSS. Donc la définition actuelle des XSS prend tout le sens désiré dans le cadre du problème PRP. La définition suivante des XSS est une adaptation directe de la définition des XSS pour le problème des réponses vides : seule la définition de la propriété d'échec ou de réussite est modifiée.

Définition 7 *Une XSS (Maximal Succeeding Subquery), est une sous-requête maximale qui réussit d'une requête Q pour un seuil K dont toutes les super-requêtes échouent. L'ensemble de toutes les XSS d'une requête Q est noté xss(Q).*

$$xss(Q) = \{Q^* \mid Q^* \subseteq Q \ \wedge \ |[[Q^*]]_D| \leq K \ \wedge \ \nexists \ Q' \text{ telle que } Q^* \subset Q' \wedge |[[Q']]_D| \leq K\}.$$

Nous avons identifié la propriété suivante qui permet de situer les XSS par rapport aux FIS dans le treillis des sous-requêtes de la requête initiale.

Propriété 8 *Toute XSS, si elle existe, est une sous-requête d'une FIS.*

***Preuve* 10** *Si une super-requête d'une XSS n'est pas une FIS, alors une des super-requêtes de la XSS réussit et donc la XSS n'en est pas une, car ce n'est pas la plus grande.*

Cette propriété sera utilisée pour traiter le problème décrit dans la section suivante.

Description du problème

En entrée du problème PRP considéré dans ce chapitre, nous avons une requête RDF Q qui retourne plus de K réponses $|[[Q]]_D| > K$. L'objectif est de trouver un algorithme efficace

BC		
sujet	**prédicat**	**objet**
fp_1	type	FullProfessor
fp_1	age	45
fp_1	nationality	US
fp_1	advisor	S1
fp_1	teacherOf	SW
fp_1	teacherOf	DB
fp_1	teacherOf	ES
fp_2	type	FullProfessor
fp_2	age	52
fp_2	nationality	FR
fp_2	teacherOf	PR
fp_2	teacherOf	CA
fp_2	teacherOf	IA
fp_3	type	FullProfessor
fp_3	age	49
fp_3	nationality	US
fp_3	teacherOf	ML
s_1	type	Student
s_1	nationality	US

FIGURE 5.3 – Une BC

pour identifier l'ensemble des MFIS et XSS de Q pour le seuil K donné par l'utilisateur. Dans la section suivante, nous illustrons, par un exemple, les deux notions de MFIS et XSS.

5.2.2 Illustration des MFIS et XSS

Nous considérons la base de connaissances présentée dans la figure 5.3, et la requête Q_1 illustrée dans la figure 5.1.a. La requête Q_1 cherche les `Professeurs titulaires` avec leurs `Age, Nationalité` et les `cours` qu'ils enseignent. Nous notons cette requête par $t_1t_2t_3t_4$. Les réponses de Q_1 sont présentées dans la figure 5.1.b. Cette requête renvoie 7 réponses au total. Supposons maintenant que l'utilisateur souhaite avoir au plus 3 résultats (K = 3). La requête originale formulée par l'utilisateur est insatisfaisante car le nombre de résultats dépasse le seuil donné.

En considérant la figure 5.4 qui décrit toutes les sous-requêtes de Q, notre proposition pour traiter le problème des réponses pléthoriques, consiste à fournir à l'utilisateur les deux informations suivantes (1) $MFIS = \{t_4\}$ et (2) $XSS = \{t_1t_2t_3\}$. Ces informations sont interprétées comme suit :

1. Le patron de triplet t_4 (?fp teacherOf ?c) ne peut pas être inclus dans une requête pour que celle-ci réussisse. Pour cette requête, il n'est donc pas possible de lister les cours enseignés si le nombre de réponses souhaité par l'utilisateur ne doit pas dépasser 3 réponses.
2. La requête alternative qui renvoie moins de 3 réponses consiste à rechercher les `professeurs titulaires` avec leurs `âge` et `nationalité`.

Ces commentaires peuvent aider l'utilisateur à mieux comprendre le contenu de la base de connaissances, à reformuler sa requête ou à ajuster le nombre maximum de résultats voulus. Pour fournir ces retours, nous proposons deux approches pour calculer les MFIS et XSS, présentées dans la suite de ce chapitre.

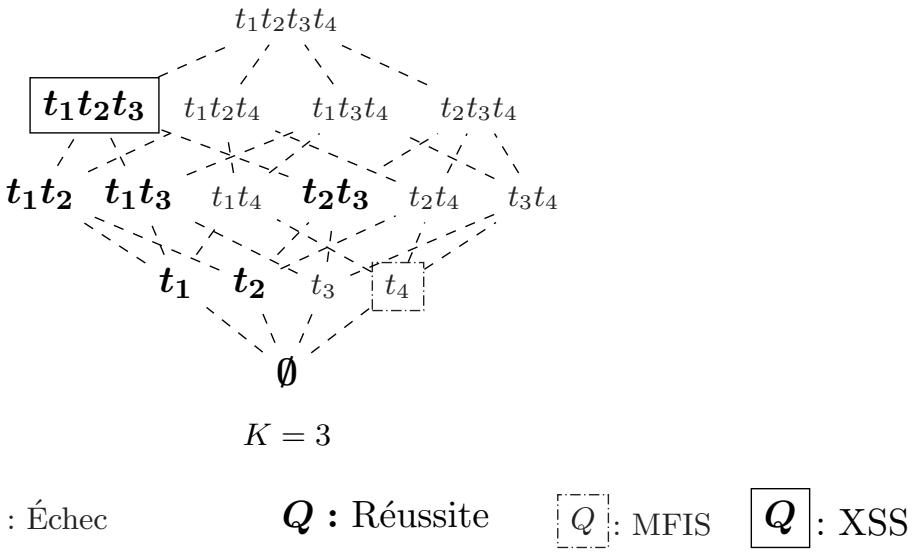


FIGURE 5.4 – Treillis des sous-requêtes de Q pour K=3

5.3 Approche naïve pour trouver les MFIS et XSS

Dans l'approche de base Breadth-First Search (BFS), nous développons un algorithme (voir l'algorithme 7) qui réalise un parcours en largeur du treillis pour rechercher les deux ensembles des $mfis(Q)$ et $xss(Q)$ d'une requête Q. Comme pour l'algorithme DFS du chapitre 3, la découverte se base sur une exploration intégrale du treillis. Cependant, les tests réalisés diffèrent puisque les propriétés des MFIS ne sont plus les mêmes que celles des MFS.

BFS applique d'abord une recherche en largeur : pour chaque requête Q' du treillis (ligne 5), nous testons et stockons les sous-requêtes de Q' dans une file (6-10). Si Q' réussit et que toutes ses super-requêtes sont des FIS, Q' est ajoutée à la listes des XSS (11-16). Si Q' échoue (ligne 17) et que toutes ses super-requêtes sont des FIS, alors Q' est insérée dans la liste des FIS (ligne 18). Ensuite les MFIS sont mises à jour (lignes 19-20) en supprimant tous les éléments contenant Q' (seuls les éléments minimaux sont conservés). Cet algorithme de base nécessite l'exécution de toutes les requêtes du treillis pour atteindre notre objectif, à savoir, la découverte des MFIS et XSS d'une requête Q.

Pour illustrer cet algorithme, nous considérons la requête $Q_1 = t_1t_2t_3t_4$ avec le paramètre K fixé à 3 et la BC de notre exemple courant. BFS exécute $2^4 - 2 = 14$ requêtes, c'est à dire tous les éléments du treillis excepté la requête initiale Q_1 qui échoue de base et l'ensemble vide qui, pour ce problème, échoue par convention.

Même si la monotonie n'est pas vérifiée dans ce contexte, une amélioration possible de cet algorithme de base peut-être faite en se basant sur les définitions des MFIS et XSS et la propriété 8. En effet, nous pouvons restreindre l'espace de recherche dans le treillis en ne considérant que les sous-requêtes des FIS, car les autres ne peuvent être ni MFIS, ni XSS. Nous n'avons volontairement pas intégré cette optimisation à la méthode de base. L'étude de l'impact de cette amélioration fait partie de nos perspectives. Dans la section suivante, nous présentons notre deuxième approche qui améliore l'algorithme BFS en utilisant des techniques dont le but est d'éviter l'exécution de certains éléments du treillis.

Algorithm 7: Trouver les MFIS et XSS d'une requête Q pour un seuil K

```
BFS(Q, D, K)
    entrées: Q, une requête SPARQL, Q = t_1 ∧ ... ∧ t_n ;
             D, une base de données RDF;
             k, un seuil sur le nombre de résultats maximum ;
    sorties : mfis, l'ensemble des mfis de Q (mfis(Q)) ;
             xss, l'ensemble des xss de Q (xss(Q));
1   mfis(Q) ← ∅;
2   xss(Q) ← ∅;
3   fis(Q) ← ∅;
4   file.enfiler(Q); // file est une file sans doublons
5   while file n'est pas vide do
6       Q' ← file.défiler(); // récupérer l'élément en tête de file
7       foreach t_i ∈ Q' do
8           file.enfiler( Q' − t_i) // on enfile les sous-requête de Q'
9       parentFIS = vrai ;
10      foreach t_i ∈ (Q − Q') do // On vérifie que toutes les super-requêtes
         directes sont des fis
11          parentFIS = parentFIS ∧ [(Q' ∧ t_i) ∈ fis(Q)]
12      if parentFIS then // Q' réussit
13          if |[[Q']]_D| ≤ K then
14              xss(Q) = xss(Q) ∪ Q'
15          else // Q' échoue
16              fis(Q) = fis(Q) ∪ Q'; // Q' est une FIS
17              foreach t_i ∈ (Q − Q') do // les super-requêtes de Q' ne sont pas
                 minimales
18                  mfis(Q) = mfis(Q) - (Q' ∪ t_i)
19              mfis(Q) = mfis(Q) ∪ Q' ; // Q' est minimale jusqu'à preuve du
                 contraire
20  return mfis(Q), xss(Q);
```

5.4 Approche basée sur les cardinalités

Sans la monotonie pour la propriété de réussite d'une requête, il n'est pas possible de déduire l'état (réussite ou échec) d'une requête sans l'exécuter. Pour cette raison, l'approche de base BFS doit exécuter une à une, l'ensemble des requêtes des treillis.

Dans cette section, nous identifions une condition suffisante pour obtenir une propriété monotone sur une partie du treillis lorsque la requête initiale est une requête en étoile, c'est-à-dire une requête RDF dont tous les patrons de triplets commencent par une même variable. Pour identifier cette condition nous nous sommes basés sur le fait que nous avions besoin de connaître l'impact de chaque patron de triplet de la requête sur le nombre de résultats obtenus. Cet impact dépend directement de la cardinalité des prédicats qui se trouvent dans les patrons de triplets, c'est à dire le nombre d'occurrences d'un prédicat dans la BC. Autrement dit, grâce aux cardinalités, nous pouvons prédire si le nombre de réponses d'une requête Q diminue, augmente ou reste identique lorsque nous ajoutons ou supprimons un patron de triplet à Q. Cela nous permet de découvrir les MFIS et les XSS en exécutant moins de requêtes que BFS.

Dans la littérature, plusieurs approches ont été proposées pour définir la notion de cardinalité d'un prédicat RDF, et notamment pour le problème d'optimisation de requêtes SPARQL [NM11]. Dans la suite de cette section, nous donnons deux définitions possibles sur cette notion et en déduisons l'impact que cela a sur le nombre de résultats d'une requête Q lorsqu'un patron de triplet lui est ajouté.

5.4.1 Cardinalités globales

Nous considérons une requête Q= $Q' \wedge t$ où t est un patron de triplet $\in (U \cup V) \times U \times (U \cup V \cup L)$, c'est à dire que son prédicat, noté p, n'est pas variable. Intuitivement, la cardinalité globale d'un prédicat p est le nombre d'occurrences minimum (noté $CardGlobal_{min}(p)$) et maximum (noté $CardGlobal_{max}(p)$) du prédicat RDF pour l'ensemble des sujets de la BC. Nous définissons maintenant formellement cette notion.

Nous considérons un ensemble de données RDF D. Les sujets de D sont définis comme suit :

$$sujet(D) = \{\ s \mid \exists\ p, o : (s, p, o) \in D\ \}\ (1)$$

Prenons l'exemple de la BC de la figure 5.3, $sujet(D) = \{fp_1, fp_2, fp_3, s_1\}$.

Pour un sujet s et un prédicat p, le nombre d'occurrences du prédicat p dans s est défini par:

$$count(s, p) = \mid \{(s, p, o) \mid (s, p, o) \in D\} \mid\ (2)$$

Pour le même exemple de BC, $count(fp_1, teacherOf) = 3$.

La cardinalité globale minimale d'un prédicat p est définie comme le nombre minimal d'occurrences de ce prédicat chez les sujets de D.

$$CardGlobal_{min}(p) = \min_{s \in sujet(D)} count(s, p)\ (3)$$

De même, nous définissons la cardinalité globale maximale d'un prédicat p comme étant le nombre maximal d'occurrences de ce prédicat chez les sujets de D.

$$CardGlobal_{max}(p) = \max_{s \in sujet(D)} count(s, p)\ (4)$$

Prédicat	Cardinalités	Impact sur le nombre de réponses
type	1-1	Reste le même
age	0-1	Baisse ou reste le même
nationality	1-1	Reste le même
teacherOf	0-n	Imprévisible

TABLEAU 5.1 – Les cardinalités globales des prédicats de Q et leurs impacts

Réponses de Q_1			
?t	**?c**	**?a**	**?n**
fp1	SW	45	US
fp1	DB	45	US
fp1	ES	45	US
fp2	PR	52	FR
fp2	CA	52	FR
fp2	IA	52	FR
fp3	ML	49	US

(a) Requête $Q_1 = t_1t_2t_3t_4$

Réponses de Q_2		
?t	**?c**	**?a**
fp1	SW	45
fp1	DB	45
fp1	ES	45
fp2	PR	52
fp2	CA	52
fp2	IA	52
fp3	ML	49

(b) Requête $Q_2 = t_1t_2t_3$

FIGURE 5.5 – Les réponses retournées par Q_1 et Q_2

Pour notre exemple de BC, $CardGlobal_{min}(teacherOf) = 0$ (s_1 n'a pas ce prédicat) ; $CardGlobal_{max}(teacherOf) = 3$.

5.4.1.1 Énumération des cardinalités globales avec leurs impacts

Voici les différentes cardinalités globales qu'un prédicat peut avoir et leurs impacts sur le nombre de résultats d'une requête :

— [0-1] : c'est à dire $CardGlobal_{min}(p) = 0$ et $CardGlobal_{max}(p) = 1$. Lorsque nous ajoutons un patron de triplet dont le prédicat est d'une cardinalité [0-1] à Q, le nombre de résultats de Q diminue ou reste identique.

— [1-1] : le nombre de résultats de Q reste identique.

— [1-n] : le nombre de résultats augmente ou reste identique.

— [0-n] : ce dernier type de cardinalité intègre les trois types précédents. En conséquence, nous ne pouvons pas avoir d'information sur la manière dont cela influence le nombre de résultats de Q.

Le tableau 5.1 récapitule ce que nous venons de citer ci-dessus, et donne les cardinalités globales des prédicats de la requête de notre exemple courant.

Pour le prédicat age, les cardinalités globales sont calculées comme suit.

1. $CardGlobal_{min}(age) = min(count(fp1, age), count(fp2, age), count(fp3, age), count(s1, age))$.
   $CardGlobal_{min}(age) = min(1, 1, 1, 0) = 0$
2. $CardGlobal_{max}(age) = max(count(fp1, age), count(fp2, age), count(fp3, age), count(s1, age)$.
   $CardGlobal_{max}(age) = max(1, 1, 1, 0) = 1$;

Par exemple, les cardinalités globales du prédicat nationalité sont, cardMin=1 et cardMax =1. Considérons la requête $Q_1 = t_1t_2t_3t_4$ de la figure 5.1. Lorsque nous supprimons le prédicat nationalité de cette requête ($t_1t_2t_4$), le nombre de réponses reste identique (7 réponses). Cet exemple est illustré dans la figure 5.5.

```
Q_4 : SELECT * {
        ?s1 age ?a
    }
```

⇒

Réponses de Q_4	
?s	**?a**
fp1	45
fp2	52
fp3	49

FIGURE 5.6 – Q_4 avec ses réponses

```
Q_5 : SELECT * {
        ?s1 age ?a .
        ?s2 nationality ?n
    }
```

⇒

Réponses de Q_5			
?s1	**?a**	**?s2**	**?n**
fp1	45	fp1	US
fp1	45	fp2	FR
fp1	45	fp3	US
fp1	45	s1	US
fp2	52	fp1	US
fp2	52	fp2	FR
fp2	52	fp3	US
fp2	52	s1	US
fp3	49	fp1	US
fp3	49	fp2	FR
fp3	49	fp3	US
fp3	49	s1	US

FIGURE 5.7 – Q_5 avec ses réponses

5.4.1.2 Types de requêtes traitées

Dans ce chapitre nous traitons les requêtes en étoile, c'est à dire les requêtes ayant la même variables comme sujet pour tous les patrons de triplets. Néanmoins, pour illustrer les types de requêtes non traités dans ce chapitre, nous considérons la requête Q_4 de la figure 5.6.

Lorsqu'un patron de triplet introduit uniquement de nouvelles variables, cela génère un produit cartésien. En conséquence, le nombre de réponses augmente indépendemment de la cardinalité des prédicats. Par exemple, la requête Q_5 de la figure 5.7 est la jointure de Q_4 avec un patron de triplet ayant deux variables différentes comme sujet (?s2) et objet (?n). Nous remarquons dans la figure 5.7 qu'en ajoutant ce patron de triplet, un produit cartésien est réalisé menant à l'augmentation du nombre de réponses (12 réponses).

Lorsque le patron ajouté n'introduit aucune nouvelle variable, celui-ci correspond à un filtre qui ne fait que réduire le nombre de réponse indépendemment de la cardinalité des prédicats. Par exemple, considérons l'exemple de la figure 5.8 où la requête Q_6 est la jointure de Q_4 avec un patron de triplet ayant une variable qui existe deja (?s) comme objet. Nous remarquons dans cette même figure que le nombre de réponses est réduit à 0.

De manière similaire, lorsque l'objet du parton de triplet ajouté est une constante, cela ne

```
Q_6 : SELECT * {
        ?s age ?a .
        ?s teacherOf ?s
    }
```

⇒

Réponses de Q6	
?s	**?a**

FIGURE 5.8 – Q_6 avec ses réponses

```
Q_7 SELECT * {
    ?s age ?a .
     ?s type fullProfessor
}
```

⇒

Réponses de Q_7	
?s	**?a**
fp1	45
fp2	52
fp3	49

FIGURE 5.9 – Q_7 avec ses réponses

fait que réduire ou garder le même nombre de réponses indépendemment de la cardinalité des prédicats. Ceci est illustré dans la figure 5.8. Lorsque on ajoute (?s type fullProfessor) à la requête Q_4, le nombre de réponses reste le même.

Les cardinalités globales ont des limites : la plupart des prédicats auront une cardinalité minimale de 0. En effet, il est rare qu'un prédicat ait une valeur pour tous les sujets d'une BC réelle. Un prédicat peut avoir une cardinalité globale de 0 à n alors que l'on peut créer des groupes de sujets pour lesquels la cardinalité est différente et donc beaucoup plus précise. L'idée est d'affiner les cardinalités globales en fonction de groupes de sujets, autrement appelés classes de données.

5.4.2 Cardinalités locales à une classe

Dans cette section, nous examinons les cardinalités relatives (locales) à une classe RDF qui nous aident à obtenir des cardinalités plus précises. La cardinalité d'un prédicat par rapport à une classe est le nombre d'occurences minimum $CardClass_{Min}$ et maximum $CardClass_{Max}$ du prédicat RDF dans les instances d'une classe donnée. Les cardinalités locales à une classe des prédicats peuvent être exprimées via un schéma OWL vu dans le premier chapitre (owl:min-Cardinality, owl:max-Cardinality). Malheureusement, le schéma OWL n'est pas utilisé par toutes les BC. Par conséquent nous proposons des définitions de cardinalités locales pour pouvoir les calculer à partir des instances de la BC. Pour cela, les sujets doivent être typés à l'aide du Schéma RDFS (rdfs:class) et les prédicats sont des propriétés RDF (rdf:property) avec un domaine (rdfs:domain) qui est une classe RDFS. Si le domaine n'est pas défini, conformément à la sémantique RDFS, nous considérons la superclasse de toutes les classes RDFS qui est rdfs:Resource.

Les instances d'une classe C dans un ensemble de données D sont définies comme suit.

$$instances(C) = \{\ s \mid \exists\ (s, Type, C) \in D\ \} \quad (5)$$

Par exemple, $instances(FullProfessor) = \{fp1, fp2, fp3\}$.

En nous basant sur le nombre d'occurrences d'un prédicat p pour un sujet s, nous pouvons définir la cardinalité minimale d'un prédicat RDF par rapport à une classe C.

$$CardClass_{min}(p, C) = \min_{s \in instances(C)} count(s, p) \quad (6)$$

Et de même, la cardinalité maximale de p par rapport à C :

$$CardClass_{max}(p, C) = \max_{s \in instances(C)} count(s, p) \quad (7)$$

Prédicat	Cardinalités locales à FullProffesor	Impact sur le nombre de réponses
type	1-1	Reste le même
age	1-1	Reste le même
nationality	1-1	Reste le même
teacherOf	1-n	Augmente ou reste le même

TABLEAU 5.2 – Les cardinalités locales à la classe "FullProfessor" des prédicats de Q et leurs impacts

5.4.2.1 Exemple pour les cardinalités locales à une classe

Considérons de nouveau le prédicat age.

1. $CardClass_{min}(age, FullProfessor) = min(count(fp1, age), count(fp2, age), count(fp3, age))$.
   $CardClass_{min}(age, FullProfessor) = min(1, 1, 1) = 1$;
2. $CardClass_{max}(age, FullProfessor) = max(count(fp1, age), count(fp2, age), count(fp3, age))$.
   $CardClass_{max}(age, FullProfessor) = max(1, 1, 1) = 1$;

Comme nous le voyons dans notre exemple, les cardinalités locales à la classe FullProfessor du prédicat age [1-1] sont plus précises que les globales [0-1]. Avec les cardinalités globales du prédicat age [0-1], nous savons que chaque sujet de la BC peut ne pas avoir un age (0), ou peut avoir un seul age au maximum. Avec les cardinalité locales à la classe FullProfessor pour l'age [1-1], nous savons que chaque sujet de type FullFrofessor doit avoir un et un seul age. Nous présentons maintenant les propriétés liées à notre algorithme se basant sur les cardinalités.

5.4.3 Exploration du treillis basée sur les cardinalités

Les cardinalités, quelles soient globales ou locales à une classe, permettent, selon leur valeur, de déduire l'état d'une requête (réussite ou échec) à partir de ses sous-requêtes ou super-requêtes. Voici les règles que nous avons identifiées.

5.4.3.1 Inférences basées sur les cardinalités

Soit p un prédicat et Q une requête RDF en étoile. Par abus de notation, nous notons, $Q \wedge p$ la conjonction de la requête Q avec un patron de triplet dont le sujet est celui utilisé dans Q, le prédicat est p et l'objet est une variable. La notation $Q - p$ sera utilisée avec une signification analogue pour la suppression d'un patron de triplet. La liste des règles d'inférences est la suivante.

Soit p un prédicat de cardinalité [0-1] :

1. Si Q réussit, alors $Q \wedge p$ réussit.
2. Si Q échoue, alors $Q - p$ échoue.

Soit p un prédicat de cardinalité [1-1] :

3. Si Q réussit, alors $Q \wedge p$ réussit.
4. Si Q échoue, alors $Q - p$ échoue.
5. Si Q réussit, alors $Q - p$ réussit.
6. Si Q échoue, alors $Q \wedge p$ échoue.

Soit p un prédicat de cardinalité [1-n] :

7. Si Q échoue, alors $Q \wedge p$ échoue.
8. Si Q réussit, alors $Q - p$ réussit.

Avec ces règles, il semble donc possible d'explorer uniquement une partie du treillis des sous-requêtes afin de déterminer les MFIS et les XSS. Une notion essentielle qui va nous servir lors du reste du chapitre est la notion de point de départ pour cette exploration.

5.4.3.2 Partie intéressante du treillis liée à Q_{base}

Nous allons définir la partie intéressante du treillis par rapport à une requête particulière appelée point de départ et notée Q_{base}. Soit Q_{base} la requête constituée de la conjonction des patrons de triplets dont les prédicats sont de cardinalités [0-1] ou [1-1]. Q_{base} est une sous-requête de la requête initiale Q. Les propriétés suivantes sont vérifiées lorsque Q_{base} échoue.

Propriété 9 *Si Q_{base} échoue, alors toutes les sous requêtes de Q_{base} échouent ($\forall\, Q \subseteq Q_{base}$, Q échoue), à cause des règles (2) et (6).*

Propriété 10 *$\forall\, Q', si\ Q' \wedge\ Q_{base}$ échoue, alors Q' échoue en appliquant récursivement les règles (2) et (6) pour tous les patrons de triplets de Q_{base}.*

Propriété 11 *$\forall\, Q'$, si Q' réussit alors $Q' \wedge\ Q_{base}$ réussit. Le résultat est la contraposée de la propriété 10, qui peut aussi se retrouver en appliquant récursivement les règles (1) et (3) à Q' pour les patrons de triplets de Q_{base}.*

Proposition 4 *Toute XSS, si elle existe, est une super-requête de Q_{base}.*

***Preuve* 11** *Cette propriété est prouvée par l'absurde. Soit Q une XSS et $Q_{base} \not\subseteq Q$, Q réussit et $\forall\, Q'$ telle que $Q \subset Q'$, Q' échoue (1).*
Q réussit donc $(Q \wedge Q_{base})$ réussit (propriété 11).
$Q \subset (Q \wedge Q_{base})$ ce qui contredit (1).

Proposition 5 *Q est une FIS si et seulement si $Q - Q_{base}$ est une FIS*

***Preuve* 12** *L'implication Q est FIS $\Rightarrow$ $Q - Q_{base}$est FIS est prouvée par l'absurde. On suppose que Q est une FIS est que $Q - Q_{base}$ n'est pas une FIS. Il existe donc une super-requête Q^* de $Q - Q_{base}$ qui réussit ($Q - Q_{base} \subseteq Q^*$). D'après la proposition 11, $Q^* \wedge\ Q_{base}$ réussit. Or $Q^* \wedge\ Q_{base}$ est une super-requête de Q, donc Q n'est pas une FIS, ce qui contredit l'hypothèse de départ. L'implication $Q - Q_{base}$ est FIS $\Rightarrow$ Q est FIS est immédiate, car Q est une super-requête de $Q - Q_{base}$.*

Définition 8 *Une $MFIS_{base}Q^*$, est une FIS pour laquelle le critère de minimalité est vérifié par rapport aux requêtes du treillis situées entre la requête initiale Q et Q_{base} uniquement :*

- *$Q_{base} \subseteq Q' \subseteq Q$*
- *Q^* est une FIS*
- *$\nexists\, Q'$ telle que $Q_{base} \subseteq Q' \wedge Q' \subset Q^* \wedge Q'$ est une FIS (minimalité)*

Proposition 6 *Si Q^* est une $MFIS_{base}$, alors $Q^* - Q_{base}$ est une MFIS*

***Preuve* 13** *D'après la proposition 5, si Q^* est une FIS, alors $Q^* - Q_{base}$ est une FIS. La minimalité est prouvée par absurde : soit Q^* une $MFIS_{base}$ telle que $Q^* - Q_{base}$ ne soit pas minimale : $\exists\ Q' \subset (Q^* - Q_{base})$ telle que Q' soit une FIS. D'après la proposition 5, $Q' \wedge\ Q_{base}$ est donc une FIS. Or $Q_{base} \subseteq (Q' \wedge Q_{base})$ et $(Q' \wedge\ Q_{base}) \subset Q^*$, ce qui contredit le critère de minimalité sur Q^*, qui n'est donc pas une $MFIS_{base}$.*

5.4.4 Algorithme basé sur les cardinalités

Dans le cadre des requêtes RDF, nous nous intéressons au calcul des $mfis(Q)$ et $xss(Q)$ d'une requête RDF qui retourne plus de K résultats. Par rapport à Q_{base}, le treillis est désormais décomposé en trois parties :

1. les sous-requêtes de Q_{base}, composées de prédicats de cardinalité [0-1] et [1-1], dont le nombre de résultat diminue avec l'ajout de prédicats. Par rapport à l'ensemble de ces sous-requêtes, Q_{base} est donc à la fois celle possédant le moins de résultat et la plus proche de la requête initiale ;
2. les sous-requêtes incomparables avec Q_{base} (Q $\not\subseteq Q_{base}$ et $Q_{base} \not\subseteq Q$). Pour ces requêtes, l'ajout de prédicats de Q_{base} diminue le nombre de résultats tout en rapprochant la requête de la requête initiale pour la découverte des XSS, et, même si cette partie du treillis peut contenir des MFIS, celles-ci peuvent être identifiées à partir des super-requêtes de Q_{base} (proposition 6) ;
3. les super-requêtes de Q_{base}, qui vont être explorées par notre algorithme.

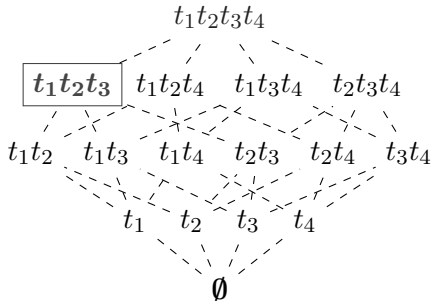
Nous appelons notre algorithme se basant sur les cardinalités CardAlgorithme (algorithme 8). Il nous permet d'éviter l'exécution des deux premières parties citées ci-dessus du treillis, pour rechercher toutes les MFIS et XSS d'une requête Q. Cet algorithme peut être utilisé avec n'importe quel type de cardinalité (globales ou locales à une classe). Dans cette section nous détaillons et illustrons cet algorithme. CardAlgorithme suit les étapes suivantes.

Etape 1 : Nous calculons le point de départ (Q_{base}), basé sur les cardinalités des prédicats. Pour rappel, Q_{base} est composé de tous les patrons de triplets ayant un prédicat de cardinalités [0-1] ou [1-1].

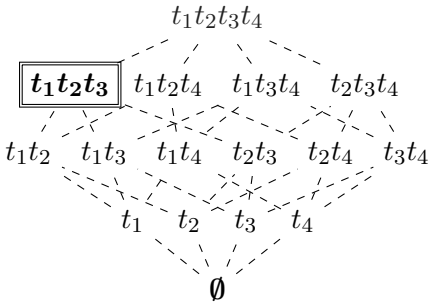
Etape 2 : Nous considérons et recherchons les XSS uniquement dans l'ensemble des super-requêtes de Q_{base} (proposition 4). Les MFIS sont également calculées en utilisant les super-requêtes de Q_{base} (proposition 6). Ainsi, notre algorithme CardAlgorithme n'exécute que les super-requêtes de Q_{base} afin de trouver les XSS et MFIS.

5.4.4.1 Illustration de l'algorithme CardAlgorithme

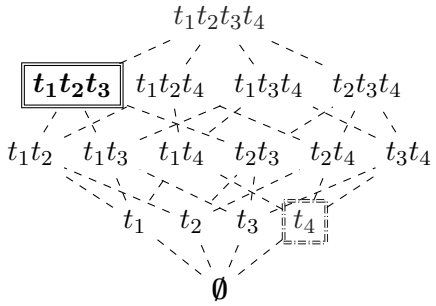
Pour illustrer l'algorithme 8, nous considérons la requête et la BC de notre exemple courant (voir la figure 5.1) et l'ensemble des treillis représentant cette illustration dans la figure 5.10. Premièrement, l'algorithme calcule Q_{base} (ligne 3) à partir des prédicats ayant les cardinalités [0-1] et [1-1], ce qui donne la sous-requête $t_1t_2t_3$ (voir 5.10.1). Comme la super-requête unique de Q_{base}, $Q' = t_1t_2t_3t_4$ est une FIS (ligne 7) et Q_{base} réussit (ligne 8), l'algorithme qualifie $t_1t_2t_3$ comme une XSS (ligne 9) (voir 5.10.2). La MFIS t_4 est trouvée en supprimant Q_{base} de Q' $t_1t_2t_3t_4 - t_1t_2t_3 = t_4$ après vérification de sa minimalité (ligne 10-14). Au final, l'algorithme se basant sur les cardinalité a exécuté une seule requête pour trouver les MFIS et XSS de la



(1) identifier $Q_{base} = t_1t_2t_3$



(2) identifier une XSS= Q_{base}



(3) identifier une MFIS = t_4

légende: Q : XSS Q : MFIS Q : Q_{base}

FIGURE 5.10 – Illustration de l'algorithme CardAlgorithme

requête Q, alors que BFS a exécuté 14 requêtes. En fait, plus Q_{base} est proche de la requête initiale, moins notre algorithme CardAlgorithme exécute de requêtes.

Algorithm 8: Trouver les MFIS et XSS d'une requête Q pour un seuil K

```
CardAlgorithme(Q, D, K)
    entrées: Q, une requête SPARQL, Q = t_1 ∧ ... ∧ t_n ;
             D, une base de données RDF;
             k, un seuil sur le nombre de résultats maximum
    sorties : mfis, l'ensemble des mfis de Q (mfis(Q))
             xss, l'ensemble des xss de Q (xss(Q))
1   mfis ← ∅;
2   xss ← ∅;
3   fis ← ∅;
4   Q_base ← CalculerQbase(); // calcul du point de départ Q_base
5   List ← SuperRequêtes(Q_base);// lister les super-requêtes de Q_base
6   while List ≠ ∅ do
7       Q' ← List.pop(); // récupérer le premier élément de la liste
8       if SuperRequêtes(Q') ∈ fis then // les super requêtes de Q' sont FIS
9           if |[[Q']]_D| ≤ K then // Q' réussit
10              xss = xss ∪ Q';
11          else // Q' échoue
12              fis ← fis ∪ Q';
13              foreach t_i ∈ (Q − Q') do
14                  mfis = mfis - ((Q' − Q_base) ∧ t_i);
15              mfis = mfis ∪ (Q' − Q_base);
16  return mfis, xss;
```

5.5 Implémentation et expérimentations

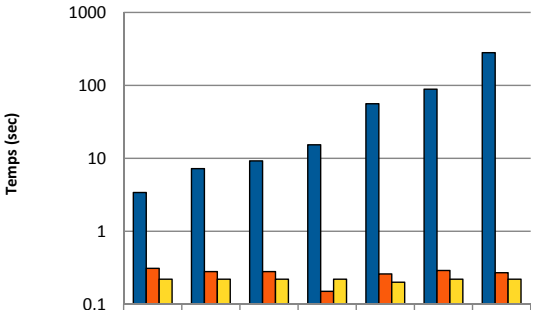
Dans cette section, nous étudions les performances de l'algorithme CardAlgorithme appliqué avec les deux types de cardinalité : cardinalités Globales (CardGlobal) et cardinalités locales à une classe (CardClass). Nous comparons les deux versions avec la méthode de base BFS (exécution de toutes les sous-requêtes d'une requête Q avec des réponses pléthoriques).

Algorithmes

Nous avons implémenté CardAlgorithme et BFS avec Oracle Java 1.8 64 bits. Ces algorithmes prennent en entrée une requête avec des réponses pléthoriques et un nombre maximal de résultats (seuils). Ils renvoient les ensembles de MFIS et de XSS de cette requête pour le seuil K. Dans notre implémentation actuelle, ces algorithmes sont exécutés sur Jena TDB.

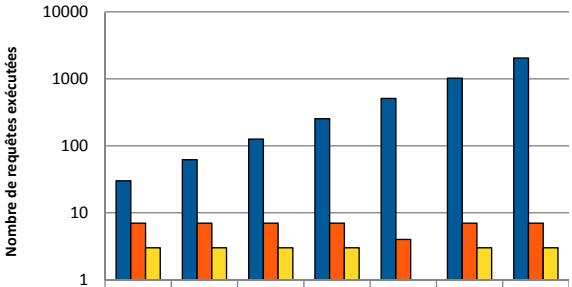
Configuration expérimentale

Nos expériences ont été menées sur un système Ubuntu Server 16.04 LTS avec processeur Intel XEON E5-2630 v3 @ 2,4 GHz et avec 16 Go de RAM. Pour nos expériences, les résultats présentés sont la moyenne de cinq exécutions consécutives des algorithmes. Pour éviter un effet de démarrage à froid, une analyse préliminaire est effectuée mais n'est pas incluse dans les résultats.



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
DFS	3,4	7,22	9,22	15,33	55,98	88,55	280,48
CardGlobal	0,31	0,28	0,28	0,15	0,26	0,29	0,27
CardClass	0,22	0,22	0,22	0,22	0,2	0,22	0,22

FIGURE 5.11 – Temps d'exécution (32M triplets)



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
DFS	30	62	126	254	510	1022	2046
CardGlobal	7	7	7	7	4	7	7
CardClass	3	3	3	3	1	3	3

FIGURE 5.12 – # Nombre de requêtes exécutées

Jeu de données et requêtes

Nous avons utilisé un jeu de données de 34 millions de triplets générés avec le banc d'essai Lehigh University Benchmark (LUBM) [GPH05]. Nous considérons 7 requêtes du banc d'essai LUBM que nous avons modifiées pour obtenir des requêtes de réponses pléthoriques (voir Tableau 5.2). Ces requêtes possèdent de 4 à 10 patrons de triplets.

Experimentation

Dans cette expérience, nous avons évalué les performances de notre algorithme CardAlgorithme lorsqu'il utilise les cardinalités globales (CardGlobal) ou locales à une classe (CardClass) par rapport à la méthode de base BFS. Cette expérience a été exécutée avec un seuil défini arbitrairement $K = 10$. La figure 5.11 indique le temps d'exécution de chaque méthode pour chaque requête sur Jena TDB avec le jeu de données de 32 millions de triples. La figure 5.12 donne le nombre de requêtes exécutées par chaque méthode.

Calcul des cardinalités

Une étape préliminaire est nécessaire pour que nos deux approches CardGlobal et CardClass puissent s'exécuter. Cette étape concerne le calcul des cardinalités que nous avons effectué à l'aide d'un programme basé sur Apache Spark. Les temps de calcul des cardinalités sont les suivants.

— 27 secondes pour le calcul des cardinalités globales.

— 17 secondes pour le calcul des cardinalités locales à une classe.

Cette étape n'est réalisée qu'une seule fois pour un jeu de données et n'est pas prise en compte dans le temps d'exécution des requêtes.

Experimentation basée sur LUBM

Cette expérience montre l'avantage de nos méthodes CardGlobal et CardClass. En comparaison avec BFS, nos méthodes exécutent peu de requêtes pour trouver les MFIS et les XSS pour toutes les requêtes. En conséquence, ces méthodes ont des temps d'exécution plus courts. Par exemple, BFS a besoin de 88,55 secondes pour rechercher les MFIS et XSS de la requête Q6, alors que nos méthodes nécessitent environ 0,29 secondes. La différence de temps d'exécution

Q1 (5TP)	`SELECT * WHERE { ?X memberOf University0 . ?X type FullProfessor . ?X degreeFrom> ?u . ?X teacherOf ?t . ?X undergraduateDegreeFrom University451 }`
Q2 (6TP)	`SELECT * WHERE { ?X memberOf University0 . ?X emailAddress ?Y1 . ?X type FullProfessor . ?X degreeFrom ?u . ?X teacherOf ?t . ?X undergraduateDegreeFrom University451 }`
Q3 (7TP)	`SELECT * WHERE { ?X memberOf University0 . ?X emailAddress ?Y1 . ?X type FullProfessor . ?X degreeFrom ?u . ?X teacherOf ?t . ?X undergraduateDegreeFrom University451 . ?X worksFor ?k }`
Q4 (8TP)	`SELECT * WHERE { ?X memberOf University0 . ?X emailAddress ?Y1 . ?X type FullProfessor . ?X degreeFrom ?u . ?X teacherOf ?t . ?X undergraduateDegreeFrom University451 . ?X telephone ?r . ?X worksFor ?g }`
Q5 (9TP)	`SELECT * WHERE { ?X emailAddress ?Y1 . ?X type FullProfessor . ?X teacherOf ?t . ?X name ?Y3 . ?X telephone ?u . ?X worksFor ?k . ?X researchInterest ?r . ?X doctoralDegreeFrom ?d . ?X mastersDegreeFrom University196> }`
Q6 (10TP)	`SELECT * WHERE { ?X memberOf University0 . ?X emailAddress ?Y1 . ?X type FullProfessor . ?X degreeFrom ?u . ?X teacherOf ?t . ?X telephone ?r . ?X name FullProfessor4 . ?X researchInterest ?ss . ?X mastersDegreeFrom ?ssq . ?X worksFor ?g }`
Q7 (11TP)	`SELECT * WHERE { ?X memberOf University0 . ?X emailAddress ?Y1 . ?X type FullProfessor> . ?X degreeFrom> ?u . ?X teacherOf ?t . ?X telephone ?r . ?X name FullProfessor4 . ?X researchInterest ?ss . ?X doctoralDegreeFrom ?d . ?X mastersDegreeFrom ?ssq . ?X worksFor ?g }`

Tableau 5.3 – Requêtes utilisées pour les expérimentations

dépend fortement du nombre de requêtes évitées par nos méthodes. Par exemple, nos méthodes exécutent environ 7 requêtes pour Q6 alors que BFS nécessite 1022 requêtes. Pour CardClass, ce résultat représente le gain de performance le plus important grâce aux cardinalités locales à une classe.

5.6 Conclusion

Dans cet chapitre, nous avons considéré le problème des réponses pléthoriques dans le contexte des donées RDF. Pour ce problème, une requête échoue si elle renvoie un nombre de réponses supérieur à un seuil K donné par l'utilisateur. Afin de fournir à l'utilisateur un retour d'information pertinent, nous avons proposé de calculer les MFIS et XSS de la requête défaillante, car ils donnent un aperçu clair des causes d'échec de la requête, et un ensemble de requêtes relâchées qui sont proposées comme alternatives à sa requête défaillante. La méthode de base appelée BFS consiste à exécuter chaque sous-requête de la requête initiale. Cependant, nous avons observé et prouvé que, dans le contexte des requêtes en étoile, les MFIS et les XSS pour un seuil donné K peuvent être trouvées plus facilement en utilisant les cardinalités des prédicats (globales et locales à une classe). Ainsi, nous avons défini une approche alternative à BFS appelée CardAlgorithme qui évite l'exécution de certaines sous-requêtes de la requête initiale. Nous avons réalisé une mise en œuvre complète de cet algorithme et montré de manière expérimentale sur l'ensemble de données du banc d'essai LUBM que notre approche requiert des temps de traitements significativement réduits par rapport à la méthode de base. Notre expérience montre en effet l'intérêt de prendre en compte les cardinalités globales ou les cardinalités locales à une classe. En comparaison avec BFS, nos méthodes exécutent moins de requêtes pour trouver les MFIS et XSS et ont des temps d'exécution plus courts.

Comme perspectives, nous comptons d'abord exploiter les règles concernant les prédicats de cardinalité [1-n]. Grâce à ces règles nous pensons pouvoir déduire la réussite ou l'échec de plus de requêtes sans avoir à les exécuter. Une autre perspective consiste à prendre en compte le caractère incertain des données RDF présentes dans les BC usuelles comme nous l'avons fait dans les deux chapitres précédents pour le problème des requêtes à réponse vide.

Conclusion

Dans ce travail, nous avons abordé un aspect important lié aux larges bases de connaissances disponibles sur le Web. Il s'agit de l'exploitation de ces bases, en présence de degrés de confiance, à des fins de prise de décision. L'approche d'exploitation proposée offre un traitement coopératif et intelligent des requêtes utilisateurs retournant des résultats insatisfaisants ne permettant pas de servir les besoins des utilisateurs ou ne répondant pas à leurs attentes. Ce travail constitue un pas significatif dans l'amélioration de l'ergonomie et de la convivialité des systèmes modernes d'exploitation des bases de connaissances à grande échelle, des applications du monde réel issues du domaine industriel ou du domaine académique.

La première contribution discutée dans ce travail concerne le traitement des requêtes SPARQL (paramétrées avec un degré de confiance) conduisant à une vacuité de résultats, i.e. aucun résultat n'est retourné à l'utilisateur. Les raisons de cet échec peuvent être : (i) dépendantes des données de la base interrogée (il n'y a aucune donnée dans la base qui satisfait les conditions de la requête formulée); (ii) dépendantes de la fiabilité de la base (certaines données de la base satisfont les critères de la requête mais avec un niveau de fiabilité inférieur au seuil α défini par l'utilisateur). Le traitement proposé permet non seulement de fournir des requêtes alternatives à la requête initiale mais aussi des explications sur les parties de la requête réellement responsables de cet échec. L'aspect explication est illustré par l'identification des αMFS et les requêtes alternatives sont obtenues par le calcul des αXSS de la requête initiale. Nous avons d'abord défini les conditions nécessaires pour que l'algorithme de nos travaux, 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 à NLBA, appelées ascendante, descendante et hybride, qui considèrent des seuils α dans des ordres différents. Nous avons effectué une mise en œuvre complète de ces algorithmes et montré expérimentalement sur différents jeux de données des bancs d'essai WatDiv et LUBM que nos approches sont plus performantes que la méthode de base.

Dans une seconde contribution, nous avons considéré le problème des réponses pléthoriques, problème dual au problème à réponses vides. Nous nous sommes efforcés de garder la même philosophie de traitement que dans le cas des réponses vides. C'est-à-dire, fournir une explication à ce type de réponse et identifier des requêtes alternatives dont les réponses sont de taille acceptable (i.e., $\leq$ K). Les premiers résultats obtenus s'inscrivent dans le contexte des BC traditionnelles. Le résultat le plus important concerne la définition des contreparties des MFS et des XSS dans le contexte du problème des réponses pléthoriques. La première contrepartie, nommée MFIS, permet d'identifier les parties de la requête réellement responsables de la pléthore de réponses. La contrepartie des XSS fournit un ensemble de sous requêtes alternatives ne conduisant

pas à un échec, i.e., ne retournant pas de réponses pléthoriques.

Perspectives

Le travail présenté dans ce manuscrit pourrait être poursuivi selon différentes directions à court et à moyen termes. Les pistes que nous avons identifiées comme prioritaires sont les suivantes.

— La première perspective concerne le traitement du problème des réponses vides. L'approche proposée est de nature automatique dans le sens où l'utilisateur n'est pas mis dans la boucle de construction de la solution. Nous comptons développer une autre approche de nature interactive où l'intervention de l'utilisateur serait requise. Par exemple, (i) pour aider l'utilisateur à ajuster son seuil de confiance et lui proposer à partir de quel seuil sa requête (ou une partie d'elle) n'est plus en échec ; (ii) pour aussi exprimer ses préférences sur les parties de la requête qu'il désire préserver dans les XSS qui lui seront retournées.

— À court terme, nous envisageons d'illustrer l'intérêt de nos approches proposées dans le contexte des réponses vides 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 par rapport aux autres approches lorsque les degrés de confiance ne sont pas générés aléatoirement ou lorsque de nombreux seuils de confiance sont considérés.

— Comme nous avons observé dans nos expérimentations qu'aucun de nos algorithmes n'offre les meilleures performances pour toutes les requêtes, 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 ces approches au contexte des données massives.

— Notre méthode, CardAlgorithme, exploite les cardinalités globales et locales à une classe de valeur [0-1] et [1-1]. Comme perspective, nous comptons profiter des règles concernant les prédicats de cardinalité [1-n] pour déduire le résultat d'une requête sans avoir à l'exécuter. Aussi, il serait intéressant d'utiliser d'autres type de cardinalités comme les "characteristic sets" [NM11]. Cette méthode n'a été appliquée qu'aux requêtes en étoile. Nous souhaitons identifier les adaptations nécessaires pour les requêtes en chaîne et les requêtes composites.

— Le traitement du problème des réponses pléthoriques n'a été abordé que dans le contexte des bases de connaissances traditionnelles. Il serait intéressant d'examiner si les définitions et les propriétés proposées sont préservées dans le cas des bases de connaissances incertaines. Notamment, la généralisation des concepts de MFIS et de XSS aux requêtes paramétrées par un seuil de confiance α.

— Enfin, à moyen terme, et pour rendre notre travail plus générique, il serait intéressant d'aborder les autres problèmes mentionnés dans le chapitre 2, comme le "Why-not" and le "Why-answer". Nous souhaitons notamment examiner si les notions de MFS et de XSS auraient du sens pour ce type de problème.

Bibliographie

[ABK+07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007. (Cité en page 35)

[Ack99] Russell Lincoln Ackoff. *Re-creating the corporation : a design of organizations for the 21st century*. New York : Oxford University Press, 1999. (Cité en page 22)

[AHÖD14] Güneş Aluç, Olaf Hartig, M Tamer Özsu, and Khuzaima Daudjee. Diversified Stress Testing of RDF Data Management Systems. In *ISWC'14*, pages 197–212, 2014. (Cité en page 17), (Cité en page 23), (Cité en page 32), (Cité en page 67), (Cité en page 86)

[AKJF99] M. N. Murty A. K. Jain and P. J. Flynn. Data clustering : a review. *ACM Comput. Surv*, page 264–323, 1999. (Cité en page 54)

[AS12] Engineering Amit Singhal, SVP. Introducing the knowledge graph: things, not strings. In *Inside Search Blog*, 2012. (Cité en page 35), (Cité en page 36)

[AVH04] Grigoris Antoniou and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004. (Cité en page 27)

[BB14] Volha Bryl and Christian Bizer. Learning conflict resolution strategies for cross-language wikipedia data fusion. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 1129–1134. ACM, 2014. (Cité en page 36)

[BBM05] Prasan Roy Bhuvan Bamba and Mukesh Mohania. Osqr : overlapping clustering of query results. *In CIKM '05 : Proceedings of the 14th ACM international conference on Information and knowledge management*, page 239–240, 2005. (Cité en page 54)

[BCMT13] Roi Blanco, Berkant Barla Cambazoglu, Peter Mika, and Nicolas Torzec. Entity recommendations in web search. In *International Semantic Web Conference*, pages 33–48. Springer, 2013. (Cité en page 35)

[BH98] Krishna Bharat and Monika R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. *In SIGIR '98 : Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval,*, page 104–111, 1998. (Cité en page 52)

[BHP06] P. Bosc, A. Hadjali, and O. Pivert. About Overabundant Answers to Flexible Queries. *11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems IPMU, Paris*, pages 2221–2228, 2006. (Cité en page 49)

[BHP08] Patrick Bosc, Allel Hadjali, and Olivier Pivert. Empty versus overabundant answers to flexible relational queries. *Fuzzy sets and systems*, 159(12):1450–1467, 2008. (Cité en page 43)

[BHPS10] Patrick Bosc, Allel Hadjali, Olivier Pivert, and Grégory Smits. Une approche fondée sur la corrélation entre prédicats pour le traitement des réponses pléthoriques. In *EGC*, pages 273–284, 2010. (Cité en page 50)

[BLFM+98] Tim Berners-Lee, Roy Fielding, Larry Masinter, et al. Uniform resource identifiers (uri): Generic syntax, 1998. (Cité en page 23)

[BLK+09] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia-a crystallization point for the web of data. *Web Semantics: science, services and agents on the world wide web*, 7(3):154–165, 2009. (Cité en page 35)

[BLW11] Lidong Bing, Wai Lam, and Tak-Lam Wong. Using query log and social tagging to refine queries based on latent topics. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 583–592. ACM, 2011. (Cité en page 46)

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst., 30(1-7)*, pages 107–117, 1998. (Cité en page 52)

[Bri07] Dan Brickley. Rdf vocabulary description language 1.0: Rdf schema w3c recommendation 10 february 2004. *http://www. w3. org/TR/2004/REC-rdf-schema-20040210/*, 2007. (Cité en page 21), (Cité en page 24)

[BYK11] Ziv Bar-Yossef and Naama Kraus. Context-sensitive query auto-completion. In *Proceedings of the 20th international conference on World wide web*, pages 107–116. ACM, 2011. (Cité en page 47)

[Cam14] Stéphane Campinas. Live SPARQL Auto-Completion. In *ISWC'14 (Posters & Demos)*, pages 477–480, 2014. (Cité en page 44)

[CBK+10] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010. (Cité en page 15), (Cité en page 31), (Cité en page 35), (Cité en page 36), (Cité en page 59)

[CCLB97] Tiziana Catarci, Maria F Costabile, Stefano Levialdi, and Carlo Batini. Visual query systems for databases: A survey. *Journal of Visual Languages & Computing*, 8(2):215–260, 1997. (Cité en page 46)

[CD03] Surajit Chaudhuri and Gautam Das. Automated ranking of database query results. *In CIDR*, page 888–899, 2003. (Cité en page 52)

[CDHW04] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, and Gerhard Weikum. Probabilistic ranking of database query results. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 888–899. VLDB Endowment, 2004. (Cité en page 51)

[CFPW14] Andrea Calí, Riccardo Frosini, Alexandra Poulovassilis, and PeterT. Wood. Flexible Querying for SPARQL. In *ODBASE'14*, pages 473–490, 2014. (Cité en page 44), (Cité en page 48)

[CJ09] Adriane Chapman and HV Jagadish. Why not? In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 523–534. ACM, 2009. (Cité en page 43)

[CNFM14] Davide Ceolin, Archana Nottamkandath, Wan Fokkink, and Valentina Maccatrozzo. Towards the definition of an ontology for trust in (web) data. In *URSW*, pages 73–78, 2014. (Cité en page 32)

[DGH+14a] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In *KDD'14*, pages 601–610, 2014. (Cité en page 15), (Cité en page 59)

[DGH+14b] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In *KDD'14*, pages 601–610, 2014. (Cité en page 31), (Cité en page 35), (Cité en page 36), (Cité en page 37)

[DJH+17] Ibrahim Dellal, Stéphane Jean, Allel Hadjali, Brice Chardin, and Mickaël Baron. On addressing the empty answer problem in uncertain knowledge bases. In *International Conference on Database and Expert Systems Applications*, pages 120–129. Springer, 2017. (Cité en page 17)

[DJH+18a] Ibrahim Dellal, Stéphane Jean, Allel Hadjali, Brice Chardin, and Mickaël Baron. Traitement coopératif des requêtes rdf dans le contexte des bases de connaissances incertaines. *Document numerique*, 21(1):9–35, 2018. (Cité en page 17)

[DJH+18b] Ibrahim Dellal, Stéphane Jean, Allel Hadjali, Brice Chardin, and Mickaël Baron. Traitement coopératif des requêtes rdf dans le contexte des bases de connaissances incertaines. In *36e édition INFormatique des ORganisations et Systèmes d'Information et de Décision*, pages 277–292, 2018. (Cité en page 17)

[DJH+19] Ibrahim Dellal, Stéphane Jean, Allel Hadjali, Brice Chardin, and Mickaël Baron. Query answering over uncertain rdf knowledge bases: explain and obviate unsuccessful query results. *Knowledge and Information Systems*, pages 1–33, 2019. (Cité en page 17)

[DLT+13a] Omkar Deshpande, Digvijay S Lamba, Michel Tourn, Sanjib Das, Sri Subramaniam, Anand Rajaraman, Venky Harinarayan, and AnHai Doan. Building, maintaining, and using knowledge bases: a report from the trenches. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1209–1220. ACM, 2013. (Cité en page 15), (Cité en page 59)

[DLT+13b] Omkar Deshpande, Digvijay S Lamba, Michel Tourn, Sanjib Das, Sri Subramaniam, Anand Rajaraman, Venky Harinarayan, and AnHai Doan. Building, maintaining, and using knowledge bases: a report from the trenches. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1209–1220. ACM, 2013. (Cité en page 35)

[DP+98] Thomas H Davenport, Laurence Prusak, et al. *Working knowledge: How organizations manage what they know*. Harvard Business Press, 1998. (Cité en page 22)

[DSWD09] Peter Dolog, Heiner Stuckenschmidt, Holger Wache, and Jörg Diederich. Relaxing RDF queries based on user and domain preferences. *Journal of Intelligent Information Systems (JIIS)*, 33(3):239–260, 2009. (Cité en page 44), (Cité en page 48)

[ERW11] Shady Elbassuoni, Maya Ramanath, and Gerhard Weikum. Query Relaxation for Entity-Relationship Search. In *ESWC'11*, pages 62–76, 2011. (Cité en page 44), (Cité en page 48)

[FG15] Valeria Fionda and Gianluigi Greco. Trust models for rdf data: semantics and complexity. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. (Cité en page 32)

[FJH14] Geraud Fokou, Stephane Jean, and Allel Hadjali. Endowing Semantic Query Languages with Advanced Relaxation Capabilities. In *ISMIS'14*, pages 512–517, 2014. (Cité en page 44), (Cité en page 48)

[FJHB15] Géraud Fokou, Stéphane Jean, Allel Hadjali, and Mickaël Baron. Cooperative techniques for SPARQL query relaxation in RDF databases. In *ESWC'15*, pages 237–252, 2015. (Cité en page 48)

[FJHB16] Géraud Fokou, Stéphane Jean, Allel HadjAli, and Mickaël Baron. RDF Query Relaxation Strategies Based on Failure Causes. In *ESWC'16*, pages 439–454, 2016. (Cité en page 16), (Cité en page 44), (Cité en page 48), (Cité en page 49)

[FJHB17] Géraud Fokou, Stéphane Jean, Allel Hadjali, and Mickaël Baron. Handling Failing RDF Queries: From Diagnosis to Relaxation. *Knowledge and Information Systems (KAIS)*, 50(1), 2017. (Cité en page 17), (Cité en page 48), (Cité en page 49), (Cité en page 59), (Cité en page 61), (Cité en page 62), (Cité en page 66), (Cité en page 69)

[FVHH+01] Dieter Fensel, Frank Van Harmelen, Ian Horrocks, Deborah L McGuinness, and Peter F Patel-Schneider. Oil: An ontology infrastructure for the semantic web. *IEEE intelligent systems*, 16(2):38–45, 2001. (Cité en page 27)

[GB01] P.R. Gamble and J. Blackwell. *Knowledge Management: A State of the Art Guide*. Kogan Page Series. Kogan Page, 2001. (Cité en page 22)

[GC14] Inc Gavin Carothers, Lex Machina. Rdf 1.1 n-quads. *W3C Recommendation*, 2014. (Cité en page 86)

[GKM+03] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharm. Discovering All Most Specific Sentences. *ACM Trans. on Database Systems*, 28(2):140–174, 2003. (Cité en page 49)

[God97] Parke Godfrey. Minimization in Cooperative Response to Failing Database Queries. *International Journal of Cooperative Information Systems*, 6(2):95–149, 1997. (Cité en page 48), (Cité en page 49), (Cité en page 82)

[GPH05] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2):158 – 182, 2005. Selcted Papers from the International Semantic Web Conference, 2004. (Cité en page 112)

[GS14] Yves Raimond Guus Schreiber. Rdf 1.1 primer. In *W3C recommendation*, 2014. (Cité en page 91)

[GTHS15] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast Rule Mining in Ontological Knowledge Bases with AMIE+. *VLDB Journal*, 24(6):707–730, 2015. (Cité en page 44)

[Har09] Olaf Hartig. Querying Trust in RDF Data with tSPARQL. In *ESWC 2009*, 2009. (Cité en page 15), (Cité en page 21), (Cité en page 32), (Cité en page 34), (Cité en page 59)

[HLZ12] Hai Huang, Chengfei Liu, and Xiaofang Zhou. Approximating query answering on RDF databases. *Journal of the World Wide Web: Internet and Web Information Systems (WWW)*, 15(1):89–114, 2012. (Cité en page 44), (Cité en page 48)

[HM00] James Hendler and Deborah L McGuinness. The darpa agent markup language. *IEEE Intelligent systems*, 15(6):67–73, 2000. (Cité en page 27)

[HMPS12] Aidan Hogan, Marc Mellotte, Gavin Powell, and Dafni Stampouli. Towards Fuzzy Query-relaxation for RDF. In *ESWC'12*, pages 687–702, 2012. (Cité en page 44), (Cité en page 48)

[HPS14] Patrick J Hayes and Peter F Patel-Schneider. Rdf 1.1 semantics. w3c recommendation, february 2014. *World Wide Web Consortium. Retrieved from https://www.w3. org/TR/2014/REC-rdf11-mt-20140225*, 2014. (Cité en page 13), (Cité en page 26)

[HPW09] Carlos A. Hurtado, Alexandra Poulovassilis, and Peter T. Wood. Ranking Approximate Answers to Semantic Web Queries. In *ESWC'09*, pages 263–277, 2009. (Cité en page 44), (Cité en page 48)

[HSBW13] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. *Artificial Intelligence*, 194:28–61, 2013. (Cité en page 59), (Cité en page 91)

[IFIS08] George Beskales Ihab F. Ilyas and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, pages 1–58, 2008. (Cité en page 97)

[ILZ13] Md Saiful Islam, Chengfei Liu, and Rui Zhou. A framework for query refinement with user feedback. *Journal of Systems and Software*, 86(6):1580–1595, 2013. (Cité en page 43), (Cité en page 45)

[Jan09] Dietmar Jannach. Fast Computation of Query Relaxations for Knowledge-based Recommenders. *AI Communications*, 22(4):235–248, 2009. (Cité en page 49)

[JKCC14] Jyun-Yu Jiang, Yen-Yu Ke, Pao-Yu Chien, and Pu-Jen Cheng. Learning user reformulation behavior for query auto-completion. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 445–454. ACM, 2014. (Cité en page 47)

[JKL+14] Nandish Jayaram, Arijit Khan, Chengkai Li, Xifeng Yan, and Ramez Elmasri. Towards a query-by-example system for knowledge graphs. In *Proceedings of Workshop on GRAph Data management Experiences and Systems*, pages 1–6. ACM, 2014. (Cité en page 48)

[JKL+15] Nandish Jayaram, Arijit Khan, Chengkai Li, Xifeng Yan, and Ramez Elmasri. Querying knowledge graphs by example entity tuples. *IEEE Transactions on Knowledge and Data Engineering*, 27(10):2797–2811, 2015. (Cité en page 48)

[JR71] N. Jardine and C. J. Van Rijsbergen. The use of hierarchical clustering in information retrieval. *Information Storage and Retrieval.*, pages 217–240, 1971. (Cité en page 54)

[Kap82] S Jerrold Kaplan. Cooperative responses from a portable natural language query system. *Artificial Intelligence*, 19(2):165–187, 1982. (Cité en page 43)

[KCwH04] Surajit Chaudhuri Kaushik Chakrabarti and Seung won Hwang. Automatic categorization of query results. *In SIGMOD '04 : Proceedings of the 2004 ACM SIGMOD international conference on Management of data,*, page 755–766, 2004. (Cité en page 54)

[Ker15] Fadhela Kerdjoudj. *Gestion de l'incertitude dans le processus d'extraction de connaissances à partir de textes.* PhD thesis, Paris Est, 2015. (Cité en page 31)

[KI05] Georgia Koutrika and Yannis Ioannidis. Personalized queries under a generalized preference model. In *21st International Conference on Data Engineering (ICDE'05)*, pages 841–852. IEEE, 2005. (Cité en page 46)

[Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM,*, pages 604–632, 1999. (Cité en page 51), (Cité en page 52), (Cité en page 97)

[Kly04] Graham Klyne. Resource description framework (rdf): Concepts and abstract syntax. *http://www. w3. org/TR/2004/REC-rdf-concepts-20040210/*, 2004. (Cité en page 21), (Cité en page 23)

[Kob01] Alfred Kobsa. Generic user modeling systems. *User modeling and user-adapted interaction*, 11(1-2):49–63, 2001. (Cité en page 46)

[LIJ+15] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195, 2015. (Cité en page 15), (Cité en page 29), (Cité en page 59)

[LTYX07] Peng Li, Manghui Tu, I-Ling Yen, and Zhonghang Xia. Preference update for e-commerce applications: Model, language, and processing. *Electronic Commerce Research*, 7(1):17–44, 2007. (Cité en page 46)

[LWP00] Katia P. Sycara Leejay Wu, Christos Faloutsos and Terry R. Payne. Feedback adaptive loop for content-based retrieval. *In VLDB '00 : Proceedings of the 26th International Conference on Very Large Data Bases*, page 297–306, 2000. (Cité en page 52), (Cité en page 54)

[MBS13] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M Suchanek. Yago3: A knowledge base from multilingual wikipedias. 2013. (Cité en page 36)

[MHHD07] Knud Möller, Tom Heath, Siegfried Handschuh, and John Domingue. *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings*, chapter Recipes for Semantic Web Dog Food — The ESWC and ISWC Metadata Projects, pages 802–815. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. (Cité en page 29)

[Mil95] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995. (Cité en page 36)

[Mot96] Amihai Motro. Cooperative database systems. *International Journal of Intelligent Systems*, 11(10):717–731, 1996. (Cité en page 47)

[MPB14] Robert Meusel, Petar Petrovski, and Christian Bizer. The webdatacommons microdata, rdfa and microformat dataset series. In *International Semantic Web Conference*, pages 277–292. Springer, 2014. (Cité en page 36)

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. Introduction to Information Retrieval. *Cambridge University Press,New York, NY, USA*, 2008. (Cité en page 52)

[MT97] Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, Sep 1997. (Cité en page 49), (Cité en page 82)

[NJ07] Arnab Nandi and HV Jagadish. Assisted querying using instant-response interfaces. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1156–1158. ACM, 2007. (Cité en page 47)

[NM11] Thomas Neumann and Guido Moerkotte. Characteristic sets: Accurate cardinality estimation for rdf queries with multiple joins. In *2011 IEEE 27th International Conference on Data Engineering*, pages 984–994. IEEE, 2011. (Cité en page 103), (Cité en page 116)

[Oza94] J Ozawa. Cooperative answering with macro expression of a database. In *Fifth international conf. on Information processing and management of uncertainty in knowledge-based systems (IPMU)*, pages 17–22, 1994. (Cité en page 50)

[PAG09] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and Complexity of SPARQL. *ACM Transaction on Database Systems (TODS)*, 34(3):16:1–16:45, 2009. (Cité en page 30)

[PPEB15] Minh-Duc Pham, Linnea Passing, Orri Erling, and Peter A. Boncz. Deriving an Emergent Relational Schema from RDF Data. In *WWW'15*, pages 864–874, 2015. (Cité en page 44)

[PPNH07] Maja Pantic, Alex Pentland, Anton Nijholt, and Thomas S Huang. Human computing and machine understanding of human behavior: A survey. In *Artifical Intelligence for Human Computing*, pages 47–71. Springer, 2007. (Cité en page 46)

[PS15] Olivier Pivert and Grégory Smits. 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*, pages 499–517. Springer, 2015. (Cité en page 49)

[RGW16a] Miguel Rodríguez, Sean Goldberg, and Daisy Zhe Wang. Consensus maximization fusion of probabilistic information extractors. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1208–1216, 2016. (Cité en page 35)

[RGW16b] Miguel Rodríguez, Sean Goldberg, and Daisy Zhe Wang. Sigmakb: multiple probabilistic knowledge base fusion. *Proceedings of the VLDB Endowment*, 9(13):1577–1580, 2016. (Cité en page 35)

[RH17] Jennifer Rowley and Richard Hartley. *Organizing knowledge: an introduction to managing access to information*. Routledge, 2017. (Cité en page 22)

[RK13] Kuldeep BR Reddy and P Sreenivasa Kumar. Efficient Trust-Based Approximate SPARQL Querying of the Web of Linked Data. In *Uncertainty Reasoning for*

the Semantic Web II, pages 315–330. Springer, 2013. (Cité en page 48), (Cité en page 49)

[Rob97] S. E. Robertson. The probability ranking principle in ir. *In Readings in information retrieval, San Francisco, CA, USA,*, pages 281–286, 1997. (Cité en page 51), (Cité en page 97)

[Row07] Jennifer Rowley. The wisdom hierarchy: representations of the dikw hierarchy. *Journal of information science*, 33(2):163–180, 2007. (Cité en page 22)

[SAH+15] Muhammad Saleem, Muhammad Intizar Ali, Aidan Hogan, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. LSQ: The Linked SPARQL Queries Dataset. In *ISWC'15*, pages 261–269, 2015. (Cité en page 15), (Cité en page 59)

[SAM14] Silvio do Lago Pereira Samyr Abrahão Moises. Dealing with Empty and Overabundant Answers to Flexible Queries. *Journal of Data Analysis and Information Processing*, pages 12–18, 2014. (Cité en page 49)

[SCW04] Vagelis Hristidis Surajit Chaudhuri, Gautam Das and Gerhard Weikum. Probabilistic ranking of database query results. *In Proceedings of the Thirtieth international conference on Very large data bases*, page 888–899, 2004. (Cité en page 52)

[SDMB02] Avinash C. Ka Sean D. MacArthur, Carla E. Brodley and Lynn S. Broderick. Interactive content-based image retrieval using relevance feedback. *Comput. Vis. Image Underst.*, page 55–75, 2002. (Cité en page 52), (Cité en page 54)

[SH13] Andy Seaborne Steve Harris, Garlik. Sparql 1.1 query language (march 2013). *W3C Recommendation*, 2013. (Cité en page 15), (Cité en page 21), (Cité en page 28), (Cité en page 59)

[SKP11] Kostas Stefanidis, Georgia Koutrika, and Evaggelia Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Transactions on Database Systems (TODS)*, 36(3):19, 2011. (Cité en page 46)

[SKW07] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007. (Cité en page 15), (Cité en page 31), (Cité en page 35), (Cité en page 36)

[SLHA12] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. Linkedgeodata: A core for a web of spatial open data. *Semant. web*, 3(4):333–354, October 2012. (Cité en page 29)

[SLLP10] Umberto Straccia, Nuno Lopes, Gergely Lukacsy, and Axel Polleres. A general framework for representing and reasoning with annotated semantic web data. In *AAAI*, 2010. (Cité en page 91)

[SM86] Gerard Salton and Michael J. McGill. Introduction to Modern Information Retrieval. *McGraw-Hill, New York, NY, USA*, 1986. (Cité en page 51), (Cité en page 97)

[SNB+11] Satya S Sahoo, Vinh Nguyen, Olivier Bodenreider, Priti Parikh, Todd Minning, and Amit P Sheth. A unified framework for managing provenance information in translational research. In *BMC bioinformatics*, volume 12, page 461, 2011. (Cité en page 91)

[SRB+08] Paul R Smart, Alistair Russell, Dave Braines, Yannis Kalfoglou, Jie Bao, and Nigel R Shadbolt. A visual approach to semantic query design using a web-based graphical query designer. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 275–291. Springer, 2008. (Cité en page 46)

[SSST08] Bernhard Schueler, Sergej Sizov, Steffen Staab, and Duc Thanh Tran. Querying for meta knowledge. In *Proceedings of the 17th international conference on World Wide Web*, pages 625–634. ACM, 2008. (Cité en page 91)

[SWHL06a] Weifeng Su, Jiying Wang, Qiong Huang, and Fred Lochovsky. Query result ranking over e-commerce web databases. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 575–584. ACM, 2006. (Cité en page 51)

[SWHL06b] J. Su W, Wang, Q. Huang, and F Lochovsky. c. *In Proc. of CIKM*, 2006. (Cité en page 52), (Cité en page 53)

[Thi99] Robert J Thierauf. *Knowledge management systems for business*. Greenwood Publishing Group, 1999. (Cité en page 21), (Cité en page 22)

[TPR13] Dominik Tomaszuk, Karol Pak, and Henryk Rybiński. Trust in RDF graphs. In *ADBIS'13*, 2013. (Cité en page 15), (Cité en page 21), (Cité en page 32), (Cité en page 59)

[VBEZ10] Marco Viviani, Nadia Bennani, and Elod Egyed-Zsigmond. A survey on user modeling in multi-application environments. In *2010 Third International Conference on Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services*, pages 111–116. IEEE, 2010. (Cité en page 46)

[Vra12a] Denny Vrandečić. Wikidata: A new platform for collaborative data collection. In *Proceedings of the 21st international conference on world wide web*, pages 1063–1064. ACM, 2012. (Cité en page 35)

[Vra12b] Denny Vrandečić. Wikidata: A new platform for collaborative data collection. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, pages 1063–1064, New York, NY, USA, 2012. ACM. (Cité en page 15), (Cité en page 59)

[vRC75] C. J. van Rijsbergen and W. Bruce Croft. Document clustering : An evaluation of some experiments with the cranfield 1400 collection. *Inf. Process. Manage.*, pages 171–182, 1975. (Cité en page 54)

[VTBL16] Elena Vasilyeva, Maik Thiele, Christof Bornhövd, and Wolfgang Lehner. Answering "why empty?" and "why so many?" queries in graph databases. *Journal of Computer and System Sciences*, 82(1):3–22, 2016. (Cité en page 43)

[WW08] C Wesley W. Cooperative databasesystems. *John Wiley & Sons*, 2008. (Cité en page 46)

[XQW+13] Chuan Xiao, Jianbin Qin, Wei Wang, Yoshiharu Ishikawa, Koji Tsuda, and Kunihiko Sadakane. Efficient error-tolerant query autocompletion. *Proceedings of the VLDB Endowment*, 6(6):373–384, 2013. (Cité en page 47)

[Zlo75a] Moshé M Zloof. Query-by-example: the invocation and definition of tables and forms. In *Proceedings of the 1st International Conference on Very Large Data Bases*, pages 1–24. ACM, 1975. (Cité en page 47)

[Zlo75b] Moshé M. Zloof. Query-by-example : the invocation and definition of tables and forms. *In VLDB '75 : Proceedings of the 1st International Conference on Very Large Data Bases.*, page 1–24, 1975. (Cité en page 54)

Résumé

Avec l'émergence et la prolifération des applications du Web sémantique, de nombreuses et récentes larges bases de connaissances (BC) sont disponibles sur le Web. Ces BC contiennent des entités (nommées) et des faits sur ces entités. Elles contiennent également les classes sémantiques de ces entités et leurs liens mutuels. De plus, plusieurs BC peuvent être interconnectées au niveau entités, formant ainsi le noyau du Web des données liées (ou ouvertes). Une caractérisation essentielle de ces BC est qu'elles contiennent des millions à des billions de triplets RDF incertains. Les causes de cette incertitude sont diverses et multiples. Elle peut résulter de l'intégration de sources de données de différents niveaux de fiabilité ou elle peut être causée par des considérations de préservation de la confidentialité. Aussi, elle peut être due à des facteurs liés au manque d'informations, à la limitation des équipements de mesures ou à l'évolution d'informations. L'objectif de ce travail de thèse est d'améliorer l'ergonomie et la convivialité des systèmes modernes visant à exploiter des BC entachées d'incertitude. En particulier, ce travail propose des techniques coopératives et intelligentes aidant l'utilisateur dans ses prises de décisions quand ses recherches retournent des résultats insatisfaisants en termes de quantité ou de fiabilité.

Dans un premier temps, nous nous sommes intéressés au problème des requêtes RDF retournant un ensemble vide de réponses. Ce type de réponse est frustrant et ne sert pas les attentes de l'utilisateur. L'approche proposée pour le traitement de ce problème est guidée par la requête initiale et offre un double avantage : (i) elle permet de fournir une explication sur l'échec de la requête en identifiant les MFS (Minimal Failing Sub-queries) et, (ii) elle permet de calculer des requêtes alternatives appelées XSS (maXimal Succeeding Sub-queries), sémantiquement proches de la requête initiale et dont les réponses sont non-vides. Par ailleurs, d'un point de vue utilisateur, cette solution présente un niveau élevé de flexibilité dans le sens où plusieurs degrés d'incertitude peuvent être simultanément considérés.

Dans une seconde contribution, nous avons abordé l'étude du problème dual au problème cité ci-dessus, c'est-à-dire le cas des requêtes retournant un nombre trop élevé de réponses dans le contexte des données RDF. La solution préconisée vise à réduire cet ensemble de réponses pour permettre à l'utilisateur de les examiner. Des contreparties des MFS et des XSS ont été établies, ce qui a permis d'identifier, d'une part, les causes du problème et, d'autre part, des requêtes alternatives dont les résultats peuvent être directement et facilement exploitables à des fins de décision.

L'ensemble de nos propositions ont été validées par une série d'expérimentations portant sur différentes larges bases de connaissances en présence d'incertitude (WatDiv et LUBM). Nous avons aussi utilisé plusieurs Triplestores pour mener nos tests.

Mots-clés : Web sémantique, Bases de connaissances, Requêtes RDF, Incertitude, Réponses coopératives, Triplestores.

Abstract

In the era of digitilization, and with the emergence of several semantic Web applications, many new knowledge bases (KBs) are available on the Web. These KBs contain (named) entities and facts about these entities. They also contain the semantic classes of these entities and their mutual links. In addition, multiple KBs could be interconnected by their entities, forming the core of the linked data web. A distinctive feature of these KBs is that they contain millions to trillions of unreliable RDF triples. This uncertainty has multiple causes. It can result from the integration of data sources with various levels of intrinsic reliability or it can be caused by some considerations to preserve confidentiality. Furthermore, it may be due to factors related to the lack of information, the limits of measuring equipment or the evolution of information. The goal of this thesis is to improve the usability of modern systems aiming at exploiting uncertain KBs. In particular, this work proposes cooperative and intelligent techniques that could help the user in his decision-making when his query returns unsatisfactory results in terms of quantity or reliability.

First, we address the problem of failing RDF queries (i.e., queries that result in an empty set of responses). This type of response is frustrating and does not meet the user's expectations. The approach proposed to handle this problem is query-driven and offers a twofold advantage: (i) it provides the user with a rich explanation of the failure of his query by identifying the MFS (Minimal Failing Sub-queries) and (ii) it allows the computation of alternative queries called XSS (maXimal Succeeding Sub-queries), semantically close to the initial query, with non-empty answers. Moreover, from a user's point of view, this solution offers a high level of flexibility given that several degrees of uncertainty can be simultaneously considered.

In the second contribution, we study the dual problem to the above problem (i.e., queries whose execution results in a very large set of responses). Our solution aims at reducing this set of responses to enable their analysis by the user. Counterparts of MFS and XSS have been defined. They allow the identification, on the one hand, of the causes of the problem and, on the other hand, of alternative queries whose results are of reasonable size and therefore can be directly and easily used in the decision making process.

All our propositions have been validated with a set of experiments on different uncertain and large-scale knowledge bases (WatDiv and LUBM). We have also used several Triplestores to conduct our tests.

Keywords : Semantic Web, Knowledge Bases, RDF Queries, Uncertainty, Cooperative answering, Triplestores

---

Secteur de recherche : Informatique et applications

Laboratoire d'Informatique et d'Automatique pour les Systèmes
Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Téléport 2 – 1 avenue Clément Ader – BP 40109 – 86961 FUTUROSCOPE CHASSENEUIL CEDEX
Tél : 05.49.49.80.63 – Fax : 05.49.49.80.64