Traitement coopératif du problème des réponses pléthoriques dans les bases de connaissances RDF

Louise Parkin
louise.parkin@ensma.fr
LIAS, ISAE-ENSMA
Chasseneuil-du-Poitou, France

Ibrahim Dellal
ibrahim.dellal@ensma.fr
LIAS, ISAE-ENSMA
Chasseneuil-du-Poitou, France

Brice Chardin
brice.chardin@ensma.fr
LIAS, ISAE-ENSMA
Chasseneuil-du-Poitou, France

Stéphane Jean
stephane.jean@ensma.fr
LIAS, ISAE-ENSMA
Chasseneuil-du-Poitou, France

Allel Hadjali
allel.hadjali@ensma.fr
LIAS, ISAE-ENSMA
Chasseneuil-du-Poitou, France

RÉSUMÉ

Les utilisateurs d'une base de connaissances sont confrontés à un grand volume de données dont ils peuvent ignorer la structure sous-jacente. Ainsi, ils peuvent commettre des erreurs dans la formulation de leurs requêtes et obtenir des réponses non satisfaisantes. Nous nous intéressons ici au cas particulier du problème des réponses pléthoriques, où une requête produit beaucoup plus de résultats que n'attendait l'utilisateur. L'approche la plus connue pour traiter ce problème, la méthode dite top-K, consiste à classer les résultats pour ne retourner que les meilleures réponses. Cependant, si la requête comporte de mauvaises préconceptions, cette stratégie ne règle pas la source du problème et donc ne constitue pas une solution satisfaisante. Nous proposons donc une nouvelle méthode coopérative, permettant aux utilisateurs de comprendre l'origine des réponses pléthoriques de leur requête. Pour cela nous fournissons deux informations : (i) les parties minimales de la requête entraînant des réponses pléthoriques, et (ii) les parties maximales de la requête dont les réponses ne sont pas pléthoriques. Pour identifier ces deux informations, nous proposons deux algorithmes et montrons leur efficacité par rapport à une méthode naïve en utilisant des données synthétiques et réelles.

1 INTRODUCTION

Le développement du Web sémantique a permis la création de multiples bases de connaissances (BC) académiques et industrielles. Les BC stockent l'information sous forme de triplets RDF (sujet, prédicat, objet) et sont interrogées via le langage SPARQL [1]. Un utilisateur d'une BC a rarement une connaissance approfondie des données manipulées, ce qui peut le conduire à formuler des requêtes basées sur des préconceptions erronées et donnant ainsi des réponses insatisfaisantes. Les différents types de réponses insatisfaisantes sont : les réponses vides, les réponses pléthoriques, les réponses insuffisantes, l'absence d'une réponse attendue ou la présence d'une réponse inattendue. Nous nous intéressons ici au deuxième problème, où l'utilisateur reçoit un nombre trop important de réponses et ne peut pas en extraire des informations pertinentes. On parle de réponses pléthoriques lorsque le nombre de réponses dépasse un seuil K prédéfini.

L'approche la plus connue pour ce problème, la méthode top-K, repose sur un classement des réponses et une sélection des K premiers résultats. Si cette stratégie limite le nombre de réponses à K, elle n'agit pas au niveau de la requête, et ne peut donc pas traiter les problèmes que celle-ci peut présenter. Aussi, nous nous basons sur les solutions proposées dans le cadre du problème des réponses vides [2] pour fournir deux notions coopératives qui aideront les utilisateurs à comprendre l'origine des réponses pléthoriques. Nous fournissons d'abord des causes d'échec, appelées MFIS (*Minimal Failure Inducing Subqueries*). Ce sont les plus petites parties de la requête qui produisent des réponses pléthoriques sur lesquelles l'utilisateur devra focaliser son attention afin de modifier sa requête. Nous fournissons également des requêtes alternatives, appelées XSS (*maXimal Succeeding Subqueries*). Ce sont les plus grandes parties de la requête qui produisent des réponses non pléthoriques. Elles peuvent être utilisées à la place de la requête originale avec une garantie qu'un nombre de réponses n'excédant pas K sera retourné. A partir d'un algorithme naïf nécessitant l'exécution d'un nombre exponentiel de requêtes, nous proposons trois améliorations qui exploitent des propriétés identifiées sur les requêtes ou les données manipulées. Nous montrons leur intérêt expérimentalement en utilisant des données et requêtes générées avec le banc d'essai WatDiv [3], ainsi que des données et requêtes réelles provenant respectivement de DBpedia et du projet Linked SPARQL Queries Dataset [4].

2 ÉTAT DE L'ART

Les approches existantes pour le problème des réponses pléthoriques sont de deux types : les approches orientées données et les approches orientées requêtes. Les méthodes orientées données, telles que les approches top-K [5] ou les stratégies de groupement [6, 7] considèrent que la requête est correcte et cherchent à présenter les résultats de façon synthétique. Comme les méthodes orientées données ne s'intéressent pas à la requête, elles ne sont pas adaptées dans les situations où la requête pose problème.

A l'inverse, les méthodes orientées requêtes considèrent que la requête est à l'origine des réponses pléthoriques et proposent de la modifier. Cette modification peut se faire en ajoutant [8], supprimant [9] ou modifiant les patrons de triplets présents dans la requête [10, 11]. Cependant, les solutions existantes le font sans

chercher à identifier les causes des réponses pléthoriques. C'est l'objet de nos travaux.

3 DÉFINITION ET CALCUL DES MFIS ET XSS

Les sous-requêtes d'une requête conjonctive $Q = t_1 \wedge \cdots \wedge t_n$ sont les requêtes $Q' = t_i \wedge \cdots \wedge t_j$ où $\{t_i, \cdots, t_j\} \subseteq \{t_1, \cdots, t_n\}$, on dit alors que Q est une super-requête de Q'. Dans le cadre des réponses pléthoriques, une requête échoue si elle produit plus que K réponses, dans le cas contraire elle réussit. On appelle FIS (*Failure Inducing Subquery*) une sous-requête de la requête initiale qui échoue et dont toutes les super-requêtes échouent. Les MFIS (*Minimal FIS*) sont alors les FIS dont aucune sous-requête n'est une FIS. Les XSS (*maXimal Succeeding Subqueries*) sont les requêtes qui réussissent dont toutes les super-requêtes échouent.

Pour identifier les MFIS et XSS, une méthode naïve, appelée Base consiste à exécuter l'ensemble des sous-requêtes de la requête initiale. Pour une requête avec n patrons de triplets, cela implique d'exécuter $2^n - 1$ sous-requêtes. Pour réduire le temps de calcul des MFIS et XSS, nous introduisons des propriétés permettant de diminuer le nombre de requêtes à exécuter. Des définitions de MFIS et XSS, on déduit qu'une sous-requête d'une requête qui réussit ne peut être ni MFIS ni XSS, donc nous n'avons pas besoin de les exécuter. Une première amélioration, Bfs, exploite cette propriété.

La deuxième propriété que nous proposons affirme que si on retire à une requête $Q \wedge t$ un patron de triplet t en conservant toutes les variables, alors la nouvelle requête Q ne peut pas avoir moins de réponses que $Q \wedge t$. Ainsi, si $Q \wedge t$ échoue, on en déduit l'échec de Q sans l'exécuter. L'ajout de cette propriété constitue un nouvel algorithme, Var.

La dernière amélioration exploite une propriété qui implique à la fois les requêtes et les données. Nous introduisons le concept de cardinalité maximale d'un prédicat [12], qui décrit le nombre maximal d'occurrences d'un prédicat pour tous les sujets de la BC. La propriété associée indique que si on retire à une requête $Q \wedge t$ un patron de triplet t dont le prédicat a une cardinalité maximale de 1, alors la nouvelle requête Q ne peut pas avoir moins de réponses que $Q \wedge t$. Ainsi, si $Q \wedge t$ échoue, on en déduit l'échec de Q sans l'exécuter. En ajoutant cette propriété aux précédentes, on obtient notre dernier algorithme, Full.

Si les deux premières propriétés, et donc l'algorithme Var, peuvent s'appliquer à toutes les requêtes, la dernière nécessite de disposer d'une information supplémentaire : les cardinalités des prédicats de la BC. Comme il peut être difficile de maintenir de telles informations sur des bases de connaissances régulièrement modifiées, l'algorithme Full ne sera pas toujours applicable.

4 RÉSULTATS EXPÉRIMENTAUX

Nous avons évalué la performance de nos quatre algorithmes avec deux expérimentations réalisées sur le triplestore JenaTDB, en fixant le seuil des réponses pléthoriques à K=100. La première expérimentation utilise des données (11M triplets) et requêtes générées avec le banc d'essai WatDiv. Nous avons utilisé 21 requêtes, de forme étoile, chaîne et composite contenant 4 à 12 patrons de triplet. Les temps d'exécution mesurés pour les requêtes en étoile sont fournis en figure 1, avec le nombre de requêtes exécutées par chaque algorithme en table 1. On observe la même allure pour les autres formes de requêtes. Les algorithmes Bfs, Var et Full économisent respectivement 2%, 65% et 83% du temps d'exécution de l'algorithme naïf Base. On peut noter que même si l'algorithme Bfs peut exécuter la moitié du nombre de requêtes de Base, leurs temps d'exécution restent proches.

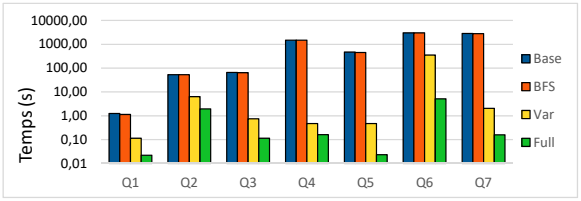

Figure 1 : Temps d'exécution (requêtes en étoile - WatDiv)

	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Base	15	31	63	127	255	511	1023
BFS	15	31	33	75	129	511	513
Var	8	16	9	39	65	255	257
Full	2	4	2	12	3	8	9

Table 1 : Nombre de requêtes exécutées (requêtes en étoile - WatDiv)

La seconde expérimentation se base sur une BC réelle, DBpedia, et des logs de requêtes réelles. Le comportement est proche de WatDiv, les algorithmes Bfs, Var et Full économisent respectivement 14%, 78% et 78% du temps d'exécution de l'algorithme Base. Dans ce cas, Full n'est pas significativement plus rapide que Var. Cela s'explique par les cardinalités de DBpedia qui permettent rarement d'exploiter la propriété supplémentaire de Full. Notre expérimentation montre que l'algorithme Var permet d'obtenir les MFIS et XSS dans un temps raisonnable, et que, selon les données manipulées, l'exploitation des cardinalités peut offrir une amélioration supplémentaire de performance.

RÉFÉRENCES

[1] Harris, S., Seaborne, A. : Sparql 1.1 query language. W3C Recommendation (2013)
[2] Godfrey, P. : Minimization in Cooperative Response to Failing Database Queries. International Journal of Cooperative Information Systems **6**(2) (1997) 95–149
[3] Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K. : Diversified stress testing of rdf data management systems. In : ISWC'14, Springer (2014) 197–212
[4] Saleem, M., Ali, M.I., Hogan, A., Mehmood, Q., Ngomo, A.N. : LSQ : The Linked SPARQL Queries Dataset. In : ISWC'15. (2015) 261–269
[5] Ilyas, I.F., Beskales, G., Soliman, M.A. : A survey of top-k query processing techniques in relational database systems. ACM Comput. Surv. **40**(4) (2008)
[6] Chakrabarti, K., Chaudhuri, S., Hwang, S.w. : Automatic categorization of query results. In : SIGMOD'04. (2004) 755–766
[7] Ozawa, J., Yamada, K. : Discovery of global knowledge in a database for cooperative answering. In : IEEE'95. Volume 2. (1995) 849–854
[8] Bosc, P., Hadjali, A., Pivert, O., Smits, G. : Une approche fondée sur la corrélation entre prédicats pour le traitement des réponses pléthoriques. In : EGC'10. 273–284
[9] Vasilyeva, E., Thiele, M., Bornhövd, C., Lehner, W. : Answering "why empty ?" and "why so many ?" queries in graph databases. Journal of Computer and System Sciences **82**(1) (2016) 3–22
[10] Bosc, P., HadjAli, A., Pivert, O. : About overabundant answers to flexible queries. In : IPMU'06. Volume 6. (2006) 2221–2228
[11] Moises, S.A., Pereira, S.d.L. : Dealing with empty and overabundant answers to flexible queries. Journal of Data Analysis and Inf. Proc. (2014) 12–18
[12] Dellal, I. : Management and Exploitation of Large and Uncertain Knowledge Bases. PhD thesis, ISAE-ENSMA - Poitiers (2019)