# Supervised tracking and correction of inconsistencies in RDF data

Youen Péron, Frédéric Raimbault, Gildas Ménier, Pierre-François Marteau

▶ **To cite this version:**

Youen Péron, Frédéric Raimbault, Gildas Ménier, Pierre-François Marteau. Supervised tracking and correction of inconsistencies in RDF data. 2011. inria-00593579v1

**HAL Id: inria-00593579**

**https://inria.hal.science/inria-00593579v1**

Preprint submitted on 18 May 2011 (v1), last revised 20 May 2011 (v2)

**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.

# Supervised tracking and correction of inconsistencies in RDF data

Youen Péron, Frédéric Raimbault, Gildas Ménier, and Pierre-François Marteau

Valoria, University of South Brittany, European University of Brittany, Vannes, France
E-mail: {firstname.lastname}@univ-ubs.fr

**Abstract.** The rapid Web development is cursed by the increase of errors in published data, mainly related to inconsistencies with domain's ontologies. This problem can alter application's reliability when analysis of independent and heterogeneous RDF data is involved. Therefore, it is important to improve the quality of the published data to promote the development of Semantic Web. We introduce a process designed to detect and identify specific inconsistencies in published data, and to fix them at an ontological level. We propose a method to quantitatively evaluate domain misuses and property range problems. A diagnosis is then established and used to drive a supervised process for the correction (or possibly enhancement) of the ontology. We show the interest of this method on a case of study involving DBpedia. The results of experiments on large masses of data generated with RDF SP2BENCH benchmark validates the applicability and scalability of our method.

## 1 Introduction

Launched in 2006 by Tim Berners-Lee, the Linked Data *Linked Data*[1] movement and its recommendations is playing a lead role for publishing, sharing, and interconnecting data on the Semantic Web [**?**]. Many companies (eg. the BBC and The New York Times), organizations (eg. Wikipedia Geonames, FreeBase, UN) and governments (eg. U.S. and U.K.) have adopted the principles of *Linked Data* to publish their data using RDF[2] standard and their ontologies[3] in RDFS[4] and OWL[5]: the size (and variety) of available data sets keeps growing. April 2011, this giant global graph data has been estimated over 200 large data sets (more than 1000 triplets each), totaling 29 billion RDF triples and 400 million links[6]. Unfortunately, this uncontrolled growth is accompanied by a proliferation of errors in published data and ontological inconsistencies. This problem alters application's reliability when analysis of independent and heterogeneous RDF data is involved. Therefore, it is important to improve the quality of the published data to promote the development of Semantic Web. Our method quantitatively evaluates domain and property misuses and helps fix these problems.

Most of the research –dealing with data quality in Semantic Web– focus either on ontology (class and properties definition) or raw data (class and literal instances).

Ontology validation has been extensively studied in [**?**,**?**,**?**,**?**]. These works verify the consistency and completeness of an ontology within the assumptions of the open world and non-unique identification characterizing the semantic web : The tools are targeted to ontology designers, irrespective of the use made the publication of data.

Surprisingly, validation of raw data has been much less studied, even if it represent almost all of the published data. Some tools have been designed for validation syntactic, such as VRP [**?**] a from tool from ICS-FORTHRFDFSUITE[7]. This kind of tools checks the data compliance with

[1] `http://www.w3.org/DesignIssues/LinkedData.html`
[2] `http://www.w3.org/TR/rdf-concepts/`
[3] http://www.w3.org/TR/webont-req/#onto-def
[4] `http://www.w3.org/TR/rdf-schema/`
[5] `http://www.w3.org/TR/owl-semantics/rdf-concepts/`
[6] source : `http://www4.wiwiss.fu-berlin.de/lodcloud/`
[7] http://139.91.183.30:9090/rdf

the RDF syntax and semantic constraints (also called consistency logic) in ontologies - under the assumption that the used ontologies are errors free. The origin of the error in raw data is searched at syntactic level. Error reporting is performed individually for each erroneous instance.

The Pedantic Web's project[8] proposes diagnosis of ontology errors and (most important) recommendation to avoid them. In [**?**] the initiators of this project perform an analysis of errors found in a sample obtained by crawling : They identify four symptoms of recurring errors and propose recommendations to manage them. The authors also introduce an online tool, RDF:ALERTS[9] designed to detect these errors. Our work take part to the Pedantic Web main works : We propose to enhance the semantic analysis by adding range and domain verification for properties on non literal data (unlike [**?**] where the analysis is strictly restricted to range checking on literal data). Our analysis is performed on class instances, being subject or object of property. We also propose a way to measure the importance of errors detected so that the priority of the correction can be evaluated : If a correction has to be done, we propose a diagnosis and an adapted fix.

They propose a generic evaluation method of data, based on systematic search of some error patterns : This search is performed by SPARQL queries applied to the deductive closure of the graph of all inference rules. Their formulation is complex because they require negations (which is non trivial for SPARQL). Since we want to test our method on very large set of RDF Data, we propose a one-rule inference (hierarchy) process to speed up the detection of possible errors. We performed our test using the parallel environment Hadoop and the request language PIG. In [**?**], the authors also address the problem of data compliance to ontologies without providing any assistance to correct the ontology.

The rest of the paper is organized as follows: In paragraph 2, we recall some concepts of the semantic web and we define the notations used in this article. The third part is the description of our processing method. The evaluation and tests are introduced in the part 4. Then we conclude with a discussion in part 5.

## 2 Context

In this part, we recall some concepts of the RDF data model, RDFS and OWL vocabulary and we define the notations used in this paper.

### 2.1 rdf model

In the RDF data model, all information is expressed as a triplet $(subject, predicate, object)$. The subject and predicate of a triple are resources identified by URIs[10]. In this article, we use the prefixed URI form to simplifies the examples. A triplet object can be either a ressource or a literal. Let $\mathcal{U}$ be the set of resources and $\mathcal{L}$ the set or literals. A triplet $t$ is defined as follows :

$$t = (s, p, o) \in \mathcal{U} \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L})$$

Since a same resource can be subject or object of a triplet, a set of triplet can be represented as a graph. We address the ABox/TBox separation problem (see [**?**]) by distributing the triplet in the terminology data (TBox) and assertion data (ABox).

### 2.2 Meta Data Vocabulary

Terminology information is described using specific RDF, RDFS or OWL vocabulary as defined by W3C. We recall the terms we use in this paper :

[8] site : `http://pedantic-web.org`
[9] `http://swse.deri.org/RDFAlerts/`
[10] URI : Uniform Resource Identifier

`rdf:type` is a resource that provides an elementary system of belonging to a class. A triplet $(r, rdf : type, c)$ translates as '$r$ belongs to the class $c$'. Let $resources(c)$ be the set of resources belonging to the class $c$ and let $\mathcal{C}$ be the set of classes. The number of resources in a class $c$ is $|c|$. A same resource can belong to several classes. All resources belong to the same class owl:thing.

`rdf:Property` is a class which instances are used as predicate in a triplet. These resources are called properties. Let the set of properties be $\mathcal{P} = resources($`rdf:Property`$)$. A property's range and TBox domains are defines respectively by `rdfs:range` and `rdfs:domain`. Domain and range can be classes or even set of classes. Let $domain(p)$ be the set of graph resources that belong to the object class of `rdfs:domain` and let $range(p)$ be the set of resources belonging to the object class of `rdfs:range`. By default, the domain and the range of a property $p$ are `owl:Thing` .

`owl:ObjectProperty` is the class of properties which range cannot be a literal, but only a class instance (also called property-object in the text below).

`rdfs:subClassOf` is a property describing a subsumption relationship between a class c and a parent class f ( c ¡ f ). This is a transitive relationship :

$$\forall c_1, c_2, c_3 \in \mathcal{C}\ (c_1 \prec c_2) \wedge (c_2 \prec c_3) \implies (c_1 \prec c_3)$$

Each class can subsume itself :

$$\forall c \in \mathcal{C},\ c \prec c$$

Let $\downarrow (c)$ be the set of classes that subsume a class $c$ :

$$\downarrow (c) = \{j \in \mathcal{C}, j \prec c\}$$

The generated hierarchy is defined as $\mathcal{H}$. Let assume that, for each graph, the closure of this relationship has already been performed (as preprocessing) on the graph so that the following formula is true :

$$(\forall u \in \mathcal{U}), (\forall c, \forall f \in \mathcal{C}),\ (u \in c) \wedge (c \prec f) \implies (u \in f)$$

We used the method of reasoning described in articles [**?**] and [**?**], which enable the implementation of inference rules on large (distributed) RDF data sets.

### 2.3 Effective domain and range

Let $domain_e(p)$ be the set of resources subjects of a property $p$. Let $range_e(p)$ be the set of resources object of a property $p$.

We check 3 that the effective domain (resp. effective range) is included in the domain defined in the TBox (resp. in the range defined in the TBox). Let a domain error be defined as the occurrence of a subject resource of an object-property $p$ that does not belong to $domain(p)$. Same wise, let a range error be defined as one occurrence of an object resource of an object-property $p$ that does not belong to $range(p)$.

As a preliminary test, we have evaluated the domain and range error $\epsilon_d(p)$ and $\epsilon_r(p)$ in DBpedia for each object-property : 13.9 % of the properties has at least one domain error. These errors are depicted in 1 : axis x : the properties are sorted by descending number of errors. axis y : error number per property. Each axis has a log scale.

The shape of the curve reveals that the errors are concentrated on a small number of object-properties - left to the vertical dotted line. The fixing of a small number of object-properties can enhance greatly the general consistency of the set of data. The method described above has two drawbacks :

(i) first, it is time and space expensive (especially for large data set) because it relies on a dictionary for the classes of resources.

(ii) second, the lack of indication of which instances has caused the error, prevents a detailed tracking of the problem and therefore the design of a solution.

In the following parts of this paper, we describe another evaluation method that overcome these problems

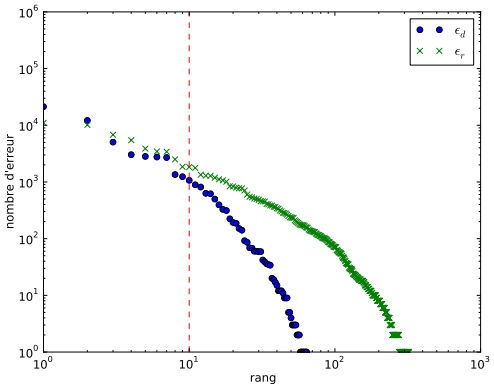


**Fig. 1.** Number of errors per property.

## 3 Our method

We propose a three-stage processing to compute the higher bound of error number per object-property. Then we show how to use it to diagnose the source of errors.

To illustrate this method, we'll describe each step of application on the DBpedia's object-property `dbpedia:hometown` : Its definition's domain is `dbpedia:Person`, but in reality it is used with 21287 resources that do not belong to this class.

Dans la figure 2 sont résumés les résultats obtenus après application de notre méthode : dans la partie gauche le domaine de définition (la portée n'est pas représenté) de la propriété `dbpedia:hometown` en pointillé et son domaine effectif en hachuré ; dans la partie droite la hiérarchie entre les classes concernées avec le nombre de ressources qui utilisent cette propriété.

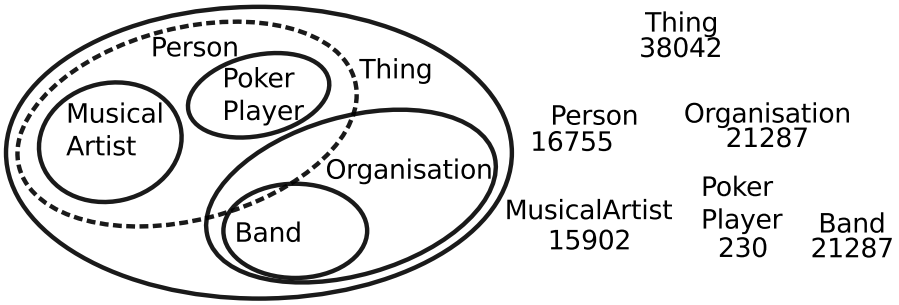


**Fig. 2.** Class `dbpedia:Person` is the domaine de definition of `dbpedia:hometown`. the effective domain, hatched shaded, is divided between classes: `dbpedia:Person` and `dbpedia:Band`. The values appearing in the class hierarchy are thos of the vector classes of property.

### 3.1 Construction des vecteurs des classes d'une propriété

La première étape consiste à calculer la répartition des ressources du domaine effectif $domain_e(p)$ et de la portée effective $range_e(p)$ d'une propriété-objet $p$ dans l'ensemble des classes $\mathcal{C}$.

On définit deux vecteurs de longueur $|\mathcal{C}|$, un vecteur $\boldsymbol{p}_d$ pour les ressources du domaine de $p$ et un vecteur $\boldsymbol{p}_r$ pour les ressources de la portée de $p$ :

$$\boldsymbol{p}_d[c] = |ressources(c) \cap domain_e(p)|$$
$$\boldsymbol{p}_r[c] = |ressources(c) \cap range_e(p)|$$

Dans le cas de la propriété-objet `dbpedia:homeTown`, les valeurs du vecteur de domaine sont indiquées sur l'arbre de la figure 2 pour les classes concernées et valent 0 pour les autres classes du graphe.

De manière générale, le maximum de ces vecteurs (de domaine et de porté) est pour la classe `owl:Thing` car elle inclut toutes les instances membre du domaine et de la portée effective.

$$\boldsymbol{p}_d[\texttt{owl:Thing}] = \max_{c \in \mathcal{C}} \{\boldsymbol{p}_d[c]\}$$
$$\boldsymbol{p}_r[\texttt{owl:Thing}] = \max_{c \in \mathcal{C}} \{\boldsymbol{p}_r[c]\}$$

Dans l'exemple de DBpedia, 38042 ressources utilisent la propriété-objet `dbpedia:HomeTown`, réparties globalement entre des personnes et des organisations.

### 3.2 Recherche de la classe la plus spécifique

La construction des vecteurs des classes d'une propriété-objet $p$ sert, dans une seconde étape, à déterminer la classe la plus spécifique incluant le domaine ou la portée effective de $p$.

Trouver une autre notation pour $domain_e^*$ ?

On note cette classe $domain_e^*(p)$. On l'obtient en recherchant la classe la plus basse dans la hiérarchique $\mathcal{H}$ parmi les classes contenant le domaine effectif de cette propriété. Par construction $domain_e^*(p)$ vérifie à la fois les relations suivantes.

$$(i)\ \ \boldsymbol{p}_d[domain_e^*(p)] = \boldsymbol{p}_d[\texttt{owl:Thing}]$$
$$(ii)\ (\forall c \in \mathcal{C})(c \neq domain_e^*(p))\ (\boldsymbol{p}_d[c] < \boldsymbol{p}_d[\texttt{owl:Thing}]) \vee (domain_e^*(p) \prec c)$$

La relation $(i)$ assure que $ressources(domain_e^*(p))$ inclut $domain_e(p)$ et la relation $(ii)$ assure que $domain_e^*(p)$ est la classe la plus spécifique des classes qui incluent $domain_e(p)$. Comme nous avons fait l'hypothèse que les classes des instances respectent une hiérarchie, on a :

$$c_1 \prec c_2 \implies (\boldsymbol{p_d}[c_1] \leq \boldsymbol{p_d}[c_2])$$

ce qui prouve que $domain_e^*(p)$ existe et est unique pour toute propriété $p$.

La preuve est un peu rapide ?

Dans l'exemple de la figure 2 la classe la plus spécifique du domaine de la propriété-objet `dbpedia:homeTown` incluant le domaine effectif est `owl:Thing`.

De même que pour le domaine, on définit $range_e^*(p)$ la classe la plus spécifique incluant la portée réelle de $p$ par :

$$(i)\ \ \boldsymbol{p}_r[range_e^*(p)] = \boldsymbol{p}_r[\texttt{owl:Thing}]$$
$$(ii)\ (\forall c \in \mathcal{C})(c \neq range_e^*(p)\ (\boldsymbol{p}_r[c] < \boldsymbol{p}_r[\texttt{owl:Thing}]) \vee (range_e^*(p) \prec c)$$

### 3.3 Recherche de la borne supérieure du nombre d'erreur

À partir de la répartition des ressources utilisant la propriété $p$ dans les classes de $\mathcal{C}$ (voir paragraphe 3.1) et des classes les plus spécifiques (voir paragraphe 3.2), la troisième étape est le calcul d'une borne supérieure du nombre d'erreurs $\varepsilon_d^{sup}(p)$ associées au domaine de $p$, et du nombre d'erreurs $\varepsilon_r^{sup}(p)$ associées à sa portée.

$$\varepsilon_d^{sup}(p) = \boldsymbol{p}_d[domain_e^*(p)] - |domain(p)|$$
$$\varepsilon_r^{sup}(p) = \boldsymbol{p}_r[range_e^*(p)] - |range(p)|$$

$\varepsilon_d^{sup}(p)$ et $\varepsilon_r^{sup}$ sont des bornes supérieures de $\varepsilon_d(p)$ et $\varepsilon_r(p)$. L'écart entre le nombre d'erreurs et les bornes supérieures calculées provient des ressources du domaine ou de la portée effectif qui appartiennent à plusieurs classes : ces ressources peuvent être comptabilisées plusieurs fois suivant la hiérarchie des classes auxquelles elles appartiennent.

Dans l'exemple de la figure 2 le domaine de définition est la classe `dbpedia:Person`. On obtient :

$$\varepsilon_d^{sup}(\texttt{dbpedia:homeTown}) = \boldsymbol{p}_d[\texttt{owl:Thing}] - \boldsymbol{p}_d[\texttt{dbpedia:Person}] = 21287$$

On conclu sur cet exemple qu'il y a au plus 21287 ressources qui apparaissent comme sujet de la propriété `dbpedia:hometown` alors qu'elles ne sont pas dans son domaine de définition (`dbpedia:Person`). On montre dans le paragraphe 4.1 que cette borne supérieure coïncide exactement avec le nombre d'erreurs obtenues par la méthode de comptage naïve.

### 3.4 Diagnostique des erreurs et correction

Après quantification de l'erreur, il reste à déterminer son origine, en proposant un diagnostique basé sur une classification des erreurs, puis à proposer une solution adaptée.

Par exemple, on cherche à diagnostiquer l'erreur détectée sur le domaine de définition de la propriété `dbpedia:hometown` de la figure 2. D'après le vecteur de répartition du domaine effectif de `dbpedia:hometown`, toutes les erreurs proviennent des instances de la classe `dbpedia:Band` qui utilisent la propriété `dbpedia:hometown` sans être membre de son domaine de définition (`dbpedia:Person`).

Une première solution est de considérer que toute ressource utilisée comme sujet d'une propriété $p$ appartient à la classe du domaine de définition de cette propriété. Celà revient à appliquer de manière systématique la règle d'inférence suivante :

$$r \in domain_e(p) \implies r \in domain(p)$$

Appliquée à la propriété `dbpedia:hometown` cela signifierait que tout groupe de musique (instance de `dbpedia:Band`) est aussi une personne (instance de la classe `dbpedia:Person`). Par exemple la ressource `dbpedia:The_Beach_Boys` utilise la propriété `dbpedia:hometown` et n'est pas une instance de la classe `dbpedia:Person`. Si on infère systématiquement que la ressource `dbpedia:The_Beach_Boys` appartient à la classe `dbpedia:Person` tout résultat de recherche ou croisement de données portant sur des `dbpedia:Person` sera faussé.

Une seconde solution est d'utiliser deux propriétés différentes pour les groupes de musique et pour les personnes. Mais dans les deux cas la propriété `dbpedia:hometown` possède la même sémantique. Ce n'est pas une solution satisfaisante au niveau ontologique car on duplique la propriété alors qu'elle ne possède qu'un seul sens.

Une troisième solution est d'assouplir le domaine de définition de `dbpedia:hometown`. Dans la hiérarchie des classes de DBpedia la seule classe moins spécifique que `dbpedia:Person` est `owl:Thing`. Or la propriété `dbpedia:hometown` n'est pas applicable à toutes les classes de DBpedia.

Une quatrième solution est de compléter le domaine de définition de la propriété `dbpedia:hometown` en y ajoutant la classe `dbpedia:Band`. Cette solution nous semble, dans ce cas particulier, la plus adéquate.

De manière générale, nous proposons un diagnostique d'une erreur sur le domaine de définition d'une propriété [11] basé sur l'arbre de décision suivant :

**si** il est raisonnable de considérer que toutes instances du domaine effectif de la propriété appartienne à la classe du domaine de définition alors on applique la règle d'inférence écrite ci-dessus,

**sinon si** la propriété ne possède pas la même sémantique pour toutes les ressources du domaine effectif alors on créée autant de propriétés que de sémantique,

**sinon si** la classe la plus spécifique incluant le domaine effectif satisfait à la sémantique de la propriété alors on élargit le domaine de définition à cette classe,

**sinon si** on trouve une classe incluant une partie du domaine effectif, qui satisfait à la sémantique de la propriété et pour laquelle le nombre d'erreurs est acceptable alors on élargit le domaine de définition à cette classe,

**sinon** on complète le domaine de définition de la propriété en incluant les classes des ressources du domaine effectif.

## 4 Tests et résultats

Dans cette partie nous analysons la qualité des résultats obtenus par notre approche sur DBpedia (version 3.6), puis nous validons son passage à l'échelle sur le benchmark SP2BENCH [**?**].

[11] On utilise des questions similaires pour diagnostiquer une erreur sur la porté d'une propriété.

### 4.1 Expérimentations sur dbpedia

Les membres du projet DBpedia ont construit collaborativement une ontologie[12] contenant 272 classes hiérarchisées par des relations de subsomption, 629 propriétés-objet et 706 propriétés-littérales. Le domaine et la portée de chaque propriété sont définies par une classe ou un ensemble de classes. Les données présentes dans 843000 articles de Wikipedia (version anglaise) sont extraites et projetées automatiquement dans cette ontologie ce qui produit un graphe de $3,5$ millions de ressources.

Nous avons appliqué à cet ensemble de données les différentes étapes de la méthode décrite au paragraphe 3 en déterminant le domaine effectif et la portée effective de toutes les propriétés-objet puis en calculant pour chacune d'elles la borne supérieure du nombre d'erreurs présentes.

**Comparaison avec le nombre réel d'erreurs** Le tableau 1 contient, pour les propriétés ayant le plus grand nombre d'erreurs, la borne supérieure calculée par notre méthode ($\varepsilon_d^{sup}(p)$) et le nombre exact d'erreurs ($\varepsilon_d(p)$) calculé par la méthode de comptage naïve du paragraphe 2.3.

| Erreurs sur le domaine de $p$ | $\varepsilon_d(p)$ | $\varepsilon_d^{sup}(p)$ |
|---|---|---|
| dbpedia:class | 166 004 | 166 004 |
| dbpedia:hometown | 21 287 | 21 287 |
| dbpedia:network | 12 171 | 12 171 |
| dbpedia:sisterStation | 5 044 | 5 044 |
| dbpedia:designer | 3 028 | 3 028 |
| **Erreurs sur la portée de $p$** | **$\varepsilon_r(p)$** | **$\varepsilon_r^{sup}(p)$** |
| dbpedia:city | 12 168 | 12 168 |
| dbpedia:associatedMusicalArtist | 11 027 | 11 027 |
| dbpedia:artist | 10 166 | 10 166 |
| dbpedia:associatedBand | 6 809 | 6 809 |
| dbpedia:sisterStation | 5 464 | 5 464 |

**Table 1.** Nombre d'erreurs sur la portée des cinq propriétés les plus concernées : par la méthode naive ($\varepsilon_d(p)$) et par notre méthode ($\varepsilon_d^{sup}(p)$).

On constate que notre borne supérieure s'avère être le nombre exact d'erreurs. Ceci s'explique par le fait que les ressources utilisées dans toutes ces propriétés n'appartiennent qu'à une seule classe.

**Étude de cas** Nous analysons ci-dessous quelques cas d'erreurs rencontrés dans DBpedia en utilisant le diagnostique de l'origine des erreurs présenté dans le paragraphe 3.4

`dbpedia:architect` est prévue à l'origine pour instances de bâtiments (classe `dbpedia:Buildings`). Mais elle est utilisée avec des instances de lieux (classe `dbpedia:Place`) qui est une classe parente de `dbpedia:Buildings`. Il est raisonable de considérer que tous les lieux ayant un architecte sont des bâtiments.
Par exemple la place Montgomery à New york est une instance de la classe `dbpedia:Place`. Bien qu'elle possède par ailleurs un architecte elle n'est pas déclarée comme étant un bâtiment (instance de la classe `dbpedia:Building`). On peut inférer que la place Montgomery est bien un bâtiment sans que cela introduise des erreurs. Au contraire, l'inférence précise la classe de la ressource.

`dbpedia:class` est la propriété à l'origine du plus grand nombre d'erreurs. La notion de classe est utilisée dans deux contextes différents. L'ontologie prévoit l'utilisation de cette propriété avec des moyens de transport. Les vecteurs de répartition des ressources du domaine effectif

[12] `http://DBpedia.org/Downloads36`

nous indiquent que la propriété est majoritairement utilisée avec des instances sous-classes de `dpedia:Species` qui n'a pas de rapport hiérarchique avec les moyens de transports.
Les espèces ne sont pas (sauf exception comme les chevaux) des moyens de transport. Dans ce cas il n'est pas raisonnable d'appliquer systématiquement la règle d'inférence du paragraphe 3.4.
La propriété `dbpedia:class` possède deux sémantiques différentes suivant qu'elle s'applique à des instances de la classe `dbpedia:Species` ou à des instances de la classe `dbpedia:MeansOfTransport`. Cette propriété est donc ambiguë : la solution que nous proposons est de créer une propriété différente pour chaque contexte dans la TBox et d'utiliser la propriété adaptée au contexte dans l'ABox.

`dbpedia:city` est définie avec comme portée la classe `dbpedia:City`. Les ressources de sa portée effective sont majoritairement des instances de lieux (classe `dbpedia:Place` une classe parente de `dbpedia:City`).
Par exemple le triplet (`Cincinnati_Bengals`,`dbpedia:city`,`Paul_Brown_Stadium`) indique que l'équipe de football américain `Cincinnati_Bengals` est basée au stade `Paul_Brown_Stadium`.
On ne peut pas inférer que tous les lieux utilisés comme objet de la propriété `dbpedia:city` sont des villes. La propriété semblant posséder la même sémantique pour toutes les ressources de la portée effective il n'est pas utile de la dupliquer. La classe la plus spécifique incluant la portée effective est `owl:Thing`. Cette classe est trop générique pour être utilisé comme portée.
La classe `dbpedia:Place` est le meilleur compromis entre le nombre d'erreurs (seulement 37 ressources de la portée effective de `dbpedia:city` ne sont pas des instances de la classe `dbpedia:Place`) et le respect de la sémantique de la propriété.

### 4.2 Expérimentations sur sp2bench

Nous avons codés notre méthode en PIG [?] et réalisé des expérimentations sur un cluster de 16 machines pouvant exécuter jusqu'à 250 tâches mapreduce Hadoop [?] en parallèle.

Pour étudier le temps d'exécution de notre méthode en fonction de la taille du graphe nous l'avons appliqué à une série de jeux de tests de taille croissante généré avec le programme SP2BENCH [?]. Le benchmark SP2BENCH a été prévu à l'origine pour évaluer les moteurs de requêtes SPARQL. Il génère un nombre donné de triplets constituant une ABox. Nous avons modifié l'ontologie de référence (TBox) de façon à introduire des erreurs dans l'ABox.

Après génération des instances de l'ABox par le programme de SP2BENCH, une phase de pré-traitement les filtre pour ne garder que les occurrences des propriétés-objets et les relations d'appartenance d'une ressource à une classe. Puis les nouvelles relations d'appartenance sont inférées dans l'ABox en fonction des relations de subsomption définies dans la TBox.

Le calcul de l'erreur de toutes les propriétés est effectué en deux phases :

1. le calcul en parallèle des vecteurs de répartition décrit dans le paragraphe 3.1,
2. le calcul séquentiel de la borne supérieure à partir des vecteurs de répartition et de la TBox.

Le temps de calcul de la borne supérieure étant négligeable par rapport au calcul des vecteurs il n'est pas pris en compte dans nos observations.

La figure 4.2 contient la représentation graphique des résultats de cette expérimentation. Une échelle logarithmique est utilisée sur les deux axes. On observe que le temps d'exécution augmente approximativement d'un facteur 3 quand la taille des données est multipliée par un facteur $10^7$. Ce résultat valide le passage à l'échelle de notre méthode et démontre son applicabilité aux ensembles de triplet RDF les plus importants.

## 5 Conclusions et perspectives

Nous avons décrit et implémenté une méthode qui permet d'analyser la totalité d'un graphe RDF pour calculer une borne supérieure du nombre d'erreurs associées au domaine ou à la portée d'une

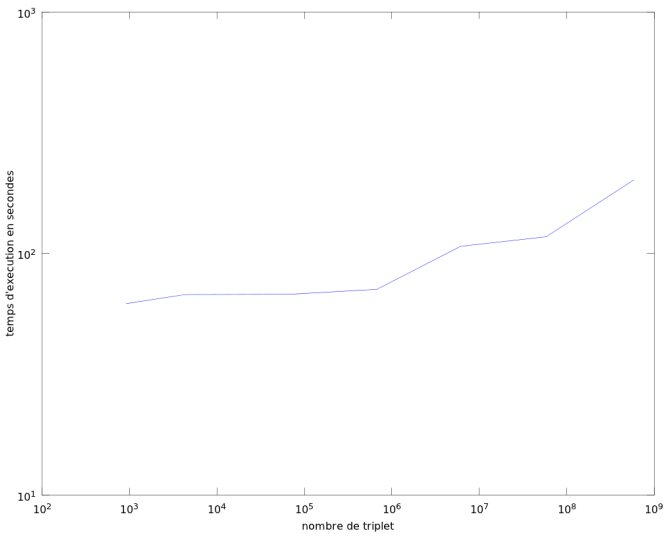


**Fig. 3.** Temps d'exécution du calcul des vecteurs de répartition en fonction du nombre de triplets générés par SP2BENCH

propriété. Pour l'ensemble des données de DBpedia cette borne supérieure correspond exactement aux nombres d'erreurs présents.

Nous avons montré comment diagnostiquer l'origine de l'erreur et superviser la correction de l'ontologie. L'analyse des erreurs de DBpedia nous à permit de valider au cas par cas la pertinence des corrections qu'il faudrait apporter à l'ontologie.

Les performances obtenues par notre méthode sur de grands ensembles de données démontrent son passage à l'échelle et son applicabilité à l'évolution quantitative du web des données.

Les limites de notre approche sont de considérer la hiérarchie des classes comme correcte et que l'on peut inférer de nouvelles relations en fonction de cette hiérarchie. Il existe des systèmes pour proposer une hiérarchie entre les classes à base de clustering hiérarchique [**?**] qui peut être une piste pour valider les relations hiérarchiques entre les classes. Cependant une telle approche se baserait probablement sur les propriétés utilisées par les instances, en les considérant comme exactes. Ces deux problèmes étant duaux, une utilisation conjointe et incrémentale, propriété par propriété, des deux méthodes permettrait de vérifier à la fois les relations hiérarchiques et les propriétés.

Nous envisageons d'étendre notre approche à d'autres contraintes OWL, comme la restriction de la portée (`owl:allValueFrom`), les disjonctions (`owl:disjoint`) entre les classes, la cardinalité (`owl:cardinality`)... Concernant le diagnostique il pourrait être amélioré en sélectionnant et en affichant des exemples pertinents d'instances à l'aide d'algorithmes basé sur le page-rank[**?**]