Interrogation et Analyse Efficiente des Données du Web Sémantique

Synthèse de la thèse en français

Introduction

L'utilité et la pertinence des données se trouvent dans l'information qui peut en être extraite. Le taux élevé de publication des données et leur complexité accrue, par exemple dans le cas des données du Web sémantique auto-descriptives et hétérogènes, motivent l'intérêt de techniques efficaces pour la manipulation de données.

L'objectif de cette thèse est double :

- Premièrement, d'utiliser la technologie mature de gestion de données relationnelles existante pour répondre efficacement à des requêtes sur les données soumises à des contraintes sémantiques ;
- Deuxièmement, de formaliser les procédures pour l'analyse puissante de ces données.

Ensuite, nous décrivons les problèmes de recherche traités dans cette thèse et nous présentons nos solutions.

Contexte

Nous commençons par fournir les informations de base nécessaires pour suivre les problèmes soulevés dans cette thèse et les solutions proposées. Tout d'abord, nous décrivons le Resource Description Framework (RDF), un modèle de données basé sur les graphes recommandé par le W3C pour l'échange de données sur le Web. Ensuite, nous présentons le standard du W3C pour l'interrogation de données RDF, notamment le SPARQL Protocol and RDF Query Language (SPARQL). En particulier, nous utilisons le sous-ensemble bien connu de SPARQL constitué de (unions de) requêtes sur de motifs de graphe élémentaire (BGP), aussi connu comme requêtes conjonctives SPARQL.

Graphes RDF. Un *graphe RDF* est un ensemble de *triplets* de la forme `s p o`. Un triplet dit que son *sujet* `s` est décrit par la *propriété* `p`, et que la valeur de cette propriété est *l'objet* `o`.

La figure 1 (haut) montre comment utiliser des triplets pour décrire des ressources, c'est-à-dire, pour exprimer des assertions de classe (relation unaire) et de propriété (relation binaire). Le standard RDF [17] fournit un ensemble de classes et propriétés prédéfinies via les *espaces de noms* normalisés rdf: et rdfs:. Nous utilisons ces espaces de noms exactement pour ces classes et propriétés, e.g., la propriété rdf:type permet d'indiquer à quelles *classes* des ressources appartiennent.

RDF Schema (RDFS) est une extension importante de RDF dont le but est d'enrichir les descriptions de ressources dans les graphes. Un schéma RDF permet de déclarer des *contraintes sémantiques* entre les classes et les propriétés utilisées dans ces graphes. La figure 1 (bas) montre les contraintes autorisées et comment les déclarer ; le *domaine* et le *range* désignent respectivement le premier et le second attribut de toute propriété.

Les contraintes RDFS (figure 1) sont interprétées sous l'hypothèse du monde ouvert (OWA) [2], qui implique que certains faits peuvent être vrais même s'ils ne sont pas explicitement présents dans un graphe RDF. Par exemple, étant donné deux relations R_1 et R_2, l'interprétation OWA de la contrainte $R_1 \subseteq R_2$ est : tout n-uplet t dans la relation R_1 *est considéré comme étant aussi dans la relation* R_2 (la contrainte d'inclusion *propage* t *à* R_2).

Assertion	Triplet	Notation relationnelle
Classe	s rdf:type o	o(s)
Propriété	s p o	p(s, o)

Contrainte	Triplet	Interprétation OWA
Sous-classe	s rdfs:subClassOf o	$\mathtt{s} \subseteq \mathtt{o}$
Sous-propriété	s rdfs:subPropertyOf o	$\mathtt{s} \subseteq \mathtt{o}$
Typage de domaine	s rdfs:domain o	$\Pi_{\text{domain}}(\mathtt{s}) \subseteq \mathtt{o}$
Typage de range	s rdfs:range o	$\Pi_{\text{range}}(\mathtt{s}) \subseteq \mathtt{o}$

Figure 1: Déclarations RDF (en haut) & RDFS (en bas).

Entailment. Une caractéristique importante de RDF est la modélisation de *triplets implicites* : ils sont considérés comme faisant partie d'un graphe, même s'ils ne sont pas explicitement présents dans celui-ci. Le W3C nomme *RDF entailment* le mécanisme par lequel les triplets implicites sont dérivés (ou engendrés) à partir des triplets explicites d'un graphe et de *règles d'entailment*. Nous notons $\vdash^i_{\text{RDF}}$ un *entailment immédiat*, i.e., la dérivation de triplets par l'application d'une règle d'entailment. Plus généralement, un triplet $\mathtt{s\ p\ o}$ est engendré par un graphe $\mathtt{G}$, noté $\mathtt{G} \vdash_{\text{RDF}} \mathtt{s\ p\ o}$ si et seulement si il existe une séquence d'entailments immédiats menant de $\mathtt{G}$ à $\mathtt{s\ p\ o}$ (o à chaque pas de la séquence, les triplets précédemment engendrés sont aussi pris en compte).

Saturation de graphe. Les règles d'entailment permettent de définir la *saturation* (finie) $\mathtt{G}^\infty$ d'un graphe $\mathtt{G}$, qui est un graphe RDF, définie comme le point fixe obtenu en appliquant de façon exhaustive les règles d'entailment immédiat $\vdash^i_{\text{RDF}}$ sur $\mathtt{G}$.

La saturation d'un graphe RDF est unique (au renommage des nœuds blancs près) et ne contient plus de triplets implicites (tous ont été explicités par saturation). Les triplets engendrés par un graphe $\mathtt{G}$ et la saturation de ce graphe ont un lien évident : $\mathtt{G} \vdash_{\text{RDF}} \mathtt{s\ p\ o}$ si et seulement si $\mathtt{s\ p\ o} \in \mathtt{G}^\infty$.

Il est important de noter que l'entailment fait partie de la norme RDF. Par conséquent, en RDF, tout graphe $\mathtt{G}$ est (sémantiquement) équivalent à sa saturation $\mathtt{G}^\infty$.

Requêtes Basic Graph Pattern. Nous considérons un sous-langage de SPARQL modélisant des (unions de) *requêtes conjonctives*. Une requête conjonctive est définie par un *Basic Graph Pattern* (BGP), c'est-à-dire un ensemble d'*atomes triplets* ou plus simplement de triplets. Chaque triplet a un sujet, une propriété et un objet. Les sujets et les propriétés peuvent être des URIs, des nœuds blancs ou des *variables* ; les objets peuvent aussi être des littéraux.

On notera que l'évaluation *assimile les nœuds blancs d'une requête à des variables non distinguées.*

L'évaluation d'une requête q sur un graphe $\mathtt{G}$ n'utilise que les triplets explicitement présents dans $\mathtt{G}$, donc conduit à un ensemble incomplet de réponses dans le cas général. L'*ensemble (complet) de réponses* à q sur $\mathtt{G}$ est obtenu par l'évaluation de q sur $\mathtt{G}^\infty$, notée $q(\mathtt{G}^\infty)$, i.e., sur la saturation de $\mathtt{G}$.

Répondre aux requêtes dans les bases de données RDF

La première partie se concentre sur l'apport de réponse aux requêtes sur les données soumises à des contraintes RDFS, stockées dans un système de gestion de données relationnelles. L'information implicite, résultant du raisonnement RDF est nécessaire pour répondre correctement à ces requêtes. Nous introduisons le fragment des bases de données RDF, allant au-delà de l'expressivité des fragments étudiés précédemment. Nous élaborons de nouvelles techniques pour répondre aux requêtes dans ce fragment, en étendant deux approches connues de manipulation de données sémantiques RDF, notamment par saturation de graphes et reformulation de requêtes. En particulier, nous considérons les mises à jour de graphe au sein de chaque approche

et proposerons un procédé incrémental de maintenance de saturation. Nous étudions expérimentalement les performances de nos techniques, pouvant être déployées au-dessus de tout moteur de gestion de données relationnelles.

Le fragment des bases de données RDF

Nous définissons le *fragment "Bases de données" (BD) de RDF* en *limitant* le RDF entailment (défini dans les spécifications [17]) *aux seules règles d'entailment dédiées aux schémas RDF*. Notamment, ceci est l'unique restriction imposée à RDF pour obtenir par le fragment BD. Allant au-delà de l'expressivité des fragments étudiés précédemment, le fragment BD permet la modélisation d'informations incomplètes (grâce aux nœuds blancs) et la non distinction entre les constantes et les classes/propriétés. Notre but est d'avoir une séparation claire entre le *schéma* et les *données*, comme dans la majorité des modèles de données, tout en restant le plus fidèle possible à RDF.

Nous appelons *base de données (BD) RDF* tout graphe RDF appartenant à notre fragment BD. Une BD `db` est un couple $\langle \mathtt{S}, \mathtt{D} \rangle$, où `S` et `D` sont deux ensembles disjoints de triplets. Les triplets de `S` sont uniquement des assertions RDFS. Nous appelons ces triplets le *schéma* de `db`. Les autres triplets sont des assertions RDF appartenant à `D`. Nous les appelons *l'instance* de `db`. Observons que `S` et `D` forment une partition de tout graphe RDF (tout triplet appartient seulement à l'un d'eux). Une façon équivalente de dire les choses est que notre fragment BD n'impose aucune restriction sur les graphes RDF.

En général, les requêtes "utilisateur" peuvent utiliser les deux parties d'une BD (schéma et instance). Toutefois, la séparation entre schéma et instance, correspondant à la vision usuelle qu'ont les utilisateurs d'une base de données ou de connaissances, mène souvent à spécifier des requêtes sur le schéma seulement ou sur l'instance seulement. D'un point de vue "bases de données" (BD), les *requêtes sur l'instance* sont les plus courantes. Répondre à ces requêtes nécessite les triplets du schéma, puisque la saturation de la BD (nécessaire pour obtenir un ensemble complet de réponses) repose sur les triplets du schéma. D'un point de vue "Représentation des connaissances" (RC), les *requêtes sur le schéma* jouent aussi un râle important. Ces requêtes offrent un moyen d'explorer les relations entre classes et propriétés du schéma, y compris les relations implicites. Notre formalisme est assez général pour intégrer à la fois des requêtes sur l'instance typiques en BD, des requêtes sur le schéma typiques en RC, ainsi que des requêtes sur le schéma et l'instance.

Techniques de réponse aux requêtes

Nous étudions deux techniques pour répondre aux requêtes sur des BD RDF : l'une par saturation de la BD, l'autre par reformulation des requêtes.

Répondre par saturation revient à calculer l'ensemble des réponses à une requête exactement comme cet ensemble est formellement défini. La saturation de la BD est calculée (en utilisant les règles d'entailment autorisées), de sorte que l'ensemble de réponses à une requête sur la BD est obtenue par évaluation de la requête sur la BD saturée. L'avantage de cette technique est sa facilité de mise en œuvre. Ses inconvénients sont que la saturation nécessite du temps pour être calculé, de l'espace pour être stockée, et que celle-ci doit être recalculée lors de mises-à-jour.

Répondre par reformulation consiste à reformuler une requête q w.r.t. une BD `db` en une nouvelle requête q', de sorte que l'évaluation de q' sur la BD (originale) `db`, soit exactement l'ensemble de réponse de q sur `db`. L'avantage de cette technique est que la saturation n'a pas à être calculée. L'inconvénient est que chaque requête doit être reformulée, et que sa reformulation résulte généralement en une requête plus complexe à évaluer.

Dans la suite, nous nous intéressons à répondre aux requêtes par saturation et par reformulation, dans le cas des requêtes sur l'instance uniquement (les plus courantes).

Répondre aux requêtes par saturation

Notre technique de réponse aux requêtes par saturation se fonde sur notre algorithme `Saturate`, qui ne calcule que la partie pertinente de la saturation d'une BD afin de répondre aux requêtes sur l'instance.

Évaluer la requête originale sur cette saturation fournit alors l'ensemble exact de réponses.

`Saturate` repose sur un ensemble de règles de saturation. Les règles définissent un ensemble de transformations de BD de la forme $\frac{input}{output}$, où *input* et *output* sont des BD. Intuitivement, étant donnée une BD `db`, `Saturate(db)` applique exhaustivement les règles de saturation, sur `db` et les triplets engendrés successivement.

Maintenance de la saturation sur les mises à jour. Répondre aux requêtes par saturation est efficace au moment de la requête, car il suffit d'évaluer la requête initiale. Mais, la saturation doit être recalculé pour tenir compte de l'impact des mises à jour. L'objectif est de concevoir des algorithmes incrémentaux, qui n'ont pas à recalculer la saturation, mais juste à la modifier pour tenir compte de la mise à jour.

Nous étendons la notion précédente de saturation de base de données, de sorte qu'il devient un multi-ensemble $\texttt{Saturate}_+$, dans lequel un triple apparaît autant de fois que on peut en déduire. Pour toute base de données RDF `db` nous avons que $\texttt{Saturate}(\texttt{db}) = \texttt{ensemble}(\texttt{Saturate}_+(\texttt{db}))$.

En utilisant le saturation multi-ensemble, l'insertion d'un triple déjà dans la base de données, ou la suppression d'un triple qui n'est pas dans la base de données, ne nécessite pas de travaux. Sinon, l'insertion (suppression) d'un triple donnée (de l'instance ou le schéma) ajoute également aux (supprime de) la saturation actuelle tous les triplets de niveau de l'instance dont la dérivation utilise cette triple donnée.

Répondre aux requêtes par saturation. D'un point de vue pratique, répondre aux requêtes par saturation peut être déléguée à un SGBDR par stocker la saturation (ensemble ou multi-ensemble) dans un tableau et évaluer des requêtes en utilisant le moteur SGBDR.

Répondre aux requêtes par reformulation

Notre technique de réponse aux requêtes par reformulation se fonde sur notre algorithme `Reformulate`. Étant données une requête q et une BD `db`, $\texttt{Reformulate}(q, \texttt{db})$ reformule q en un ensemble de requêtes, tel que l'union des *évaluations non standard* (voir ci-dessous) de ces requêtes sur `db` produit $q(\texttt{db}^\infty)$, l'ensemble exact de réponses de la requête originale sur la BD.

`Reformulate` applique exhaustivement un ensemble de règles de reformulation, en partant d'une requête q et d'une BD `db`. Chaque règle définit une transformation de la forme $\frac{input}{output}$, où *input* est de la forme $\langle$ condition sur `db`, condition sur q $\rangle$ et *output* est une requête q'. Chacune des conditions de l'input, mais pas les deux, peut être vide. Intuitivement, chaque règle produit une nouvelle requête lorsque les conditions de son input sont satisfaite, l'une par la BD `db`, et l'autre par la requête (soit la requête originale q, soit une requête q' produite par une application précédente d'une règle). L'ensemble de toutes les requêtes générées par application des règles est le résultat de la reformulation de q w.r.t. `db`.

Répondre par reformulation de requête. Un prérequis pour toute technique de réponse aux requêtes par reformulation est que les requêtes obtenues par reformulation soient équivalentes ou incluses dans la requête originale (w.r.t. aux contraintes de la BD), sinon leur évaluation produirait des réponses erronées. Il apparaît que notre technique de reformulation ne respecte pas ce prérequis, si l'on considère les définitions précédemment fournies d'évaluation et d'ensembles de réponses à une requête sur une BD.

Le problème vient des nœuds blancs. Il y a un décalage entre la sémantique des nœuds blancs dans les requêtes et les raisons pour lesquelles ils sont introduits dans les requêtes produites par notre algorithme `Reformulate`. Pour mémoire, la sémantique d'un nœud blanc dans une requête sur une BD est celle d'une variable non distinguées. Toutefois, lorsque notre algorithme `Reformulate` introduit un nœud blanc dans une requête à partir d'une instanciation de variable ou un remplacement de triplet, il réfère précisément à ce nœud blanc particulier de la BD.

Évaluation et ensemble de réponses non standard d'une requête sur une BD. Pour résoudre le problème identifié ci-dessus, nous introduisons les *notions alternatives d'évaluation et d'ensemble de réponses*

d'une requête partiellement instanciée sur une BD. La différence fondamentale entre les définitions alternatives et les définitions standards concerne les nœuds blancs. L'évaluation d'une requête précédemment définie se fonde sur des assignations de tous les variables et nœuds blancs de la requête, à des valeurs dans la BD. En revanche, la définition alternative cherche seulement à assigner les variables de la requête ; les nœuds blancs ne sont pas touchés, comme les URIs et littéraux.

Pour répondre à une requête q sur une BD `db`, il suffit de (i) reformuler q w.r.t. `db` et (ii) évaluer chacune des requêtes produites par reformulation sur la BD *originale*, en utilisant l'évaluation *non standard*. En d'autres termes, la reformulation et l'évaluation non standard de requête permettent de calculer l'ensemble *standard* des réponses à une requête, *sans* saturer la BD.

Étude pratique

Nos expériences ont montré que `Saturate` et `Reformulate` peut être utilisé pour traiter les requêtes BGP de manire efficace en exploitant un SGBDR typique. Cependant, ils se comportent trs différemment en fonction de la sélectivité de la requête et de l'impact du schéma par le raisonnement.

En ce qui concerne les mises à jour, nous avons montré que la saturation peut être maintenue à un cot raisonnable pour les mises à jour au niveau de l'instance, tandis que les mises à jour au niveau du schéma sont beaucoup plus coteux. Les mises à jour, cependant, ont un faible impact sur la reformulation, faisant la reformulation appropriée pour des taux élevés de mise à jour. Lorsqu'on examine l'exécution répétée de requête, nous avons souligné un certain nombre de seuils déterminant lorsque la saturation est rentable; ces seuils sont fortement influencés par la taille de reformulation de requête et sa sélectivité. Alors que la saturation est le défaut dans de nombreuses plates-formes RDF, nos expériences démontrent l'intérêt pratique de répondre aux requêtes BGP par reformulation.

Les travaux connexes

Contrairement à d'autres travaux sur la saturation, notre technique de maintenance de saturation est basée sur le nombre de fois que des triplets sont déduits, ce qui facilite le stockage des données et la manipulation. Le travau ultérieur [16] emploie également l'utilisation de compteurs de dérivation pour faciliter la maintenance de la saturation lors de les suppressions de données. Contrairement à ce travail, nos algorithmes sont adaptés pour travailler avec le nombre total de différentes dérivations, afin d'éviter les calculs nécessaires pour distinguer les différentes voies d'implication.

Les algorithmes de reformulation de requête de [3, 6, 9, 10] sont des restrictions de notre `Reformulate`.

Dans [4], les requêtes SPARQL sont reformulées dans SPARQL imbriqué, soit une extension de SPARQL dans lequel les propriétés de triplets peuvent être des expressions régulires imbriquées. Même si ces requêtes reformulées imbriquées sont plus compacts, les requêtes que nous produisons sont plus pratiques, car leur évaluation peut être directement déléguée à un SGBDR, ou à un moteur RDF, même s'il n'a pas connaissance de raisonnement.

Analyse de données RDF

Avec le nombre croissant de jeux de données RDF disponibles, notamment via l'engouement pour les données ouvertes et liées, les besoins applicatifs sont en train d'évoluer. Par rapport à l'entreposage de données, nous avons identifié cinq besoins actuels, résumés comme suit : (i) la prise en compte de données *hétérogènes*, (ii) la gestion de plusieurs entités centrales, (iii) la prise en compte de la *sémantique des données RDF* lors de l'interrogation de l'entrepôt, (iv) la possibilité d'*interroger les relations entre entités* (i.e., interroger le schéma), (v) un choix flexible des dimensions d'agrégation de données.

La deuxième partie de cette thèse considère les nouvelles exigences pour les outils et méthodes d'analyse de données, issues de l'évolution du Web sémantique. Nous revisitons intégralement les concepts et les outils fondamentaux pour l'analyse de données, dans le contexte de RDF. Nous proposons le premier cadre formel pour l'analyse de données RDF de type entrepôt de données. Notamment, nous définissons des schémas analytiques

adaptés aux graphes RDF hétérogènes à sémantique riche, des requêtes analytiques qui (au-delà de cubes relationnels) permettent l'interrogation flexible des données et schémas, ainsi que des opérations d'agrégation puissantes de type OLAP. Des expériences sur une plateforme entièrement implémentée démontrent l'intérêt pratique de notre approche.

Schémas analytiques et instances

Nous modélisons un schéma d'analyse de graphes RDF, nommé *schéma analytique* (*AnS*), par un graphe orienté et étiqueté. Selon les concepts bien connus dans le domaine des entrepôts de données, *chaque nœud de notre schéma analytique représente un ensemble de faits* qui pourront être analysés. De plus, *les faits représentés par un nœud du schéma analytique peuvent être analysés en utilisant (soit en tant que dimensions, soit en tant que mesures) les nœuds du schéma atteignables depuis ce nœud*. Ceci rend notre modèle de schéma analytique bien plus général que dans le contexte des entrepôts de données traditionnels, où les faits (au centre d'un schéma dit "en étoile" ou "en flocon") sont analysés selon un ensemble fixe de dimensions et de mesures.

D'une perspective Web Sémantique, un nœud du schéma analytique correspond à une classe RDF, tandis qu'un arc du schéma analytique correspond à une propriété RDF. Les instances de ces classes et propriétés, modélisant le contenu de l'entrepôt de données qui doit être analysé par la suite, sont définies de façon *intensionnelle* dans le schéma, selon l'approche classique d'intégration de données connue sous le nom "Global As View" (GAV) [11].

Tout comme un schéma analytique définit (et délimite) les données à la disposition de l'analyste dans un scénario traditionnel d'entrepôt de données relationnelles, dans notre approche, *les classes et propriétés modélisées par un AnS sont les seules visibles pour la suite de l'analyse RDF*. En d'autres termes, les requêtes analytiques seront formulées dans les termes du schéma *AnS* et non pas sur la base de données (le graphe RDF) d'origine.

Les nœuds et arcs d'un schéma analytique définissent la perspective (ou *l'angle*) d'analyse d'un ensemble de données RDF. La *sémantique disjonctive d'un AnS*, est au cœur de notre notion d'entrepôt RDF. *Chaque nœud et chaque arc d'un AnS peuple l'instance par une requête BGP indépendante*, et l'union des triplets ainsi obtenue est l'instance $\mathcal{I}$ de *AnS*. Le fait de définir les nœuds et les arcs de l'*AnS* indépendemment les uns des autres est crucial pour qu'un entrepôt défini selon notre approche puisse :

- être lui-même un *graphe RDF* (et non pas un tableau, éventuellement comprenant de nombreuses valeurs nulles, qui apparatraient si on avait cherché à décomposer le graphe RDF dans un entrepôt relationnel). Ceci permet de répondre au besoin (*i*) du haut. Cela guarantit aussi que l'instance de l'*AnS* peut être *partagée, liée, et publiée* selon les meilleures pratiques actuelles du Web Sémantique ;
- bénéficier directement des mécanismes de *réponse aux requêtes SPARQL tenant compte de la sémantique*, fournis par les moteurs SPARQL. Ceci répond aussi à notre besoin de prise en compte de la sémantique (*iii*), ainsi que (*iv*) (la possibilité d'interroger le schéma, ce qui n'est pas possible de faire dans les entrepôts relationnels) ;
- fournir *autant de points d'entrée pour l'analyse qu'il y a de nœuds dans l'AnS*, en cohérence avec la nature flexible, décentralisée des graphes RDF (besoin (*ii*)). En conséquence, les requêtes d'agrégation sont très flexibles, c'est-à-dire qu'elle peuvent agréger une entité en relation avec une autre; par exemple, elles permettent de compter des restaurants proches de points touristiques importants (besoin (*v*)) ;
- *s'adapter facilement à des changements de l'AnS* (besoin (*ii*)) puisque des définitions de nœuds et/ou d'arcs peuvent être librement ajoutées à (ou retirés de) l'*AnS*, sans aucun impact sur les définitions d'autres nœuds ou arcs, ni sur leurs instances.

Requêtes analytiques

L'analyse des entrepôts de données résume des faits selon certains critères d'analyse dans des structures appellées des *cubes*. Formellement, un cube (ou requête analytique) analyse des faits caractérisés par des

dimensions en utilisant une *mesure*. Nous considérons un ensemble de dimensions $d_1, d_2, \ldots, d_n$, tel que chaque dimension d_i peut prendre des valeurs dans un ensemble de valeurs $\{d_i^1, \ldots, d_i^{n_i}\}$; le produit cartésien de toutes les dimensions $d_1 \times \cdots \times d_n$ définit un espace multidimensionnel $\mathcal{M}$. A chaque n-uplet t de cet espace multidimensionnel $\mathcal{M}$ correspond un *sous-ensemble* $\mathcal{F}_t$ des faits analysés, ayant pour chaque dimension $d_{i,1\leq i\leq n}$, la valeur de t dans la dimension d_i.

Une *mesure* est un ensemble de valeurs (c'est un ensemble plutôt qu'une valeur unique, dû à l'hétérogénéité structurelle de l'instance de l'AnS, qui est elle-même un graphe RDF : chaque fait peut avoir zéro, une, ou plusieurs valeurs pour une mesure donnée) caractérisant chaque fait analysé f. Les faits dans $\mathcal{F}_t$ sont réprésentés dans la *cellule du cube* $\mathcal{M}[t]$ par le résultat d'une fonction d'*agrégation* $\oplus$ (telle que *count*, *sum*, *average*, etc.) appliquée à l'union des mesures des faits de $\mathcal{F}_t$: $\mathcal{M}[t] = \oplus(\bigcup_{f\in\mathcal{F}_t} v_f)$.

Une *requête analytique* (AnQ) consiste en deux requêtes enracinées et une fonction d'agrégation. La première requête, nommé *classifieur* dans un entrepôt de données traditionnel, définit les *dimensions* $d_1, d_2, \ldots, d_n$ selon lesquelles seront analysés les faits qui correspondent à la racine de la requête. La deuxième requête définit la *mesure* selon laquelle ces faits vont être résumés. Enfin, la fonction d'aggrégation est utilisée pour résumer les faits analysés.

Réponse à une requête analytique

Considérons maintenant des stratégies pragmatiques pour répondre à des AnQ.

L'approche par matérialisation d'*AnS*. La méthode la plus simple consiste à matérialiser l'instance d'un AnS et de la stocker dans un système de gestion de données RDF (RDF-DM) ; rappelons que l'instance d'un AnS est un graphe RDF graphe défini par des vues GAV. Ainsi, pour répondre à une AnQ, on peut utiliser le RDF-DM pour traiter les requêtes classifieur et mesure, ainsi que l'agrégation finale. Bien qu'effective, cette solution a l'inconvénient de stocker la totalité de l'instance de l'AnS ; de plus, cette instance peut nécessiter d'être mise à jour lorsque le graphe RDF analysé est mis à jour.

L'approche par reformulation d'*AnQ*. Pour éviter la matérialisation et la mise à jour de l'instance d'AnS, nous proposons une solution alternative. L'idée est de réécrire l'AnQ en termes des vues GAV de la définition de l'AnS, de sorte que l'évaluation de la requête ainsi réécrite retourne exactement les mêmes réponses que si la matérialisation avait été utilisée. Cette approche par réécriture des requêtes permet de stocker le graphe RDF original dans un RDF-DM, et d'utiliser ce dernier pour exhiber les réponses aux requêtes réécrites.

Notre technique de reformulation ci-dessous traduit la réécriture standard en termes de vues GAV [11] dans notre cadre analytique pour RDF.

Analyse OLAP RDF

Les technologies OLAP (On-Line Analytical Processing) [1] permettent l'évaluation de requêtes multidimensionnelles sur des entrepôts de données.

Les traitements OLAP permettent de transformer un cube de données en un autre cube. Dans notre contexte, un cube correspond à une AnQ. Nous modélisons les opérations OLAP classiques comme des réécritures d'AnQ.

Une opération *slice* fixe une dimension d'agrégation à une valeur possible de cette dimension. Une opération *dice* restreint plusieurs dimensions à des sous-ensembles de leurs valeurs possibles. Les opérations *drill-in et drill-out* permettent, respectivement, d'ajouter et de supprimer une dimension d'un classifieur.

Dans les scénarios d'applications d'entrepôts de données, les dimensions ont souvent une organisation hiérarchique. Par exemple, une valeur de la dimension *pays* correspond à plusieurs valeurs de *régions*, et chacune de ces valeurs correspond à plusieurs valeurs de *villes*. Notre système ne prévoit pas ce type de hiérarchie au niveau des mécanismes primitifs.

Afin de modéliser des dimensions hiérarchiques, nous introduisons des opérateurs dédiés pour modéliser des relations **niveauSuivant** entre les dimensions mères-filles. Les opérations *roll-up* et *drill-down* correspondent, respectivement, à l'ajout et l'élimination dans le classifieur d'atomes implicants des arcs **niveauSuivant**.

Étude pratique

Notre étude expérimentale a démontré la faisabilité de notre approche de spécification et exploitation d'entrepôts de données RDF. Cette approche s'appuie sur des fonctionnalités fournies par de nombreux systèmes actuels, telles que le stockage de triplets, l'évaluation de requêtes conjonctives, et le raisonnement. Nous avons démontré que la performance de notre système est robuste : la construction de $\mathcal{I}$ se fait en temps linéaire dans la taille du graphe RDF. Enfin, nous avons montré que des requêtes analytiques (AnQ) et des opérations OLAP sur celles-ci peuvent être évaluées de façon efficace. Alors que des optimisations supplémentaires pourraient être appliquées afin d'améliorer les performances, l'évaluation expérimentale décrite ci-dessus confirme l'intérêt et les bonnes performances de notre approche "tout RDF" de construction d'entrepôts de données du Web Sémantique.

Les travaux connexes

Les entrepôts de données relationnels ont été bien étudiés dans le passé [12, 13]. [8, 19] propose des vocabulaires RDF(S) (des classes et propriétés pre-définies) pour décrire en RDF de données relationnelles multidimensionnelles ; [8] définit aussi une traduction de requêtes OLAP vers des requêtes SPARQL. [15] présente une approche semi-automatique pour dériver un entrepôt RDF à partir d'une ontologie. A la diffrence de ces travaux, dans notre approche, une instance du schéma AnS est un graphe RDF préservant la nature hétérogène des données, ainsi que la possibilité d'interroger conjointement le schéma et les données.

Des cubes et des opérations OLAP sur des graphes ont été étudiés auparavant dans [20]. Toutefois, cette approche ne permet pas la gestion de graphes hétérogènes, et donc elle ne permet de gérer ni des attributs multivalués ni les données implicites ; ces deux aspects sont cruciaux dans le contexte de RDF. Qui plus est, les opérations d'agrégation proposées dans cette approche sont centrées sur le comptages d'arcs, alors que notre approche permet d'exprimer des critères d'agrégation bien plus flexibles.

Dans [5], des données de graphes peuvent être agrégées selon des critères spatiaux en groupant des nœuds connectés et situés dans des régions voisines. Ce genre d'agrégation est pertinente lorsque le graphe a une signification bidimensionelle forte, comme par exemple la carte d'une ville ou d'une région, où les arcs sont des rues. En contraste, notre approche est spécifique aux graphes RDF et introduit la notion de schéma analytique et des outils d'agrégation bien plus puissants, notamment en permettant des mesures très flexibles (spécifiées à l'aide de requêtes BGP) et des multiples fonctions d'agrégation.

[14] propose des techniques de transformation de requêtes OLAP en SPARQL. L'evaluation des requêtes est optimisée en utilisant des cubes matérialisés. Cette technique est complémentaire à la notre et peut y être ajoutée dans le but d'optimiser davantage l'évaluation de requêtes AnQ.

La séparation entre les opérations de regroupement et d'agrégation, qui est présente dans notre modéle, est similaire à l'opérateur MD-join [7] introduit dans le contexte des entrepôts de données RDF.

Enfin, le langage SPARQL propose dans sa dernière version (1.1 [18]) des opérateurs de groupement et agrégation dans le style de SQL. Le déploiement de nos techniques sur des plateformes assurant le support efficace de SPARQL 1.1 permettra de tirer profit à la fois de l'efficacité de ces systémes et du pouvoir d'expression de notre modéle pour l'analyse de graphes RDF.

Conclusion

Dans cette thèse, nous proposons des algorithmes et des formalisations pour le traitement de la complexité des données RDF, tout en permettant la portabilité aux systèmes existants de gestion de base de données relationnelle. Nous analysons deux problèmes centraux, répondre aux requêtes sur les données soumises à des contraintes sémantiques, et des analyses complexes sur des données hétérogènes, riches en sémantique.

Saturation vs. Reformulation. La littérature propose deux approches principales pour l'interrogation des données en présence de contraintes sémantiques, ce qui rend explicite les informations qui peuvent être déduites, ou utiliser les contraintes pour remodeler la question. Nous formalisons un cadre commun pour comparer les deux approches, tout en améliorant l'état de l'art pour chacun. Notamment, nous proposons un nouvel algorithme de saturation de données qui est robuste aux changements apportés à l'instance de données et le schéma. De plus, nous décrivons reformulation de la requête pour le fragment RDF considéré, qui s'étend à ceux de la littérature par l'inclusion de noeuds vides. Nous avons montré que la mise en uvre sur le dessus d'un SGBDR rivalise avec l'utilisation de systèmes dédiés. De plus, notre comparaison expérimentale des deux approches permet de quantifier leurs forces et leurs faiblesses, ce qui permet à un administrateur de base de données choisir entre eux.

L'optimisation de l'état-of-the-art actuel de la langue de reformulation de requêtes pour le fragment DB est un travau en cours comme le sujet d'une autre thèse de doctorat.

Analyse de données RDF. Les travaux existants abordent le problème de l'hétérogénéité en normalisant les données dans le processus de Extraction Transformation Chargement, permettant également les valeurs nulles et imbrication. En revanche, nous considérons l'hétérogénéité comme une caractéristique essentielle souhaitée de données RDF, qui devrait se propager à l'entrepôt de données stockant. Dans notre cadre d'analyse les faits, les dimensions et les mesures sont choisis au moment de l'exécution de la requête. Cela permet une grande flexibilité dans le choix de l'analyse, en particulier permettant même l'analyse des concepts de base au travers d'autres concepts de base. En outre, notre cadre facilite l'interrogation du schéma et l'analyse des relations entre les entités. À notre connaissances, notre cadre est le premier à conserver les données tout en RDF tout en fournissant une analyse de données significatives. Dans cette thèse, nous avons démontré à la fois les avantages théoriques d'une telle approche, et son effectivité.

La conception du schéma d'analyse automatique, et l'optimisation de répondre à des requêtes d'analyse sont des travaux en cours dans le cadre de deux thèses de Master.

Perspectives. Nous identifions plusieurs pistes pour les travaux futurs, principalement sur : l'optimisation des techniques décrites et l'automatisation de l'analyse des données :

- l'extension du fragment RDF et du langage de requête ;
- l'analyse des mises à jour RDF Schema ;
- un système offrant le choix dynamique de la technique d'inférence ;
- des méthodes efficaces pour le déploiement de notre cadre d'analyse dans un contexte parallèle ;
- l'intégration des vocabulaires connus dans notre cadre d'analyse ;
- fournir une version publique de notre cadre d'analyse.

References

[1] OLAP Council White Paper. `http://www.olapcouncil.org/research/resrchly.htm`.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] P. Adjiman, F. Goasdoué, and M.-C. Rousset. SomeRDFS in the Semantic Web. *JODS*, 2007.

[4] M. Arenas, C. Gutierrez, and J. Pérez. Foundations of RDF Databases. In *Reasoning Web*, 2009.

[5] D. Bleco and Y. Kotidis. Business intelligence on complex graph data. In *EDBT/ICDT Workshops*, pages 13–20, 2012.

[6] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *Journal of Automated Reasoning (JAR)*, 2007.

[7] D. Chatziantoniou, M. O. Akinde, T. Johnson, and S. Kim. The MD-join: An Operator for Complex OLAP. In *ICDE*, pages 524–533, 2001.

[8] L. Etcheverry and A. A. Vaisman. Enhancing OLAP Analysis with Web Cubes. In *ESWC*, pages 469–483, 2012.

[9] F. Goasdoué, K. Karanasos, J. Leblay, and I. Manolescu. View Selection in Semantic Web Databases. *PVLDB*, 2011.

[10] G. Gottlob, G. Orsi, and A. Pieris. Ontological Queries: Rewriting and Optimization. In *ICDE*, 2011. Keynote.

[11] A. Y. Halevy. Answering Queries Using Views: A Survey. *The VLDB Journal*, 10(4):270–294, Dec. 2001.

[12] M. Jarke, Y. Vassiliou, P. Vassiliadis, and M. Lenzerini. *Fundamentals of Data Warehouses*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1999.

[13] C. S. Jensen, T. B. Pedersen, and C. Thomsen. *Multidimensional Databases and Data Warehousing*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[14] B. Kämpgen and A. Harth. No Size Fits All - Running the Star Schema Benchmark with SPARQL and RDF Aggregate Views. In *ESWC*, pages 290–304, 2013.

[15] V. Nebot and R. B. Llavori. Building data warehouses with semantic web data. *Decision Support Systems*, 52(4):853–868, 2012.

[16] J. Urbani, A. Margara, C. J. H. Jacobs, F. van Harmelen, and H. E. Bal. DynamiTE: Parallel Materialization of Dynamic RDF Data. In *ISWC*, pages 657–672, 2013.

[17] W3C. Resource Description Framework. `http://www.w3.org/RDF`.

[18] W3C. SPARQL 1.1 Query Language. `http://www.w3.org/TR/sparql11-query/`, March 2013.

[19] W3C. The RDF Data Cube Vocabulary. `http://www.w3.org/TR/vocab-data-cube/`, 2014.

[20] P. Zhao, X. Li, D. Xin, and J. Han. Graph cube: on warehousing and OLAP multidimensional networks. In *SIGMOD Conference*, pages 853–864, 2011.