HAL
open science

Détection de clefs pour l'interconnexion et le nettoyage de jeux de données

Jérôme David, François Scharffe

To cite this version:

Jérôme David, François Scharffe. Détection de clefs pour l'interconnexion et le nettoyage de jeux de données. Ingénierie des Connaissances, Jun 2012, Paris, France. pp.Page 401. hal-00741734

HAL Id: hal-00741734

https://hal.science/hal-00741734v1

Submitted on 16 Oct 2012

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.

Détection de clefs pour l'interconnexion et le nettoyage de jeux de données

Jérôme David[1], François Scharffe[2]

[1] INRIA, 655 AVENUE DE L'EUROPE,
MONTBONNOT, FRANCE
JEROME.DAVID@INRIA.FR

[2] LIRMM, UNIVERSITYÉ DE MONTPELLIER 2,
161 RUE ADA, MONTPELLIER, FRANCE
FRANCOIS.SCHARFFE@LIRMM.FR

Résumé : Cet article propose une méthode d'analyse de jeux de données du Web publiés en RDF basée sur les dépendances de clefs. Ce type particulier de dépendances fonctionnelles, largement étudié dans la théorie des bases de données, permet d'évaluer si un ensemble de propriétés constitue une clef pour l'ensemble de données considéré. Si c'est le cas, il n'y aura alors pas deux instances possédant les mêmes valeurs pour ces propriétés. Après avoir donné les définitions nécessaires, nous proposons un algorithme de détection des clefs minimales sur un jeu de données RDF. Nous utilisons ensuite cet algorithme pour détecter les clefs de plusieurs jeux de données publiées sur le Web et appliquons notre approche pour deux applications : (1) réduire le nombre de propriétés à comparer dans le but de détecter des ressources identiques entre deux jeux de données, et (2) détecter des erreurs à l'intérieur d'un jeu de données.

Mots-clés : web sémantique, web de données, interconnexion, ontologies, clefs, dépendances fonctionnelles, nettoyage de données, RDF

1 Introduction

La notion de clef est fondamentale pour les bases de données relationnelles. Les clefs permettent d'identifier de manière unique chaque tuple d'une relation. La propriété d'unicité des clefs est également exploitée afin d'optimiser les accès aux données par la construction d'index. Généralement, les clefs sont identifiées et choisies par le concepteur du schéma relationnel en tenant compte du passage aux différentes formes normales. Cependant, il existe des algorithmes permettant de détecter des dépendances

fonctionnelles (et donc les clefs) à l'intérieur d'un jeu de données particulier (Mannila & Raiha, 1994), (Huhtala *et al.*, 1999).

Dans le cadre du Web sémantique, ce n'est que depuis la standardisation de la version deux du langage d'ontologies du Web OWL [1] que la modélisation de clefs est possible. Une clef dans OWL2 pour une classe donnée est un ensemble de propriétés permettant d'identifier uniquement une instance de cette classe. Selon la sémantique d'OWL2 [2], deux instances ayant les mêmes valeurs pour les propriétés d'une clef seront considérées comme identiques. L'utilisation de clefs demande donc de connaître à l'avance les données qui seront représentées à l'aide de l'ontologie.

La publication décentralisée de jeux de données sur le Web rend difficile pour le concepteur d'une ontologie de pouvoir à l'avance déterminer si un ensemble de propriétés constituera une clef pour une classe donnée. Les jeux de données réutilisent des ontologies diverses, et le concepteur d'une ontologie a intérêt à ne pas imposer trop de contraintes à l'adoption de l'ontologie afin de favoriser sa réutilisation (voir à ce sujet (Vatant *et al.*, 2011)).

Au niveau d'un jeu de données particulier, il sera possible en analysant les propriétés qu'il contient de déterminer si une ou plusieurs clefs existent pour une classe. Étant donné le caractère imparfait des données du Web, il faudra tolérer que dans certains cas des instances aient les mêmes valeurs pour les propriétés de la clef. Nous parlerons dans ce cas de *pseudo-clefs*. On pourra indiquer les clefs comme méta-données associées au jeu de données, par exemple en étendant le vocabulaire VoID (Alexander *et al.*, 2009).

Nous proposons ici un algorithme de recherche de clefs minimales adapté aux données RDF et démontrons l'utilité de la fouille de clefs à travers deux applications importantes pour le web des données : la découverte de liens entre jeux de données et la détection d'erreurs dans un jeu de données.

Nous commençons par définir formellement la notion de dépendance de clef pour un jeu de données RDF puis nous proposons un algorithme permettant de détecter ces dépendances (section 2). La détection des clefs peut permettre plusieurs applications intéressantes (section 3). Nous en donnerons une permettant de réduire le nombre de comparaisons nécessaires pour déterminer l'équivalence d'instances en provenance de deux jeux de données. Nous verrons ensuite comment l'utilisation de pseudo-clefs peut permettre de détecter la présence d'erreurs sous forme de redondances dans

1. `http://www.w3.org/TR/owl-overview/`
2. Voir la définition : `http://www.w3.org/TR/owl2-semantics/#Keys`

un jeu de données.

2 Dépendances de clefs

La représentation des données sur le web sémantique est réalisée grâce au langage RDF.

2.1 Définitions

Définition 1
Un graphe RDF G est un ensemble de triplets RDF $t = \langle s, p, o \rangle$ où :
- *s, le sujet ou l'instance est une URI ou un noeud anonyme*
- *p, le prédicat ou la propriété est une URI*
- *o, l'objet, est une URI ou une valeur littérale ou un noeud anonyme*

L'ensemble des sujets et l'ensemble des objets d'un graphe G sont notés respectivement S_G et O_G.

Un prédicat p peut être vu comme une relation entre l'ensemble des sujets et l'ensemble des objets ou comme une fonction (partielle) de l'ensemble des sujets vers l'ensemble des parties de l'ensemble des objets. Dans ce travail, nous considérons la seconde modélisation : $p : S_G \rightarrow 2^{O_G}$. L'ensemble des images d'un sujet pour un prédicat donné est donc défini de la manière suivante :

Définition 2
L'ensemble des objets images de s par p dans le graphe G

$$p(s) = \{o | \langle s, p, o \rangle \in G\}$$

Lorsque la fonction p est injective (inversement fonctionnelle) dans G, alors le prédicat p est une clef pour G. Dans ce cas, les valeurs prises par p permettent de discriminer les sujets de G qui font partie du domaine de ce prédicat.

Il est intéressant de considérer également des clefs composées de plusieurs prédicats tout comme cela est fait dans le modèle relationnel. En effet, plus on prend en compte de prédicats simultanément, plus on augmente les chances de discriminer les sujets.

La notion de clef dans un graphe RDF donné, peut être généralisée à un ensemble de prédicats de la manière suivante :

Définition 3
Un ensemble de prédicats P est une clef du graphe G, ssi pour tout $s_1, s_2 \in S_G$, nous avons : si $p(s_1) = p(s_2)$ pour tout $p \in P$, alors $s_1 = s_2$.

La clef P est minimale si il n'existe pas d'ensemble $P' \subset P$ qui est également une clef pour G.

La principale différence avec les clefs du modèle relationnel est qu'un prédicat peut être mutlivalué, i.e. $p(s)$ est un ensemble de valeurs.

Afin de vérifier concrètement si un ensemble de prédicats est une clef d'un graphe donné, l'approche que nous utilisons consiste à construire puis analyser la partition des sujets de G induite par un ensemble de prédicats donné. Si cette partition ne contient que des singletons, alors cet ensemble de prédicats est une clef dans G.

Nous définissons tout d'abord cette notion de partition à un prédicat avant de la généraliser à un ensemble de prédicats.

Définition 4
La partition des sujets engendrée par un prédicat p est définie par :

$$\pi(p) = \{p^{-1}(p(s)) | s \in dom(p)\}$$

où $dom(p)$ est le domaine de p :

$$dom(p) = \{s | \exists o \ \langle s, p, o \rangle \in G\}$$

et $p^{-1}(O)$ est l'inverse de la fonction p :

$$p^{-1}(O) = \{s \in G | p(s) = O\}$$

Définition 5
La partition des sujets engendrée par un ensemble de prédicats $P = \{p_1, ..., p_n\}$ est définie par :

$$\pi(P) = \{x_1 \cap ... \cap x_n\}_{(x_1,...,x_n) \in \pi(p_1) \times ... \times \pi(p_n)}$$

A partir de cette partition, nous pouvons dire que P est une clef pour le graphe G si $\forall x \in \pi(P)$, on a $|x| <= 1$.

La complexité du calcul d'une telle partition est polynomiale en fonction du nombre de sujets, mais exponentielle en fonction du nombre de prédicats dans P.

Un prédicat (ou plus généralement un ensemble de prédicats) n'est pas nécessairement associé à l'ensemble des sujets du graphe. Il peut donc exister des clefs qui ne concernent qu'un sous ensemble négligeable des sujets et qui sont donc a priori peu intéressantes. Nous utilisons donc un critère de support minimum d'un ensemble de prédicats.

Définition 6
Un ensemble de prédicats P vérifie le critère de support minimum T_s si :

$$support(P) = |\cap_{p \in P} Dom(p)| > T_s$$

Nous sommes également intéressés par détecter des ensembles de prédicats qui sont presque des clefs. C'est pourquoi nous introduisons la notion de pseudo clef.

Définition 7
Un ensemble de prédicats P est une pseudo-clef du graphe G au seuil de discriminabilité T_r, ssi

$$\frac{|\{x | x \in \pi(P) \text{ et } |x| = 1\}|}{support(P)} > T_r$$

2.2 Algorithme

L'algorithme de découverte de clefs minimales dans un graphe RDF utilise la même stratégie de recherche par niveau (en largeur) que l'algorithme de découverte de dépendances fonctionnelles TANE (Huhtala *et al.*, 1999). Il teste tous les ensembles de prédicats de taille 1, puis de taille 2, etc. L'intérêt de suivre une telle approche est double. Cela permet non seulement d'élaguer l'espace de recherche mais aussi de réduire le coût de calcul des partitions.

L'élagage de l'espace de recherche fonctionne de la manière suivante : lorsqu'un ensemble de prédicats est une clef, pseudo-clef, qu'il existe une dépendance fonctionelle dans cet ensemble, ou que son support est trop faible alors tous ses sur-ensembles seront ignorés. Cela permet également d'assurer que seules des clefs minimales seront générées.

La réutilisation des partitions générées au niveaux précédents permet de réduire significativement le coût de calcul des partitions aux niveaux suivants. En effet, l'opération de calcul d'une partition $\pi(P)$ est exponentielle en fonction de la taille de P. Par contre, cette opération peut être réalisée de manière incrémentale à partir des partitions $\pi(P')$ et $\pi(P'')$ (tel que $P' \cup P'' = P$) générées au niveau précédent :

$$\pi(P', P'') = \{x' \cap x''\}_{\forall (x', x')' \in \pi(P') \times \pi(P'')}$$

L'algorithme 1 fonctionne de la manière suivante. Pour chaque ensemble de prédicats c testé à l'itération précédente ($candidates$), il teste chaque

nouvel ensemble formé de l'union de c et d'un des prédicats P_G (n'appartenant pas à c). Si ce nouvel ensemble est une clef ou pseudo clef alors il est ajouté à l'ensemble des clefs. Si cet ensemble n'est pas une clef mais que son support est supérieur à la valeur seuil et qu'il n'existe pas dépendance fonctionelle sur cet ensemble, alors il est gardé pour les itérations suivantes $nextCandidates$. Les itérations se poursuivent ainsi de suite jusqu'à ce qu'il n'y ait plus d'ensemble de prédicats dans $candidates$.

Afin d'éviter toute génération de clefs non-minimales ou tout parcours d'ensemble de prédicats ayant un support trop faible, l'algorithme utilise un ensemble de "listes noires" ($skipLists$). Le principe est le suivant : lorsque l'union d'un candidat c et d'un prédicat p a généré une clef, une pseudo clef, si il existe une dépendance sur cet ensemble ou si cet ensembe a un support inférieur au seuil fixé, alors le candidat c est ajouté à la liste d'exclusion du prédicat p. A chaque génération d'un nouvel ensemble à partir d'un candidat c' et du prédicat p, on vérifie si c' ne contient pas un sous-ensemble exclu pour p.

L'algorithme 1 tel qu'il est décrit ne détecte pas les clefs d'une classe donnée, mais analyse les clefs valides sur l'ensemble du graphe. Pour obtenir les clefs propres à une classe, il suffit de l'exécuter sur le sous-graphe ne contenant que les instances de la classe considérée : $G_c = \{\langle s, p, o\rangle \in G| < s, rdf : type, c > \in G\}$.

3 Applications

3.1 Interconnexion de jeux de données

Dans le cadre du projet ANR Datalift [3](Scharffe, 2012 à paraître) nous construisons une plateforme permettant de publier un jeu de données structurées disponible dans un format "brut" (CVS, XML, base de données relationnelle) sur le Web de données. Nous avons identifié les quatre étapes suivantes, nécessaires pour arriver à un jeu de données répondant aux principes de base des données liées : (1) La sélection d'un vocabulaire pour décrire les données, (2) la conversion des données brutes en RDF en respectant le vocabulaire sélectionné, (3) la publication des données sur un serveur fournissant les services de négociation de contenu, de dé-référencement des ressources et d'interrogation à l'aide du langage SPARQL, et finalement (4) l'interconnexion des données publiées avec d'autres jeux de données déjà disponibles sur le web. Dans ce contexte, nous travaillons en par-

3. Projet Datalift ANR-10-CORD-009. `http://datalift.org`

Algorithme 1 Découverte de clefs dans un graphe G.

```
predicates ← P_G
candidates ← {∅}
parts ← ∅
skipLists ← {∅}
while |candidates| > 0 do
    nextCandidates ← ∅
    for all c ∈ candidates do
        firstIdx ← max(predicates.indexOf(c[c.length − 1]), 0)
        for i = firstIdx → predicates.length do
            if ∄l s.t. l ∈ skipLists[predicates[i]] ∧ l ⊆ c then
                P ← c ∪ predicates[i]
                parts[P] ← π(parts[c], parts[predicates[i]])
                if key(part[P]) ∨ pseudo_key(part[P], T_r) then
                    keys ← ∪{P}
                    skipLists[predicates[i]] ← ∪{c}
                else if support(P) < T_s or ∃a s.t. P − {a} → a then
                    skipLists[predicates[i]] ← ∪{c}
                else if i < predicates.length − 1 then
                    nextCandidates ← ∪{P}
                end if
            end if
        end for
        candidates ← nextCandidates
    end for
end while
```

ticulier à développer une méthode permettant de lier les données de façon automatisée.

Le problème de l'interconnexion de données est le suivant : étant donné deux jeux de données, comment déterminer quelles sont les instances équivalentes, c'est à dire quelles sont les instances qui représentent le même objet du monde réel ? L'ensemble des problématiques liées à cette question est présenté en détails dans (Ferrara *et al.*, 2011).

Notre approche est basée sur le cadre général défini dans (Scharffe & Euzenat, 2010). La première étape consiste à aligner les ontologies utilisées par les deux jeux de données. Cela permet, dans une deuxième étape, de sélectionner un ensemble de propriétés permettant de minimiser le nombre de comparaisons nécessaires pour identifier les instances en commun. Enfin, dans une troisième étape, il faut déterminer les mesures de similarité à utiliser pour comparer chaque paire de propriétés. Ces trois étapes serviront à paramétrer un outil qui réalisera la comparaison des instances. Plusieurs outils mettent en place ce processus, par exemple (Bizer *et al.*, 2009; Nikolov *et al.*, 2008; Scharffe *et al.*, 2009). Si la première étape peut être en partie automatisée, les deux dernière nécessitent un travail manuel important.

Grâce à la découverte de clefs, nous proposons d'automatiser la deuxième étape consistant à sélectionner un ensemble suffisant mais minimum de propriétés à comparer. Le processus suivi consiste tout d'abord à calculer les clefs sur les deux jeux de données et puis à sélectionner celles qui forment une clef qui est commune aux deux jeux de données. Nous considérons équivalentes deux clefs constituées exactement des mêmes propriétés mais également celles dont les propriétés sont différentes mais reliées par un alignement spécifiant leur équivalence. Prenons un exemple sur les jeux de données Drugbank et Sider.

Drugbank [4] et Sider [5] sont deux bases de données proposant des informations détaillées sur les médicaments. Nous voulons interconnecter les médicaments, décrits par les classes `drugbank:drugs` et `sider:drugs`. Les jeux de données contiennent respectivement 4772 et 924 médicaments dans leurs versions RDF [6] décrits par 108 et 10 propriétés respectivement.

Dans cet exemple, notre démarche consiste à détecter le plus petit ensemble de propriétés nécessaires pour identifier les médicaments en com-

4. `http://www.drugbank.ca/`

5. `http://sideeffects.embl.de/`

6. Voir `http://www4.wiwiss.fu-berlin.de/drugbank` et `http://www4.wiwiss.fu-berlin.de/sider/`

Propriétés constituant la pseudo-clef	Support
`foaf:page`	1
`db:genericName`	1
`db:primaryAccessionNo`	1
`db:updateDate`	1
`rdfs:label`	1
`db:limsDrugId`	1

`db:smilesStringCanonical`
`db:drugType`
`db:pubchemCompoundId`
`db:creationDate`
	0.928

`db:pubchemCompoundId`
`db:drugType`
`db:creationDate`
`db:smilesStringIsomeric`
	0.928

`db:pubchemSubstanceId`	0.922

TABLE 1 – Détection de clef pour la classe `Drugbank :drugs`.

Propriétés constituant la pseudo-clef	Support
`si:siderDrugId`	1
`si:drugName`	1
`foaf:page`	1
`rdfs:label`	1
`si:stitchId`	1
`si:sideEffect`	0.965
`rdfs:seeAlso`	0.848

TABLE 2 – Détection de clef pour la classe `Sider :drugs`.

mun dans les deux jeux de données. L'exécution de l'algorithme1 nous retourne les clefs suivantes, tables 1 et 2, ordonnées par support décroissant.

L'analyse des clef trouvées par l'algorithme nous montre deux choses. D'une part la propriété `rdfs:label` est clef pour les deux jeux de données. Cette propriété est donc un candidat potentiel pour réaliser l'interconnexion. D'autre part, les propriétés `drugbank:genericName` et `sider:drugName` sont également candidates du fait qu'elles sont des clefs dans les deux jeux de données et qu'elle sont équivalentes par alignement. Par ailleurs nous pouvons remarquer que la propriété `foaf:page` est aussi clef dans les deux jeux de données. En effet, chaque médicament possède une page web unique présentant des informations, et ce dans les

Propriétés constituant la pseudo-clef	Support
`http://dbpedia.org/ontology/deathDate` `http://dbpedia.org/ontology/birthDate`	0.203
`http://dbpedia.org/ontology/deathDate` `http://dbpedia.org/ontology/deathPlace`	0.216
`http://xmlns.com/foaf/0.1/name` `http://dbpedia.org/ontology/birthPlace`	0.442
`http://xmlns.com/foaf/0.1/surname` `http://purl.org/dc/elements/1.1/description`	0.459
`http://dbpedia.org/ontology/deathPlace` `http://dbpedia.org/ontology/birthDate`	0.480

TABLE 3 – Détection de clef pour la classe `DBPedia :Person`.

deux jeux de données. Par contre, cette propriété ne pourra pas être utilisée pour l'interconnexion car les URL de ces pages ne sont pas comparables, il sera donc impossible d'établir une similarité entre médicaments en comparant les valeurs de cette propriété.

Nous avons donc pu automatiser cette phase de sélection de la clef pour l'interconnexion de jeux de données. L'exemple présenté ici est simple car les clefs trouvées ne sont constituées que d'une seule propriété. Dans la pratique, il se peut que de multiples clefs de taille variable soient retournées par l'algorithme. Il faudra alors élaborer une stratégie de sélection de la clef.

3.2 Détection d'erreurs

L'expérimentation de l'algorithme présenté dans cet article nous a amené à considérer une application originale : la détection d'erreurs ou de défauts dans un jeu de données. Si l'on dégrade la notion de clefs pour détecter des pseudos-clefs, on voit alors apparaître des clefs qui sont valides pour la plupart des instances mais ne sont pas des clefs pour un très petit nombre de paires d'instances. Leur observation indique en fait la présence de doublons ou d'erreurs dans le jeu de données. Nous avons appliqué cette méthode sur les 242 classes du jeu de données DBPedia en utilisant l'algorithme présenté dans cet article. Voici un exemple des pseudo-clefs obtenues pour la classe dbpedia :Person calculées avec un support minimal de 0.2 et un *seuil de discriminabilité* supérieur à 0.999. La table 3 montre les clefs identifiées et leur support.

La première ligne cette table nous indique qu'il existe des personnes nées le même jour et décédées le même jour, ce qui n'est pas impossible

mais statistiquement rare. Une vérification peut être effectuée en transformant ces pseudos-clefs en requêtes SPARQL et en les exécutant sur le jeu de données. La requête doit vérifier quelles sont les ressources ayant les mêmes valeurs pour les propriétés constituant la clef pour la classe observée. On obtient donc la requête suivante pour la clef (`dbpedia:birthPlace`, `dbpedia:deathPlace`) :

```
Select distinct ?x ?y
Where {
  ?x dbpedia-owl:deathDate ?dp;
      dbpedia-owl:birthDate ?bd;
      rdf:type dbpedia-owl:Person.
  ?y dbpedia-owl:deathDate ?dp;
      dbpedia-owl:birthDate ?bd;
      rdf:type dbpedia-owl:Person.
   Filter (?x!=?y) }
```

L'analyse manuelle de la réponse [7] nous montre que les 125 paires d'instances retournées par cette requête correspondent en fait à divers types d'erreurs dans le jeu de données. L'erreur la plus fréquente advient lorsque deux ressources existent pour décrire un même objet, par exemple `dbpedia:Louis_IX_of_France__Saint_Louis___1` et `dbpedia:Louis_IX_of_France`
Un deuxième type d'erreur semble être du à un problème de conversion des «Infoboxes» de Wikipedia lors de la génération de DBPedia. Ces erreurs mènent à des défauts de classification des ressources. Par exemple `dbpedia:Timeline_of_the_presidency_of_John_F._Kennedy` est classifié comme une personne alors qu'il s'agit d'une chronologie. Finalement, un troisième type d'erreur vient des pages Wikipedia elles-mêmes ou bien de documents à partir desquels ces pages on été renseignées. [8]

L'analyse systématique des résultats de cette expérimentation sur l'ensemble des classes de DBPedia sort du cadre de cet article. Ces résultats sont disponibles en ligne en RDF [9]. Il serait aussi intéressant d'utiliser cette méthode pour corriger d'autres jeux de données.

---

7. Requête posée sur `http://dbpedia.org/sparql`

8. Voir par exemple les pages `http://dbpedia.org/resource/Merton_B._Myers` et `http://dbpedia.org/resource/William_J._Pattison` et la notice explicative à la fin des articles correspondant.

9. `http://data.lirmm.fr/keys/dbpedia_mb_keys_01_99_s10.n3`

	# de triples	# classes	# propriétés	# d'instances	# de clefs	temps de calcul
DBPedia	13,8 M	250	1 100	1 668 503	9 367	179'48"
DrugBank	0,77 M	8	119	19 693	3 040	6'58"
DailyMed	0,16M	6	28	10 015	1 171	1'46"
Sider	0,19M	4	11	2 674	14	5"

TABLE 4 – Taille des jeux de données, nombre de clefs et pseudo-clefs trouvées et temps de calcul.

3.3 Efficacité et passage à l'échelle de l'algorithme

L'algorithme de recherche de clefs et pseudo-clefs a été implémenté en Java. Afin de pouvoir passer à l'échelle, de nombreux résultats intermédiaires de l'algorithme (les partitions) sont stockés sur disque. De plus, les jeux de données ont été préalablement stockés en local sur disque et indexés en utilisant la bliblothèque Jena TDB [10].

Nous avons exécuté l'algorithme sur un ordinateur quadri-coeurs Intel(R) Xeon(R) E5430 @ 2.66GHz avec 8GO de mémoire vive et deux disques dur 15000 tours en RAID 0. La table 4 donne les temps (somme des temps CPU + accès disque) pour le calcul de toutes les clefs et pseudo clefs pour toute les classes prises successivement. L'algorithme a été paramétré avec un seuil de support $T_s = 0,1$ et avec un seuil de discriminabilité $T_r = 0,99$.

La table 4 montre que les temps de calcul dépendent fortement du nombre de classes et de propriétés et qu'ils sont moins sensibles au nombre d'instance du jeux de données. Cela est en adéquation avec la complexité exponentielle en fonction du nombre de propriétés de l'algorithme. Même si ces temps de calculs sont prohibitifs pour une utilisation interactive, ils montrent qu'il est tout à fait possible de généraliser l'approche à de très gros jeux de données.

4 Etat de l'art

Les dépendances fonctionnelles, dont les dépendances de clefs sont un cas particulier, ont largement été étudiées dans le cadre des bases de données relationnelles. Plusieurs algorithmes efficaces ont été proposés pour la détection de dépendances fonctionnelles. Nous pourrons citer l'algorithme TANE (Huhtala *et al.*, 1999) efficace lorsque le nombre de propriétés pouvant potentiellement faire partie d'une dépendance est élevé ou plus ré-

10. `http://incubator.apache.org/jena/documentation/tdb/`

cemment FD_Mine (Yao & Hamilton, 2008) améliorant significativement les performances de TANE à l'aide d'heuristiques d'élagage de l'arbre de recherche.

Une méthode de détection d'erreurs dans les données a été proposée dans (Yu *et al.*, 2011). Cette méthode repose sur la détection de "dépendance sémantique". Ce type de dépendance est présenté comme une extension des dépendances fonctionelles prenant en compte la composition de prédicats et le relâchement de l'égalité entre valeurs. Cependant, les pseudo-clefs ne sont pas considérées car elles sont élaguées par l'algorithme.

Une méthode pour détecter des clefs dans les jeux de données RDF est proposée dans (Symeonidou *et al.*, 2011). Le but de ce travail est le même que celui que nous présentons section 3.1 : l'interconnexion de données. Cette méthode repose sur la technique Gordian (Sismanis *et al.*, 2006), permettant de détecter des clefs par un parcours en pronfondeur d'abord. L'article ne mentionne pas l'efficacité de la méthode par rapport au nombre de triplets et de propriétés contenus dans le jeu de données.

Avec la même problématique de départ, l'interconnexion de données, une méthode pour calculer la couverture d'une propriété et son pouvoir discriminant est proposée dans (Song & Heflin, 2011). La couverture est le ratio entre le nombre total d'instances pour une classe donnée et le nombre d'instances de cette classe ayant cette propriété. Le pouvoir discriminant est le ratio entre le nombre total d'instances ayant une propriété et le nombre de valeurs distinctes pour cette propriété. Une propriété ayant une couverture et un pouvoir discriminant égal à 1 est donc une clef.

5 Conclusion

Ce travail propose un algorithme de découverte de clefs et pseudo-clefs adapté aux jeux de données RDF. L'algorithme est relativement efficace même sur des grand jeux de données grâce à l'utilisation d'élagages basés sur un support minimal des clefs et la non génération de redondances.

L'utilité de disposer d'un tels algorithme est démontrée au travers de deux applications : l'interconnexion de jeux de données et la détection de redondances.

Nos objectifs futurs sont de continuer sur la voie de l'interconnexion en proposant une solution verticale et complète permettant de générer automatiquement des configurations pour les outils de comparaison d'instances.

Références

ALEXANDER K., CYGANIAK R., HAUSENBLAS M. & ZHAO J. (2009). Describing linked datasets - on the design and usage of voiD, the 'vocabulary of interlinked datasets'. In *Linked Data on the Web Workshop (LDOW09), Workshop at 18th International World Wide Web Conference (WWW09)*, Madrid, Spain.

BIZER C., VOLZ J., KOBILAROV G. & GAEDKE M. (2009). Silk - a link discovery framework for the web of data. In *18th International World Wide Web Conference*.

FERRARA A., NIKOLOV A. & SCHARFFE F. (2011). Data linking for the semantic web. *Int. J. Semantic Web Inf. Syst.*, **7**(3), 46–76.

HUHTALA Y., KÄRKKÄINEN J., PORKKA P. & TOIVONEN H. (1999). Tane : An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, **42**(2), 100–111.

MANNILA H. & RAIHA K.-J. (1994). Algorithms for inferring functional-dependencies from relations. *Data & Knowledge Engineering*, **12**, 83–99.

NIKOLOV A., UREN V., MOTTA E. & DE ROECK A. (2008). Handling instance coreferencing in the KnoFuss architecture. In *Proceedings of the workshop : Identity and Reference on the Semantic Web at 5th European Semantic Web Conference (ESWC 2008)*.

SCHARFFE F. (2012, à paraître). Un ascenseur desservant chaque étape de la publication de données sur le web. *Cahiers de l'ANR*.

SCHARFFE F. & EUZENAT J. (2010). Méthodes et outils pour lier le web des données. In *Actes 17e conférence AFIA-AFRIF sur reconnaissance des formes et intelligence artificielle (RFIA), Caen (FR)*, p. 678–685.

SCHARFFE F., LIU Y. & ZHOU C. (2009). RDF-AI : an architecture for RDF datasets matching, fusion and interlink. In *Workshop on Identity and Reference in Knowledge Representation, IJCAI 2009*.

SISMANIS Y., BROWN P., HAAS P. J. & REINWALD B. (2006). Gordian : efficient and scalable discovery of composite keys. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, p. 691–702 : VLDB Endowment.

SONG D. & HEFLIN J. (2011). Automatically generating data linkages using a domain-independent candidate selection approach. In *International Semantic Web Conference (1)*, p. 649–664.

SYMEONIDOU D., PERNELLE N. & SAÏS F. (2011). Kd2r : A key discovery method for semantic reference reconciliation. In *OTM Workshops*, p. 392–401.

VATANT B., ROZAT L., BUCHER B. & VANDENBUSSCHE P.-Y. (2011). *Méthodes et indicateurs pour la sélection d'ontologies fiables et utilisables*. Rapport interne Datalift D2.1.

YAO H. & HAMILTON H. J. (2008). Mining functional dependencies from data.

Data Min. Knowl. Discov., **16**(2), 197–219.

YU Y., LI Y. & HEFLIN J. (2011). Detecting abnormal semantic web data using semantic dependency. In *ICSC*, p. 154–157.