Anonymisation de graphes de connaissances par anatomisation

Maxime Thouvenot, Olivier Curé, Lynda Temal, Sarra Ben Abbès, Philippe Calvez

▶ **To cite this version:**

Maxime Thouvenot, Olivier Curé, Lynda Temal, Sarra Ben Abbès, Philippe Calvez. Anonymisation de graphes de connaissances par anatomisation. EGC (Extraction et Gestion des Connaissances), Jan 2020, Bruxelles, Belgique. hal-02470927

HAL Id: hal-02470927
https://hal.science/hal-02470927

Submitted on 7 Feb 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Anonymisation de graphes de connaissances par anatomisation

Maxime Thouvenot*,** Olivier Curé *
Lynda Temal ** Sarra Ben Abbès ** Philippe Calvez **

*LIGM (UMR 8049), CNRS, F-77454, MLV, France.
prénom.nom@u-pem.fr,
** ENGIE France.
prénom.nom@engie.com

Résumé. Le besoin d'anonymisation de données digitales s'accentue dans un contexte de Big Data. Ce constat concerne tous les types de données et implique donc les graphes de connaissances. Dans cet article, nous présentons une extension de l'approche d'anatomisation aux données représentées avec le modèle de données RDF.

1 Introduction

Avec l'émergence du phénomène Big Data, d'énormes quantités de données sont produites chaque jour et peuvent porter sur des sujets divers tels que des organisations, des entreprises, des personnes et bien d'autres encore. Ces données sont majoritairement stockées dans des centres de données (data centers) dans le but d'être analysées afin d'en extraire de nouvelles informations. La dernière décennie a également vu la naissance du mouvement Open Data qui s'est traduit par la volonté de nombreux acteurs, publiques comme privés, de diffuser leurs données jugées d'intêret général.

Cependant, une part non négligeable de ces données concerne des individus et constitue des informations sensibles qui peuvent menacer la vie privée de ces personnes. Si l'on prend l'exemple du domaine médical, la distribution de certains jeux de données posent des problèmes pour la protection du secret médical.

Beaucoup d'études ont été effectuées en vue de trouver une solution à ce problème d'anonymisation des données. Ce domaine consiste à modifier le contenu ou la structure de ces données afin de rendre très difficile voire impossible la « ré-identification » des personnes ou des entités concernées. De manière générale, nous appellerons "entités d'intêret" (*EI*) les cibles d'une technique d'anonymisation. Chaque EI est identifiée par un ensemble de valeurs que l'on appelle des attributs que l'on peut séparer en quatre catégories différentes :

— **Les identifiants explicites** (*IDE*) : ils permettent d'identifier explicitement une entité. Exemple : le numéro de sécurité sociale.
— **Les quasi-identifiants** (*QID*) : un ensemble d'attributs qui, lorsqu'ils sont utilisés ensemble, rendent possible la ré-identification. Exemple : l'âge, le code postal, etc..

— **Les attributs sensibles** (*AS*) : ils correspondent à des informations sensibles à propos d'une entité que l'on voudrait exploiter lors de nos analyses. Exemple : maladie, salaire, opinion politique, etc..
— **Les attributs non-sensibles** (*ANS*) : ils sont inutiles pour identifier une entité.

Parmi ces catégories, seules les trois premières correspondent à des attributs privés que nous devons considérer avec notre stratégie. Les IDE permettant une ré-identification sans ambiguïté, doivent d'office être supprimés du jeu de données tandis que les QID doivent être anonymisés. Les AS constituent des informations importantes que nous souhaitons pouvoir exploiter pour effectuer des analyses, nous ne les modifierons donc pas.

À ce jour, plusieurs méthodes d'anonymisation ont été développées, le plus souvent appliquées au modèle relationnel en délaissant le modèle de données RDF du Web Sémantique. Quelque soit le modèle de données ciblé, l'objectif est double : pouvoir garantir l'anonymité des entités présentes dans le jeu de données et assurer que ces données soient toujours utilisables (il est encore possible d'en tirer des informations intéressantes). Ce compromis est difficile à atteindre du fait qu'il existe une énorme quantité de connaissances externes, facilement accessibles, qui peuvent être utilisées afin de briser une stratégie d'anonymisation. Dans cet article, nous proposons une nouvelle approche, dénotée anatomisation, dans le contexte des graphes de connaissances.

2 Anatomisation de graphes de connaissances

L'idée de l'anatomisation pour les graphes RDF a été introduite pour la première fois par Radulovic et al. [6]. Leur article ne contient aucune implantation mais offre des pistes sur la façon dont cette technique pourrait être adaptée à cette structure de données. L'objectif de notre travail a consisté à s'appuyer sur ce papier et de l'étendre en prenant en compte la capacité d'inférence propre aux graphes RDF.

Dans cette section, nous commencerons par expliquer le principe de base de l'anatomisation puis nous poursuivrons avec nos contributions avant de finir par la présentation de notre approche en pratique.

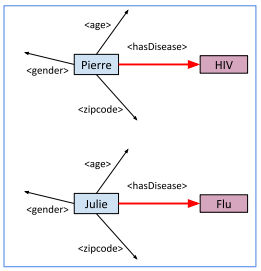
2.1 Présentation générale de l'anatomisation

L'anatomisation consiste à briser les relations entre les QID et leurs AS par l'insertion de noeuds vides intermédiaires (blank nodes). Ces derniers vont être utilisés pour pointer sur des groupes qui rassemblent les différents AS. Pour rappel, on définit un attribut comme l'objet d'un triplet dont le sujet est une EI. On peut noter que contrairement à d'autres méthodes d'anonymisation qui modifient la valeur des données (souvent par généralisation) ou les suppriment (e.g., [3] et [2]), nous ne modifions le graphe qu'au niveau structurel en ajoutant de nouveaux noeuds et arêtes.

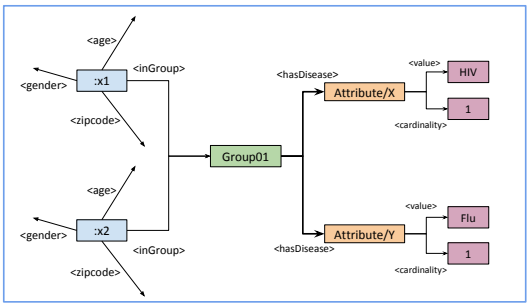
Nous donnons un exemple dans les Figures 1(a) et (b) avec des données médicales contenant des personnes reliées à leur maladie. Notons que deux types de noeuds ont été insérés :

— Les noeuds de type *groupes* utilisés pour casser la relation sensible (ici *Group1*)

FIG. 1: Anatomisation : couper les liens directs entre entités et attributs sensibles



(a) Ressources connectés à un attribut sensible (une maladie)



(b) Application de l'approche.

— Les noeuds de type *attributes* permettant de conserver des informations supplémentaires sur les attributs sensibles comme leur cardinalité (i.e., le nombre de fois qu'ils apparaissent dans le jeu de données).

2.2 Anatomisation guidée par l'ontologie

2.2.1 Principe

En préambule, nous rappelons que nous manipulons des graphes RDF donc des données structurées et associées à des ontologies : des classes ont été définies et sont organisées sous une forme hiérarchique. Ces outils constituent un moyen de comparer les différents concepts dans notre jeu de données, on souhaite donc les exploiter pour créer des groupes qui soient cohérents du point de vue sémantique.

Dans la Figure 2(a), nous proposons une ontologie pour les maladies où celles-ci peuvent être plus ou moins sérieuses (*Critical* et *Regular*) et concerner des parties ciblées du corps comme le coeur ou bien les poumons. Nous donnons aussi des instances de ces maladies tout en bas de l'arborescence. Nous présentons dans la Figure 2(b) un nouvel exemple d'anatomisation basée sur cette ontologie : le premier groupe contient les différentes maladies liées au coeur, le second les maladies des poumons et enfin le dernier regroupe les pathologies bénignes.

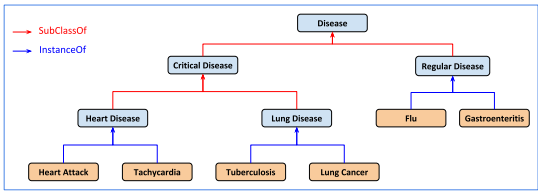
Il est également important de remarquer que les groupes possèdent désormais une sémantique propre car ils appartiennent chacun à une classe correspondant au *least common ancestor* des concepts résultant du raisonnement de *realization* de nos maladies. Nous revenons sur ces notions dans la section suivante.

2.2.2 Realization et Least Common Ancestor

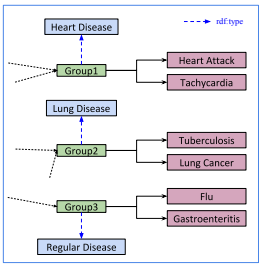
Le service de raisonnement *realization* (REA) [1] d'un invididu (ou d'un fait) correspond au concept le plus spécifique dont il est une instance. À titre d'exemple, dans la Figure 2(a) :

— $REA(Tachycardia) = HeartDisease$
— $REA(Tuberculosis) = LungDisease$

FIG. 2: Anatomisation guidée par une ontologie des maladies.



(a) Exemple de graphe de connaissances pour les maladies.



(b) Création de groupes spécifiques pour les types de maladie.

— $REA(Flu) = RegularDisease$

Par ailleurs, le *least common ancestor* (LCA) de deux concepts *A* et *B* correspond au concept le plus spécifique qui est un ancêtre de *A* et *B*. Nous étendons ce principe aux instances en considèrant que le LCA d'une instance correspondra au LCA de son REA. Si l'on reprend encore une fois le graphe de la figure 2(a) :

— $LCA(HeartAttack, Tachycardia) = HeartDisease$
— $LCA(HeartAttack, Tuberculosis) = CriticalDisease$
— $LCA(Flu, Gastroenteritis) = RegularDisease$
— $LCA(Flu, HeartAttack) = Disease$

2.2.3 Une mesure pour évaluer la similarité

Dans les exemples précédents, nous avions systématiquement plusieurs instances d'un même concept que l'on pouvait réunir dans un groupe. Cependant cette configuration n'est pas toujours présente dans la réalité : il n'est pas rare de trouver des concepts ne comptant qu'une seule instance.

Dans cette situation, notre approche précédente présente des limites pour déterminer le type d'un groupe. Ainsi, il faut définir une mesure générique afin d'évaluer la similarité entre deux instances, quand bien même elles appartiennent à des classes différentes.

Maedche et al. [4] introduisent dans leur papier la notion de *taxonomy similarity* qui calcule la similarité entre deux instances en se basant sur leur concept respectif et leur position dans l'ontologie (autrement dit l'arborescence des classes).

Cette mesure repose sur plusieurs notions que nous présentons maintenant.

Upward Cotopy. Tout d'abord, [4] définit une hiérarchie de concepts *H* comme une relation prenant en arguments deux concepts C_1 et C_2 : $H(C_1, C_2)$ signifie que "C_1 *est un sous-concept de* C_2".

Ils poursuivent ensuite en introduisant la notion d'Upward Cotopy : soient C_i un concept quelconque appartenant à un ensemble de concepts C et H la hiérarchie de concepts associés. On définit alors l'Upward Cotopy comme :

$$UC(C_i,\, H) = \{C_j \in C \mid H(C_i, C_j) \ \vee \ C_j = C_i\} \tag{1}$$

Concrètement, cela correspond à l'ensemble des super-concepts de C_i ainsi que C_i lui-même. Par extension, on considère que l'UC d'une instance correspond à l'UC de son REA.

A partir de l'exemple de la Figure 2(a), nous obtenons :

— $UC(Heart\ Attack,\, H) = \{Disease, Critical\ Disease, Heart\ Disease\}$

— $UC(Tuberculosis,\, H) = \{Disease, Critical\ Disease, Lung\ Disease\}$

— $UC(Regular\ Disease,\, H) = \{Disease, Regular\ Disease\}$

Concept Match. Basé sur l'upward cotopy, le concept match entre deux concepts C_1 et C_2 appartenant à une hiérarchie *H* est défini comme :

$$CM(C_1,\, C_2) = \frac{|(UC(C_1,\, H) \ \cap \ UC(C_2,\, H))|}{|(UC(C_1,\, H) \ \cup \ UC(C_2,\, H))|} \tag{2}$$

On effectue donc un rapport entre la taille de l'intersection des UC et celle de leur union. De la même manière que pour l'upward cotopy, nous étendons ce principe aux instances et nous présentons un exemple avec *Tachycardia* et *Tuberculosis* :

$$CM(Tachycardia,\, Tuberculosis) = \frac{|(UC(Tachycardia,\, H) \ \cap \ UC(Tuberculosis,\, H))|}{|(UC(Tachycardia,\, H) \ \cup \ UC(Tuberculosis,\, H))|}$$

$$= \frac{|\{Heart\ Disease, Critical\ Disease, Disease\} \ \cap \ \{Lung\ Disease, Critical\ Disease, Disease\})|}{|\{Heart\ Disease, Critical\ Disease, Disease\} \ \cup \ \{Lung\ Disease, Critical\ Disease, Disease\})|}$$

$$= \frac{|\{Critical\ Disease, Disease\}|}{|\{Heart\ Disease, Lung\ Disease, Critical\ Disease, Disease\}|} = \frac{2}{4} = 0.5 \tag{3}$$

Taxonomy similarity. Enfin, cette notion entre deux instances I_1 et I_2 est définie comme :

$$TS(I_1, I_2) = \begin{cases} 1 & \text{si } I_1 = I_2 \\ \frac{CM(I_1, I_2)}{2} & \text{sinon} \end{cases} \tag{4}$$

Nous avons présenté le moyen de comparer les attributs sensibles, il faut maintenant les regrouper.

2.2.4 Regroupement hiérarchique

Les algorithmes de regroupement hiérarchique sont des techniques de classification automatiques cherchant à répartir un ensemble d'individus en des classes distinctes.

[4] défend l'idée que ces algorithmes sont préférables car ils produisent des hiérarchies de clusters et contiennent par conséquent plus d'informations que les techniques non-hiérarchiques.

Ils reprennent dans leur papier un algorithme dit "*bottom up*" (littéralement "*du bas vers le haut*") décrit dans Manning et al. [5].

L'algorithme part d'une situation de départ où tous les individus sont seuls dans une classe et où l'on dispose d'une mesure de dissimilarité entre les individus. La méthode va ensuite

```
1  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2  PREFIX owl: <http://www.w3.org/2002/07/owl#>
3
4  SELECT DISTINCT ?concept
5  WHERE {
6      {?concept rdfs:type rdfs:Class.}
7      UNION
8      {?concept rdfs:type owl:Class .}
9  }
```

FIG. 3: Requête SPARQL pour la récupération des concepts de l'ontologie.

procéder de manière itérative en fusionnant à chaque étape les deux classes les plus proches au sens de cette mesure et va ainsi créer une hiérarchie de ces individus.

Dans notre cas, les individus seront bien entendus les attributs sensibles et la mesure de dissimilarité sera la taxonomy similarity décrite plus haut. Dans la section suivante, nous présentons en détail notre méthode mettant en oeuvre ces outils.

2.3 Notre approche en pratique

Notre algorithme prend deux paramètres en entrée : le graphe de connaissances à anonymiser et l'URI de la relation qui relie les entités d'intêret à leur attribut sensible.

Notre méthode consiste à briser ces liens et de les remplacer par des groupes d'attributs afin d'assurer à la fois l'anonymité des individus dans le jeu de données et de conserver au maximum l'utilité des données.

Dans la suite, nous expliquons les différentes étapes de notre algorithme pour parvenir à ce but.

2.3.1 Phase de prétraitement

Récupérer les concepts et calculer l'upward cotopy

Avant de pouvoir entreprendre l'anonymisation, il nous faut déjà récupérer tous les concepts contenus dans l'ontologie du graphe du connaissances et il nous faut un moyen de pouvoir calculer leur upward cotopy. Nous avons mis au point deux requêtes SPARQL à cet effet que nous présentons respectivement dans les Figures 3 et 4.

Pour la récupération des concepts, nous avons recours à une union de façon à combiner des patterns alternatifs dans une même requête, tous les triplets satisfaisant l'un ou l'autre (ou les deux) des patterns seront retenus dans le résultat. On peut traduire cette requête par "nous voulons obtenir toutes les ressources dont le type est *rdfs :Class* ou (non exclusif) *owl :Class*".

La requête pour récupérer les ancêtres est plus compliquée car elle emploie des *property paths* au niveau du prédicat "*subClassOf*" à la ligne 7.

Sans l'opérateur "*", la requête se traduit simplement par "obtenir le super-concept du concept en entrée" mais avec le property path, nous pouvons remonter dans l'arborescence en suivant les liens "*subClassOf*" entre les concepts et récupérer tous les ancêtres.

Nous ne retenons pas les ancêtres *owl :Thing* et *rdfs :Resource* car s'il agit des concepts racines c'est-à-dire que tout individu dans le vocabulaire OWL (respectivement RDFS) est un

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX ex: <http://example.org/examples#>

SELECT ?ancestor
WHERE {
    ex:concept  rdfs:subClassOf*  ?ancestor .
    FILTER(?ancestor  !=  owl:Thing) .
    FILTER(?ancestor  !=  rdfs:Resource)
}
```

FIG. 4: Requête SPARQL pour le calcul de l'upward cotopy.

	Disease	**Critical Disease**	**Regular Disease**	**Heart Disease**	**Lung Disease**
Disease	1				
Critical Disease	0.25	1			
Regular Disease	0.25	0.166	1		
Heart Disease	0.166	0.333	0.125	1	
Lung Disease	0.166	0.333	0.125	0.25	1

TAB. 1: Matrice symétrique des distances pour les concepts de l'ontologie des maladies dans 2(a)

descendant de *owl :Thing* (resp. *rdfs :Resource*). Ces concepts ne contiennent aucune valeur sémantique, c'est pourquoi nous les filtrons dans notre requête.

Matrice de distances

Au cours du processus d'anatomisation, nous allons être amenés à calculer la similarité entre les mêmes concepts à plusieurs reprises : on veut donc optimiser cette partie en évitant les redondances. Pour se faire, nous réalisons le calcul de la similarité au préalable et nous réutilisons cette valeur à chaque fois que l'on en a besoin.

Les taxonomy similarities sont contenues dans une structure de données que nous appelons la matrice de distance, nous garantissant un accès beaucoup plus rapide. Nous donnons un exemple de cette structure dans la Table 1 pour l'ontologie des maladies.

Du point de vue programmation, nous utilisons des tables de hachage imbriquées afin de représenter la matrice : chaque entrée correspond à un concept et pointe sur une nouvelle table de hachage où les entrées sont des concepts qui pointent sur la valeur de similarité.

Un schéma est fourni dans la Figure 5 : $H[Heart\ Disease][Disease]$ correspond à la valeur de similarité entre ces deux concepts et vaut donc 0.166.

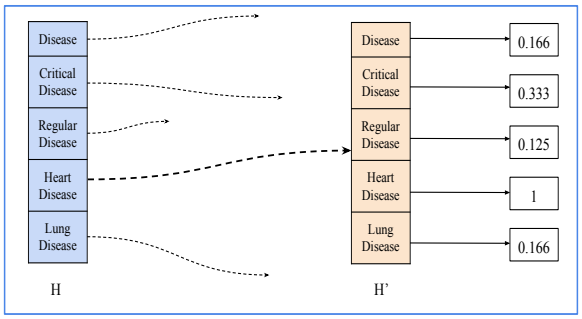


FIG. 5: Tables de hachage imbriquées pour la matrice de distances : exemple avec le concept *Heart Disease*.

Finalement, notre prétraitement se déroule en deux temps :

1. En premier lieu, nous récupérons tous les concepts dans l'ontologie du graphe de connaissances et nous calculons leur upward cotopy grâce aux requêtes des Figures 3 et 4.
2. On calcule ensuite la taxonomy similarity entre toutes les paires de concepts possibles et nous les stockons dans la matrice de distances.

On peut maintenant poursuivre avec la formation des groupes.

2.3.2 Obtenir les groupes initiaux

Durant la phase de clusterisation, nous manipulons deux types d'objets :
— **Sensitive attribute** : utilisé pour représenter les attributs sensibles à anonymiser, il est constitué de deux champs à savoir sa valeur ainsi que sa cardinalité dans le graphe.
— **Cluster** : sert à représenter les groupes lors de l'anatomisation, il est constitué de deux champs : la liste des sensitive attributes qu'il contient et son type qui correspond au LCA des attributs.

Nous avons décidé de reprendre l'algorithme bottom-up pour le hierachical clustering proposé par [5], de ce fait, les clusters initiaux contiendront tous un seul attribut sensible.

À l'aide du prédicat fourni par l'utilisateur et d'une requête SPARQL que nous avons écrite, nous sommes capables de récupérer les valeurs des attributs sensibles, leur concepts issus de la realization et leur cardinalité. Nous présentons la requête en question dans la Figure 6(a) et un exemple de résultats possibles dans la Figure 6(b).

On peut remarquer l'usage de la fonction d'agrégation *COUNT* et de l'opérateur *GROUP BY* permettant, exactement comme en SQL, de calculer des opérations statistiques sur des ensembles d'enregistrements. Dans notre cas, nous récupérons tous les attributs sensibles et leur *REA* puis nous calculons leur nombre d'occurrences dans le jeu de données.

En nous basant sur les résultats obtenus de la requête de la Figure 6(a), nous créons les SensitiveAttributes et les englobons directement dans un objet Cluster dont le type sera le *REA* de l'attribut.

FIG. 6: Construction des clusters initiaux.

```sparql
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX disease: <http://examples.ontotext.com/disease#>

SELECT ?super ?attribute (COUNT(?attribute) as ?count)
WHERE {
    ?s  disease:hasDisease ?attribute .
    ?attribute a ?super .
} GROUP BY ?super ?attribute
```

(a) Requête SPARQL

#	super	attribute	count
1	Heart Disease	Heart Attack	3
2	Heart Disease	Tachycardia	2
3	Lung Disease	Tuberculosis	2
4	Regular Disease	Flu	1

(b) Résultats possibles pour la requête

```sparql
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX ex: <http://example.org/examples#>

SELECT  ?super
WHERE {
    ex:concept1  rdfs:subClassOf*  ?super .
    ex:concept2  rdfs:subClassOf*  ?super .
    FILTER  NOT  EXISTS  {
        ?moreSpecificClass  rdfs:subClassOf  ?super .
        ex:concept1  rdfs:subClassOf*  ?moreSpecificClass .
        ex:concept2  rdfs:subClassOf*  ?moreSpecificClass .
    }
    FILTER(?super  !=  owl:Thing) .
    FILTER(?super  !=  rdfs:Resource)
}
```

FIG. 7: Requête SPARQL utilisée calculer le LCA entre deux concepts.

2.3.3 Fusion des groupes

À présent que les premiers clusters sont formés, nous les stockons dans une liste L_1 et les regroupons de manière itérative :

1. Nous prenons le premier cluster dans L_1 (et l'enlevons de la liste).
2. Nous calculons sa similarité avec tous les autres clusters, y compris ceux étant déjà le résultat d'une fusion.
3. Nous fusionnons les deux clusters les plus proches, ce qui revient à :
   (a) Concaténer leur liste d'attributs sensibles.
   (b) Calculer le LCA de leur concept respectif afin d'obtenir le concept du nouveau cluster.
   (c) Si le second cluster appartient à L_1, il est enlevé de la liste.
4. Le nouveau cluster est ajouté dans une liste L_2.

Nous répétons cette procédure jusqu'à ce que L_1 soit vide.

Une requête SPARQL est de nouveau utilisée afin de calculer le LCA de deux concepts, nous l'exposons dans la Figure 7.

Celle-ci fait à la fois intervenir des property paths et une nouvelle variante du *FILTER* avec la commande *NOT EXISTS*.

```
1  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  PREFIX owl: <http://www.w3.org/2002/07/owl#>
4  PREFIX ex: <http://example.org/examples#>
5
6  DELETE {?s  ex:predicate   ex:attributeValue.}
7  INSERT {
8      ?s   ex:inGroup  ex:groupX  .
9      ex:groupX  rdf:type  ex:concept .
10     ex:groupX  ex:predicate   ex:attributeX .
11     ex:attributeX  ex:value  ex:attributeValue .
12     ex:attributeX  ex:cardinality  ex:attributeCardinality .
13 }
14 WHERE {?s  ex:predicate  ex:attributeValue . }
```

FIG. 8: Requête SPARQL de mise à jour pour appliquer l'anatomisation

Empruntée directement au langage de requête SQL, la commande *EXISTS* renvoie un booléen (vrai ou faux) et est utilisée pour savoir si des triplets existent dans le graphe. L'opérateur *NOT* est quant à lui utilisé pour exprimer la négation.

La requête peut en fin de compte être divisée en deux parties principales :

— La première aux lignes 7 et 8 où l'on recherche l'ensemble des ancêtres communs de *Concept1* et *Concept2*.

— La seconde des lignes 9 à 13 : on filtre tous les ancêtres communs afin de garder uniquement le plus spécifique. Pour tous les ancêtres qui ne sont pas le LCA, la clause *NOT EXISTS* renverra *False*, ils seront donc captés par le FILTER et retirés des résultats de la requête.

Par ailleurs, on filtre de nouveau les concepts *owl :Thing* et *rdfs :Resource* car ils sont les ancêtres de toutes les classes et sont trop généraux.

2.3.4 Création des requêtes de mise à jour

Une fois que les clusters sont formés, l'algorithme peut produire les opérations de mise à jour à exécuter pour appliquer l'anatomisation. Nous itérons sur chacun des clusters et sur chacun des attributs qu'ils contiennent afin de créer les requêtes SPARQL adéquates.

Nous donnons un exemple de requête dans la Figure 8 ainsi qu'un schéma représentant son exécution dans la Figure 9. Les valeurs de *GroupX* et *AttributeX* sont simplement des indices dont la valeur évolue au fur et à mesure que l'on itère sur les clusters et les attributs.

La clause $DELETE$ nous sert à supprimer le lien direct entre un individu et son attribut sensible tandis que la clause $INSERT$ est utilisée dans le but d'ajouter les noeuds intermédiaires et les relations entre ces derniers.

2.4 L'implantation

Nous avons développé notre programme en Java en utilisant la librairie libre *Apache Jena* conçue pour les technologies du Web sémantique. Elle offre un large panel de modules et de classes utilitaires afin de rendre plus facile la manipulation de graphes RDF.

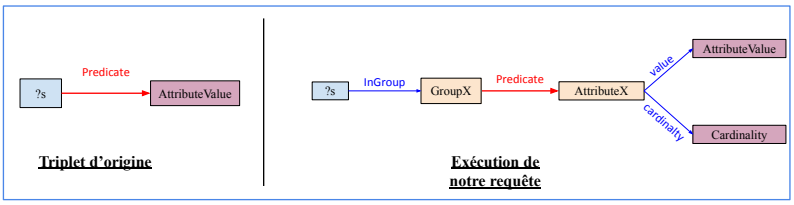
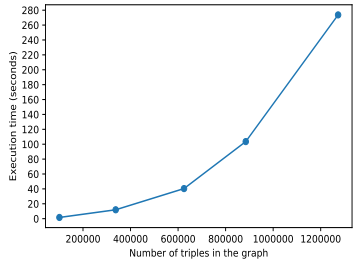
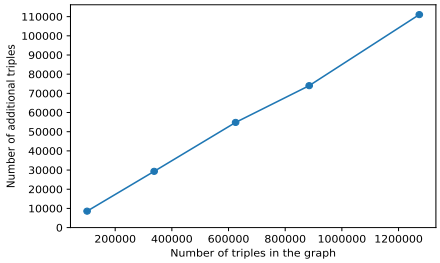


FIG. 9: Exécution d'une requête de mise à jour : exemple sur un triplet

FIG. 10: Évaluation de notre approche



(a) Temps d'exécution



(b) Nombre de triplets additionnels

Les graphes y sont représentés par l'intermédiaire d'une interface appelée "*Model*" et peuvent être construits à partir de fichiers, d'autres modèles ou bien manuellement. En outre, ce système d'interface est intéressant car il offre un plus haut niveau d'abstraction pour différents types de structures de données tels que des graphes stockés en mémoire, ceux stockés sur disque ou encore les modèles d'inférence rendant possible le raisonnement sur les données.

Nous avons également eu recours à *ARQ*, le moteur de requête de Jena pour créer et exécuter les différentes requêtes présentées précédemment.

3 Evaluation

Dans cette section, nous présentons une évaluation préliminaire de notre approche d'anatomisation sur une extension du jeu de données LUBM [1]. A partir de 5 graphes RDF de tailles différentes, allant de 100.000 à 1 million de triplets, nous avons testé (sur un laptop équipé de 16Go de RAM) les temps d'exécution de notre approche et l'évolution du nombre total de triplets des jeux de données anonymisés.

La Figure 10(a) montre que l'exécution de notre algorithme d'anatomisation croît de manière exponentielle avec le nombre distinct d'attributs sensibles. Ce résultat peut s'expliquer par notre approche bottom-up en complexité O(n^2) : à chaque étape, nous prenons un groupe et nous le comparons avec tous les autres. Par contre, la Figure 10(b) met en évidence une crois-

1. http ://swat.cse.lehigh.edu/projects/lubm/

sance linéaire du nombre de triplets du graphe anonymisé, en effet, nous ajoutons 4 triplets à chaque attribut sensible (groupe, type de groupe, valuer et cardinalité).

4 Conclusion

Dans cet article, nous avons présenté une approche d'anonymisation basée sur la solution d'anatomisation pour des graphes de connaissances. A partir de relations sensibles, notre approche déclarative est capable de casser les liens entre les ressources et ses attributs sensibles tout en préservant la connectivité des noeuds du graphe et sans altérer de valeurs contenues dans le graphe, en dehors des QID. Nos futurs travaux concernerons l'impact du raisonnement à partir d'ontologies dans un contexte d'anonymisation, par exemple en considérant les propriétés inverses et transitives.

Références

[1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook : Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[2] Remy Delanaux, Angela Bonifati, Marie-Christine Rousset, and Romuald Thion. Query-based linked data anonymization. In *ISWC, 2018*, pages 530–546.

[3] Benjamin Heitmann, Felix Hermsen, and Stefan Decker. k-RDF-neighbourhood anonymity : Combining structural and attribute-based anonymisation for linked data. In *PrivOn co-located with ISWC*, 2017.

[4] Alexander Maedche and Valentin Zacharias. Clustering ontology-based metadata in the semantic web. In *PKDD 2002*, pages 348–360, 2002.

[5] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.

[6] Filip Radulovic, Raúl García-Castro, and Asunción Gómez-Pérez. Towards the anonymisation of RDF data. In *SEKE*, pages 646–651, 2015.

Summary

The need to anonymize digital data is increasing in a Big Data context. This fact concerns all data types including knowledge graphs. In this paper, we present an extension of the anatomization method for data represented with the RDF data model.