Propagation contextuelle des propriétés pour les graphes de connaissances : une approche fondée sur les plongements de phrases

Pierre-Henri Paris, Fayçal Hamdi, Samira Si-said Cherfi

Conservatoire National des Arts et Métiers, CEDRIC

pierre-henri.paris@upmc.fr, faycal.hamdi@cnam.fr, samira.cherfi@cnam.fr

Résumé

Comme plusieurs travaux l'ont démontré, l'identité définie par la sémantique de owl:sameAs est peut-être trop rigide dans de nombreux cas, et elle devrait être considérée comme dépendante du contexte. Contrairement à l'identité classique, avec l'identité contextuelle, seules certaines propriétés peuvent être propagées entre des entités contextuellement identiques. Dans la continuité de ces travaux sur l'identité contextuelle, nous proposons une approche, fondée sur les plongements de phrases, pour trouver semi-automatiquement un ensemble de propriétés, pour un contexte d'identité donné, qui peuvent être propagées entre des entités contextuellement identiques.

Mots-clés

RDF, identité contextuelle, graphe de connaissances, données liées, plongement de phrases.

Abstract

As several studies have shown, the identity defined by owl:sameAs semantics may be too rigid in many cases, and should be considered context-dependent. Contrary to classical identity, with contextual identity, only certain properties can be propagated between contextually identical entities. In the continuity of this work on contextual identity, we propose an approach, based on sentence embedding, to find semi-automatically a set of properties, for a given identity context, that can be propagated between contextually identical entities.

Keywords

RDF, contextual identity, knowledge graph, linked data, sentence embedding.

1 Introduction

Les graphes de connaissances RDF tels que Wikidata [1] ou DBpedia [2] sont en croissance constante en matière de nombre et d'usage. Cette croissance va de pair avec une augmentation du nombre d'entités décrites dans ces graphes et entraîne un problème pour les éditeurs de données et leurs utilisateurs : **comment savoir si deux entités sont les mêmes ou pas ?** Pour interconnecter les graphes de connaissances, la propriété *owl:sameAs* a été définie par le W3C [3] en 2004 pour relier des entités qui sont les mêmes. En effet, un objet (du monde réel) peut être décrit parmi plusieurs graphes de connaissances, et ces descriptions sont liées entre elles grâce à la propriété *owl:sameAs*. Cependant, la définition sémantique de *owl:sameAs* est très stricte. Elle est fondée sur la définition de l'identité de Leibniz, c'est-à-dire l'identité des indiscernables :

$$\forall x, \forall y(\forall p, \forall o, (\langle x, p, o\rangle \; et \; \langle y, p, o\rangle) \rightarrow x = y) \qquad (1)$$

Et sa réciproque, l'indiscernabilité des identiques :

$$\forall x, \forall y(x = y \rightarrow \forall p, \forall o, (\langle x, p, o\rangle \rightarrow \langle y, p, o\rangle)) \qquad (2)$$

C'est pourquoi deux entités sont considérées comme identiques si et seulement si elles partagent tous leurs couples $\langle$*propriété, valeur*$\rangle$ dans tous les contextes possibles et imaginables. En d'autres termes, deux identités sont identiques si **toutes leurs propriétés sont indiscernables** pour chaque valeur. Une fois qu'un lien d'identité est établi entre deux entités, il est possible d'utiliser les couples $\langle$*propriété, valeur*$\rangle$ de l'une sur l'autre et inversement. C'est ce que l'on nomme "propagation" dans cet article. Cependant, il faut noter que c'est une affirmation très forte que d'établir que deux objets sont identiques quel que soit le contexte. D'un point de vue philosophique, il y a de multiples contre-arguments à la définition de l'identité selon Leibniz. Par exemple, si nous considérons deux verres d'un même ensemble, ils sont indiscernables dans leurs fonctions et apparences, mais ils n'en sont pas moins deux objets physiques distincts. Ou encore, un bateau dont on changerait toutes les pièces au cours du temps sera-t-il toujours le même au cours du temps ?
Il s'agit aussi d'un problème technique à cause de l'hypothèse du monde ouvert ([7]) d'une part, et de l'intention de l'éditeur de données d'autre part. En effet, l'éditeur pourra avoir en tête une intention d'utilisation différente de celle de l'utilisateur. En outre, lorsque les données sont

1. https://www.wikidata.org
2. https://wiki.dbpedia.org/
3. https://www.w3.org/TR/owl-ref/

publiées, il est presque impossible de connaître le **consensus** qui sous-tend la décision de créer un tel lien. Plusieurs travaux ([11] ou [6]) ont démontré que, dans certains cas, l'utilisation de *owl:sameAs* était inadéquate. En effet, les liens établis ne peuvent être considérés comme vrais que dans des contextes spécifiques. De nos jours, le problème de l'identité reste l'un des plus importants pour l'industrie travaillant avec des graphes de connaissances ([18]).
En première approximation, une identité contextuelle entre deux entités pourrait être considérée comme un sous-ensemble Π des propriétés de ces entités pour lesquelles les entités partagent les mêmes valeurs pour chaque $p \in \Pi$. Π est l'ensemble d'indiscernabilité.

Exemple 1 *Deux médicaments génériques différents Drug1 et Drug2 peuvent être identiques en ce qui concerne leur principe actif. Si un graphe de connaissances contient les triplets ⟨ Drug1 activeIngredient Molecule1 ⟩ et ⟨ Drug2 activeIngredient Molecule1 ⟩, alors* $Drug1 \equiv_{activeIngredient} Drug2$ *lorsque l'ensemble d'indiscernabilité est*
$\Pi = \{activeIngredient\}$.

L'une des principales caractéristiques de *owl:sameAs* est de pouvoir **propager toutes les propriétés** d'une entité vers d'autres entités identiques. Si $A = B$, alors toutes les caractéristiques de A sont aussi valables pour B, et inversement. Ainsi, *owl:sameAs* permet de découvrir plus de connaissances et d'accroître la complétude. De la même manière, l'identité contextuelle doit aider à découvrir **plus de connaissances et à accroître la complétude**, mais seulement dans des circonstances spécifiques. Pour être utile, une identité contextuelle doit préciser ce qui se passe avec les propriétés qui ne font pas partie de l'ensemble d'indiscernabilité. En d'autres termes, **une identité contextuelle** doit aussi pouvoir permettre de **propager certaines propriétés**.

Exemple 2 *En reprenant l'exemple 1, établir seulement* $Drug1 \equiv_{activeIngredient} Drug2$ *est d'un intérêt limité, puisqu'en dehors de* $activeIngredient$*, nous ne savons pas quoi faire des autres propriétés des médicaments. En considérant que* $activeIngredient$ *est l'ensemble d'indiscernabilité, nous savons, en tant qu'être humains, que la propriété* $targetDisease$ *est propageable, et nous pouvons conclure que si la déclaration ⟨ Drug1 targetDisease Disease1 ⟩ existe alors* $\langle Drug2\ targetDisease\ Disease1\rangle$ *aussi. A l'inverse, nous savons que la propriété* $excipient$ *n'est pas nécessairement propageable.*

De plus, la capacité à propager une propriété entre entités dépend du contexte, c'est-à-dire que la même propriété peut se propager dans un contexte C_1 et ne pas se propager dans un autre contexte C_2. Par exemple, la propriété "maladie ciblée" se propage entre deux médicaments si le contexte est la propriété "principe actif". Mais si le contexte est "produit par", alors "maladie ciblée" ne sera très certainement pas propageable entre deux médicaments.

Plusieurs travaux ont tenté de proposer une solution à l'identité contextuelle. [2], [14] et [20] ont défini trois façons différentes de traiter l'identité dans un contexte donné. Mais aucun de ces travaux ne propose de solutions pour découvrir des propriétés qui peuvent être propagées dans un contexte spécifique.
Questions de recherche : Avec un contexte d'identité donné entre deux entités, comment trouver les propriétés qui peuvent être propagées ? Est-il possible de trouver ces propriétés propageables (semi-)automatiquement ?
Dans cet article, en reprenant la définition de [14], nous proposons une approche pour **trouver les propriétés propageables** afin de faciliter la découverte de connaissances pour les utilisateurs de graphes de connaissances. Nous utilisons une méthode de plongement de phrases existante fondée sur les réseaux de neurones pour découvrir des propriétés propageables pour un contexte donné. Nous avons validé notre approche avec des expériences qualitatives.
Le reste du document est organisé comme suit. Dans la section suivante, nous présentons les travaux connexes. Dans la section 4, nous présentons notre approche. Dans la section 5, nous présentons les expériences qualitatives que nous avons menées. Enfin, nous concluons et définissons les prochaines orientations de nos travaux futurs.

2 Travaux connexes

Dans la première partie de cette section, nous décrivons les articles qui ont souligné les problèmes soulevés par l'usage de *owl:sameAs*. La deuxième partie concerne les propositions qui s'attaquent à ces problèmes.

2.1 Crise de l'identité

Comme décrit dans [13], le but de la propriété *owl:sameAs* est de relier deux entités qui sont strictement identiques, c'est-à-dire que les deux entités sont identiques dans tous les contextes possibles. *owl:sameAs* a une sémantique stricte permettant de déduire de nouvelles informations. De nombreux outils existants produisent de tels liens *owl:sameAs* ([9]), et plusieurs enquêtes sont disponibles (voir [9], [1] et [17]).
Toutefois, aucune de ces approches ne tient compte des liens d'identité contextuels. Leur but est de découvrir des liens d'identité qui seraient toujours valables. Ceci est, d'un point de vue philosophique, difficile à obtenir comme le souligne la définition de l'identité de Leibnitz.
Dès 2002, [10] a soulevé la question de l'identité pour les ontologies. Surtout lorsque le temps est impliqué, affirmer que deux choses sont identiques devient un problème philosophique. Les auteurs ont proposé de n'impliquer dans l'identité que les propriétés essentielles, c'est-à-dire les propriétés qui ne peuvent pas changer. Comme indiqué par exemple dans [11] ou [6], en raison de la sémantique stricte de *owl:sameAs*, la charge des éditeurs de données pourrait être trop lourde. En fait, ces liens ne sont pas souvent utilisés de manière adéquate. Certains peuvent être simplement erronés et, plus insidieusement, certains peuvent dépendre du contexte, c'est-à-dire que le lien ne tient pas dans tous les contextes possibles parce qu'il est difficile d'obte-

nir un consensus sur la validité d'une déclaration. Le sens donné par un modélisateur de données peut ne pas correspondre à ce qu'attend un utilisateur de données. Cette utilisation abusive de *owl:sameAs* est souvent appelée "crise de l'identité" ([11]).

2.2 Identité contextuelle

[2] ont abordé cette question en construisant un treillis de contextes d'identité où les contextes sont définis comme des ensembles de propriétés. Cela correspond à la première approximation proposée dans la Section 1. Toutes les entités identiques dans un contexte partagent les mêmes valeurs pour chaque propriété de ce contexte. Ainsi, un contexte est un ensemble de propriétés indiscernables pour une entité. Cependant, les auteurs ne donnent pas d'indications sur l'utilisation de propriétés n'appartenant pas à de tels contextes. [20] ont proposé un algorithme appelé DECIDE pour calculer les contextes, où les contextes d'identité sont définis comme des sous-ontologies. Mais comme dans le précédent article, les propriétés des entités qui ne sont pas dans la sous-ontologie sont ignorées. Ainsi, dans les deux travaux précédents, il y a une limitation concernant les propriétés qui n'appartiennent pas à un contexte. Cette limitation réduit l'intérêt d'utiliser de telles approches. En effet, l'un des objectifs d'un contexte d'identité est de définir une relation d'identité entre deux entités afin d'utiliser les informations de l'une à l'autre. La solution de [14] implique une telle propagation des propriétés, et donc, augmente la complétude d'une entité selon un contexte. Toutefois, cette proposition exige que l'utilisateur donne en entrée les propriétés de propagation et les propriétés indiscernables. Elle laisse donc à l'utilisateur la charge d'identifier et de fournir le contexte et les propriétés. L'utilisateur doit fournir les deux ensembles de propriétés indiscernables et propageables.
Dans ce travail, nous proposons d'enlever partiellement cette charge à l'utilisateur, c'est-à-dire de **calculer semi-automatiquement l'ensemble de propagation des propriétés étant donné un ensemble de propriétés indiscernables**. Pour cela, nous utiliserons une approche de plongement de phrases (présentée dans la section 4.3) pour calculer les plongements des (descriptions des) propriétés afin de découvrir les **propriétés propageables** pour un contexte d'identité donné (tel que défini dans [14]).

3 Motivation

Parfois, les entités du monde réel peuvent être proches quant à leurs propriétés, mais pas exactement les mêmes. Par exemple, la capitale française, Paris, est à la fois une ville et un département. Tout en considérant que la ville et le département sont les mêmes en ce qui concerne leur géographie, ils sont deux entités distinctes sur le plan administratif (ou juridique). Maintenant, supposons que les deux Paris soient représentés dans un graphe de connaissances en tant qu'entités distinctes, et que les deux soient liés à des cinémas (éventuellement distincts). Si l'on veut récupérer les salles de cinéma situées dans la ville de Paris, les résultats ne seront pas complets si certains d'entre eux

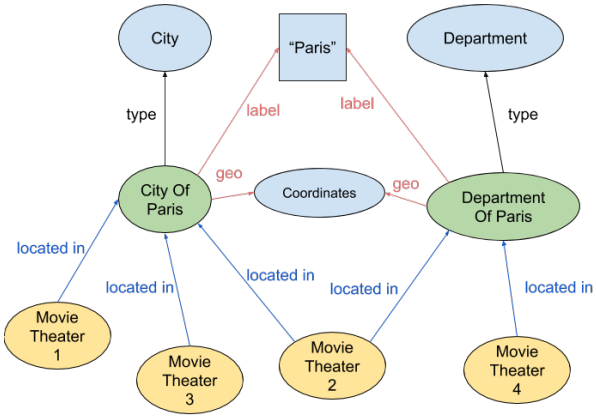

FIGURE 1 – Extrait d'un graphe de connaissances sur Paris. Les propriétés en rouge sont indiscernables pour la ville et le département. Les propriétés en bleu se propagent tant que les propriétés en rouge sont indiscernables.

sont liés au département (voir Figure 1).
Un citoyen français peut connaître cette vérité, mais comment permettre à un agent automatisé de découvrir ce fait ? L'identité contextuelle est une réponse possible à cette question, c'est-à-dire un ensemble de propriétés pour lesquelles les valeurs sont les mêmes pour les deux entités. Dans le présent exemple, les deux Paris (ville et département) sont géographiquement identiques et certaines propriétés liées à la géographie pourraient être **propagées**. Dans la figure 1, les propriétés rouges (*geo* et *label*) sont indiscernables (ont les mêmes valeurs) et les propriétés bleues (*located in*) se propagent. Dans le monde réel, les cinémas situés dans la ville ou le département selon le graphe de connaissances sont en fait situés au même endroit. Malgré le fait que les deux entités ne partagent pas les mêmes valeurs pour la propriété *located in*, celle-ci est liée au contexte géographique. En effet, pour un agent humain, la propriété *located in* pourrait être évidemment propagée entre les deux entités.
Alors que nous savons que les quatre cinémas sont situés à Paris, la requête dans Listing 1 ne donnera que les cinémas 1, 2 et 3 (voir Figure 1).

```
SELECT DISTINCT ?movieTheater WHERE {
    ?movieTheater :locatedIn
    :CityOfParis . }
```

Listing 1 – Requête SPARQL récupérant tous les cinémas de Paris.

Ainsi, la découverte de tels contextes d'identité entre entités pourrait améliorer les résultats des requêtes. Notre intuition est inspirée par la première loi de Tobler ([24]), c'est-à-dire :

> "*Tout interagit avec tout, mais deux objets proches ont plus de chances de le faire que deux objets éloignés.*"

Par conséquent, **nous émettons l'hypothèse que, d'un point de vue sémantique, plus une propriété est proche**

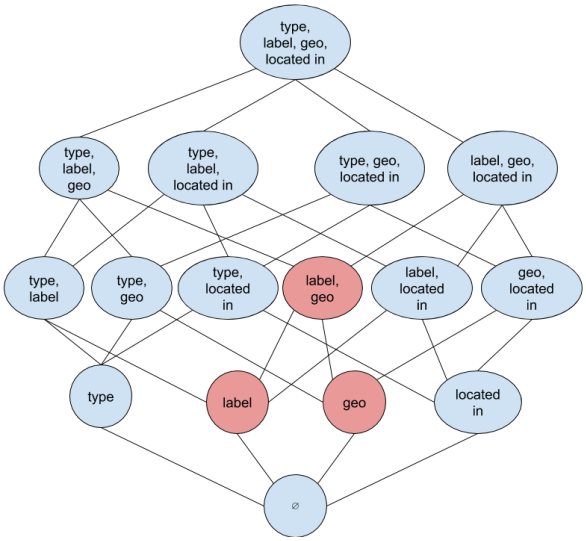

FIGURE 2 – Treillis d'identité simplifié de la figure 1 : chaque nœud est un ensemble de propriétés. Seuls les nœuds rouges ont des entités similaires.

du contexte d'identité, plus elle est susceptible d'être une bonne candidate à la propagation. Dans l'exemple précédent, *located in* fait clairement référence à la géographie, et le contexte d'identité concerne la géographie puisqu'il est composé de coordonnées géographiques. L'idée est donc de **calculer une distance entre les propriétés indiscernables** et **les propriétés candidates à la propagation**. Par conséquent, les nombres, et dans notre cas les vecteurs numériques sont les mieux adaptés pour calculer cette distance. Une représentation numérique de la description textuelle de chaque propriété grâce à la valeur donnée par $rdfs{:}comment$ ou $schema{:}description$ peut fournir une base pour obtenir ce vecteur. En effet, le plongement des descriptions des propriétés nous donne des vecteurs numériques dont la distribution dans l'espace vectoriel respecte la similarité sémantique des phrases.
Dans ce qui suit, nous décrirons notre approche avec plus de détails.

4 Approche

Dans cette section, avant de plonger plus avant dans l'approche de base, nous donnons quelques définitions nécessaires par la suite pour décrire l'approche.

4.1 Préliminaires

Comme nous l'avons vu dans la section 2, plusieurs propositions ont été faites pour définir un contexte d'identité. Nous avons choisi celle de [14] (Définition 1) car elle est la seule à prendre en compte la propagation des propriétés. Ils donnent la définition suivante du contexte d'identité :

Définition 1 ***(Contexte d'identité)*** *Un contexte d'identité $\mathcal{C} = (\Pi, \Psi, \approx)$ est défini par deux ensembles de propriétés (Π et Ψ) et une **procédure d'alignement** ($\approx$). Π est l'**ensemble d'indiscernabilité** des propriétés (équation 3) et Ψ est l'**ensemble de propagation** des propriétés (equation 4). Dans la suite, x et y sont des entités.*

$$x =_{(\Pi,\Psi,\approx)} y \leftrightarrow \forall (p_1, p_2) \in \Pi^2 \text{ avec } p_1 \approx p_2 \\ \text{et } \forall v_1, v_2 \text{ avec } v_1 \approx v_2 : \langle x, p_1, v_1 \rangle \leftrightarrow \langle y, p_2, v_2 \rangle \quad (3)$$

$$x =_{(\Pi,\Psi,\approx)} y \rightarrow \forall (p_1, p_2) \in \Psi^2 \text{ avec } p_1 \approx p_2 \\ \text{et } \forall v_1, v_2 \text{ avec } v_1 \approx v_2 : \langle x, p_1, v_1 \rangle \leftrightarrow \langle y, p_2, v_2 \rangle \quad (4)$$

*De plus, nous définissons le **niveau d'un contexte** $|\Pi_{\mathcal{C}}|$ comme étant le nombre de propriétés indiscernables (dans Π).*

Dans le cas où des entités similaires selon un contexte d'identité appartiennent au même graphe de connaissances, il n'est pas nécessaire d'avoir une procédure d'alignement. Une entité peut avoir plusieurs contextes d'identité, en fonction des propriétés de l'ensemble d'indiscernabilité Π. En effet, deux combinaisons différentes de propriétés peuvent donner des ensembles différents d'entités similaires. Par exemple, si l'on choisit comme ensemble d'indiscernabilité la propriété "principe actif", pour un médicament donné med, on aura tous les médicaments qui ont le même principe actif que med. A contrario, si l'on choisit comme ensemble d'indiscernabilité la propriété "produit par" pour la graine med, on aura tous les médicaments produits par la société qui produit med. Dans les deux cas, la graine est la même (med), mais les entitiés similaires des deux ensembles d'indiscernabilités ont de grandes chances d'être différentes. Pour finir, l'ensemble d'indiscernabilité composé des deux propriétés "principe actif" et "produit par" donnera comme entités similaires celles qui sont produites par l'entreprise fabriquant med et qui ont le même principe actif. Il se peut bien entendu qu'il n'existe pas de médicament correspondant à ce cas.
Le treillis d'identité de tous les contextes d'identité d'une entité e est défini comme suit (voir Figure 2) :

Définition 2 ***(Treillis d'identité)*** *Un treillis d'identité $\mathcal{L}$ est un treillis dont chaque élément est un contexte d'identité. L'inclusion ensembliste entre les ensembles d'indiscernabilités de chaque contexte est la relation binaire responsable de l'ordre partiel.*

La dernière notion est la graine d'un treillis ou d'un contexte que nous définissons comme suit :

Définition 3 ***(Graine d'un treillis ou d'un contexte)*** *Chaque contexte d'identité d'un treillis d'identité est construit à partir de la **même entité** e. Cette entité e est appelée la graine du treillis d'identité.*

En effet, pour construire un treillis d'identité, nous devons partir d'une graine, même si le treillis pourrait potentiellement être valide avec une autre graine, c'est-à-dire que deux graines peuvent donner le même treillis d'identité.
Maintenant que nous avons défini les concepts nécessaires, nous allons expliquer le cœur de notre approche.

4.2 Calcul des contextes

Dans cette section, nous expliquons comment calculer un treillis et ses contextes.

Nous présentons l'Algorithme 1 qui permet le calcul du treillis d'identité. Il prend en entrée l'entité graine, le graphe de connaissances source auquel la graine appartient, le graphe de connaissances cible (éventuellement le même que le graphe de connaissances source) et une procédure d'alignement si les deux graphes de connaissances sont distincts. L'idée principale est de commencer par calculer les ensembles d'indiscernabilité de bas niveau avec chaque propriété de la graine et enfin de combiner ces ensembles pour obtenir des ensembles d'indiscernabilité de plus haut niveau. **Lorsqu'on construit un contexte, le premier élément est son ensemble d'indiscernabilité, à partir duquel on obtient ensuite des entités similaires, pour finalement obtenir des propriétés candidates à la propagation et, en conclusion, des propriétés de propagation.**

La première étape, ligne 3, consiste à calculer tous les contextes d'identité de niveau 1 (voir Définition 1). En effet, pour chaque propriété p de la graine, il existe exactement un contexte d'identité (son ensemble d'indiscernabilité est $\Pi = \{p\}$). Par la suite, les contextes d'identité n'ayant qu'une seule propriété d'indiscernabilité seront fusionnés pour donner des contextes d'identité de niveau 2, puis de niveau 3, etc. Ensuite, nous récupérons l'ensemble $entities_p$ des entités similaires à la graine qui ont la (les) même(s) valeur(s) pour la propriété donnée p. Si p est multi-valué (plusieurs o pour p), alors les entités dans $entities_p$ sont similaires à la graine pour toutes les valeurs o de sorte que $\langle seed\ p\ o\rangle$, c'est-à-dire $\forall p, (\forall o, \langle seed, p, o\rangle \rightarrow \forall e \in entities_p, \langle e, p, o\rangle)$. Il est à noter que, lors du remplissage de $entities_p$, nous ne recherchons que les entités qui ont le(s) même(s) type(s) que la graine. C'est parce que nous voulons éviter les résultats absurdes, par exemple comparer une personne avec un avion. Elle présente également l'avantage de réduire considérablement le nombre de contextes d'identité possibles à calculer. Enfin, sur la base de $entities_p$, nous calculons l'ensemble de propagation Ψ (ligne 12) comme expliqué dans la section suivante.

La deuxième étape (voir Algo. 2) consiste à calculer les contextes d'identité de niveau supérieur en se basant sur ceux du niveau 1. La boucle (ligne 2) de l'algorithme calcule ces niveaux supérieurs en combinant des contextes de même niveau, et s'arrête lorsqu'il ne peut pas construire de nouveaux contextes d'identité de niveau supérieur. Ce calcul est basé sur un opérateur de treillis d'identité qui est l'ensemble d'inclusion sur les ensembles d'indiscernabilité. Par exemple, un contexte de niveau 2 (deux propriétés dans Π) est construit sur deux contextes de niveau 1 (chacun une propriété dans Π). Là encore, pour réduire le nombre de contextes d'identité possibles à calculer, s'il n'y a pas d'entité similaire à la semence pour un contexte donné C_i, il n'est pas nécessaire de calculer des contextes de niveau supérieur basés sur C_i.

Data : $\mathcal{KG}_1$: the source KG, $\mathcal{KG}_2$: the target KG, $seed$: an entity of $\mathcal{KG}_1$, $\approx$: an alignment procedure between $\mathcal{KG}_1$ and $\mathcal{KG}_2$
Result : $\mathcal{L}$: a lattice of identity contexts between the seed and entities in the target KG

1 $\mathcal{L} = \emptyset$;
/* Get all explicit and implicit types of the seed */
2 $\mathcal{T}_{seed} = \{t : \langle seed\ rdf{:}type\ t\rangle \in \mathcal{KG}_1\}$;
/* the following will create all contexts of the lower level (with only one indiscernible property) */
3 **for** *each property p of seed* **do**
4 $candidateEntities = \emptyset$;
5 **for** *each value o such as* $\langle seed\ p\ o\rangle \in \mathcal{KG}_1$ **do**
/* $entities_{p,o}$ is the set of indiscernible entities with $seed$ with respect to the p, o couple */
6 $entities_{p,o} = \{e : (\exists (p', o'), p' \approx p, o' \approx o, \langle e\ p'\ o'\rangle \in \mathcal{KG}_2) \wedge (\exists t \in \mathcal{T}_{seed}, t' \approx t, \langle e\ rdf{:}type\ t'\rangle \in \mathcal{KG}_2)\}$;
7 **if** $entities_{p,o} \neq \emptyset$ **then**
8 $candidateEntities = candidateEntities \cup \{entities_{p,o}\}$;
9 **end**
10 **end**
/* $entities_p$ is the set of indiscernible entities with $seed$ with respect to the property p */
/* intersection of all sets in $candidateEntities$ */
11 $entities_p = \bigcap candidateEntities$;
12 $\Psi = getPropagationSet(seed, entities_p, \{p\})$;
13 **if** $\Psi \neq \emptyset$ **then**
14 $\Pi = \{p\}$;
15 $\mathcal{C} = (\Pi, \Psi, \approx)$;
16 $\mathcal{L} = \mathcal{L} \cup \mathcal{C}$;
17 **end**
18 **end**
/* Now we can combine contexts of the same level */
19 **return** $constructUpperLevels(\mathcal{L})$

Algorithme 1 : createLattice : calculer le treillis d'identité d'une entité.

4.3 Plongement de phrase

Notre approche étant fondée sur le plongement de phrases ("*sentence embedding*"), nous donnons dans cette section plus de détails sur cette notion. En effet, lors du calcul d'un contexte d'identité, nous calculons son ensemble de propagation correspondant à l'aide de l'algorithme 3.

Le plongement de phrases est une technique qui permet de faire correspondre une phrase à un vecteur numérique.

Data : $\mathcal{KG}_1$: the source KG, $\mathcal{KG}_2$: the target KG, $seed$: an entity of $\mathcal{KG}_1$, $\approx$: an alignment procedure between $\mathcal{KG}_1$ and $\mathcal{KG}_2$
Result : $\mathcal{L}$: a lattice of identity contexts between the seed and entities in the target KG

```
/* lvl is the current level in the
   lattice                                  */
lvl = 1;
while ∅ ∉ L do
    contexts = ∅;
    for (C1, C2) ∈ {(Ci, Cj) ∈ L^2 : |ΠCi| = |ΠCj| = lvl, i > j} do
        Π = ΠC1 ∪ ΠC2;
        /* getEntities function gives the
           set of entities that are
           similar under the given
           identity context in the
           given KG                         */
        entities = getEntities(C1, KG2) ∩ getEntities(C2, KG2);
        if entities ≠ ∅ and Π ∉ L then
            Ψ = getPropagationSet(seed, entities, Π);
            /* see Algo. 3                  */
            if Ψ ≠ ∅ then
                C = (Π, Ψ, ≈);
                contexts = contexts ∪ C;
            end
        end
    end
    L = L ∪ contexts;
    lvl = lvl + 1;
end
return L
```

Algorithme 2 : constructUpperLevels : calculer les niveaux supérieurs du treillis d'identité d'une entité.

Idéalement, les phrases sémantiquement proches sont représentées par des vecteurs proches dans l'espace vectoriel considéré.

Exemple 3 *"Un match de football auquel participent plusieurs hommes" et "Certains hommes pratiquent un sport" sont proches sémantiquement, donc leurs vecteurs devraient être proches en ce qui concerne la distance.*

Réciproquement, deux phrases qui ne sont pas apparentées sémantiquement doivent avoir des vecteurs éloignés.

Exemple 4 *"Un homme inspecte l'uniforme d'un personnage dans un pays d'Asie de l'Est" et "L'homme dort" doivent avoir des vecteurs éloignés.*

Ces vecteurs permettent d'utiliser divers opérateurs mathématiques qui ne sont évidemment pas disponibles avec des chaînes de caractères. L'un des premiers travaux importants dans ce domaine est *Word2Vec* ([16]) qui capture la cooccurrence des mots. Chaque mot est traité de manière atomique et fournit un plongement grâce à deux approches distinctes, à savoir *Skip-Gram* et *Continuous Bag of Words* (CBOW). Alors que l'objectif de CBOW est de prédire un mot en fonction de sa fenêtre (c'est-à-dire les mots précédents et suivants dans une phrase), Skip-Gram essaiera de prédire les mots avec lesquels un mot est habituellement vu. De même, *GloVe* ([19]) fournit des plongements pour des mots uniques et peut utiliser Skip-Gram ou CBOW. Mais *GloVe*, au lieu de capturer la cooccurrence, se concentre (à la fin) sur le nombre d'apparitions parmi les fenêtres (c'est-à-dire les mots précédents et suivants dans une phrase). Ensuite, *fastText* ([3]) est une extension de *Word2Vec* qui traite les mots comme n-grammes de caractères plutôt que comme entité atomique. La taille des N-grammes dépend des paramètres d'entrée. L'utilisation de N-grammes permet une meilleure compréhension des petits mots. Chaque n-gramme est mis en correspondance avec un vecteur et la somme de ces vecteurs est la représentation du mot. Un autre avantage du *fastText* est sa capacité à fournir un plongement même pour les mots inconnus, grâce à l'utilisation de n-grammes. Alors que les trois travaux précédents sont les mieux adaptés pour travailler avec des mots atomiques, le suivant calcule le plongement pour une phrase entière.

Les raisons pour lesquelles on utilise le plongement de phrases plutôt qu'une distance plus classique, par exemple la distance d'édition, le plongement de graphe RDF comme RDF2Vec ([21]), ou une technique d'alignement ontologique sont les suivantes : *(i)* les distances de chaînes de caractères classiques ignorent la sémantique des phrases, *(ii)* les techniques de plongement de graphes RDF ne sont pas encore adaptées à une telle tâche, et *(iii)* les techniques d'alignement ontologique alignent des propriétés par paires et non des ensembles de propriétés.

Une grande attention a été accordée aux plongements de phrases ces derniers temps. Des approches telles que *Universal Sentence Encoder* ([4]), *GenSen* ([23]) et *InferSent* ([5]) font partie des encodeurs de référence pour le plongement de phrases. Nous choisissons d'utiliser ce dernier, mais notre approche pourrait bénéficier de n'importe laquelle de ces approches. *InferSent*, proposé par [5], est un encodeur de pointe qui s'est révélé efficace pour le plongement de phrases. Pour entraîner leur modèle supervisé de plongement de phrases, les auteurs ont utilisé l'ensemble de données SNLI (Stanford Natural Language Inference) qui consiste en plus de 500K paires de phrases anglaises étiquetées manuellement avec l'une des trois catégories : implication, contradiction et neutre. Ils ont testé plusieurs architectures et ont découvert qu'un réseau BiLSTM avec un *max pooling* offrait les meilleurs résultats. Un réseau BiLSTM est un LSTM bidirectionnel souvent utilisé pour les données séquentielles, c'est-à-dire un réseau de neurones récurrent (avec des boucles). Le *max pooling* est une technique qui permet de réduire le nombre de paramètres du modèle en sélectionnant la valeur maximale d'une "fenêtre" mobile. De plus, le modèle pré-entraîné est basé sur fastText, ce qui permet de calculer des représentations significatives même pour des mots hors vocabulaire, c'est-

Data : $seed$: the entity that generated Π,
$entities$: set of entities similar to $seed$ with respect to Π,
Π : an indiscernibility set
Result : Ψ : a propagation set
/* computation of the embeddings of each property in Π by using one of the encoder */
1 $indiscernibilityEmbeddings \leftarrow getEmbeddings(\Pi)$;
2 $meanVector \leftarrow mean(indiscernibilityEmbeddings)$;
/* $getCandidateProperties$ function returns the set of all candidate properties for propagation */
3 $candidates \leftarrow getCandidateProperties(\Pi, \{seed\} \cup entities)$;
/* then compute their embeddings */
4 $candidatesEmbeddings \leftarrow getEmbeddings(candidates)$;
5 $\Psi \leftarrow \emptyset$;
6 **for** *candidate in* $candidatesEmbeddings$ **do**
7 $similarity \leftarrow cosineSimilarity(candidate, meanVector)$;
8 **if** $similarity \geq threshold$ **then**
9 $\Psi \leftarrow \Psi \cup \{candidate\}$;
10 **end**
11 **end**
12 **return** Ψ
Algorithme 3 : getPropagationSet : calculer l'ensemble de propagation.

à-dire des mots qui n'apparaissent pas dans les données d'entraînement. *GenSen* ([23]) et *Universal Sentence Encoder* ([4]) sont tous deux basés sur l'apprentissage multitâche (MTL). Le but de MTL est d'apprendre de multiples aspects d'une phrase en alternant entre différentes tâches comme la traduction ou l'inférence en langage naturel. Les premiers utilisent un GRU (Gated Recurrent Units) bidirectionnel, c'est-à-dire un réseau de neurones récurrent comme le LSTM mais avec moins de paramètres. Ce dernier utilise l'architecture *transformer* qui transforme une séquence en une autre, mais sans réseau de neurones récurrent (contrairement à *InferSent* et *GenSen*).

Comme présenté dans la section 1, notre intuition, basée sur la première loi de Tobler, est qu'un ensemble de propriétés de propagation peut être trouvé étant donné un ensemble d'indiscernabilité, si les vecteurs de description de ces deux ensembles sont suffisamment proches. Dans ce travail, **nous proposons d'utiliser les descriptions textuelles longues en langage naturel des propriétés (par exemple** $rdfs{:}comment$ **ou** $schema{:}description$**) pour trouver des propriétés qui sont sémantiquement liées** et par conséquent de bonnes candidates à la propagation pour un ensemble d'indiscernabilité donné Π. Pour le calcul du plongement, n'importe lequel des encodeurs décrits précédemment peut être utilisé.

L'algorithme 3 présente notre proposition de calculer Ψ étant donné un Π. Il prend comme entrée trois paramètres : une graine (une entité), un ensemble de propriétés construites à partir de la graine (ensemble d'indiscernabilité Π), et un ensemble d'entités qui sont similaires à la graine en ce qui concerne Π. Le calcul de Π est présenté dans la section précédente (voir algorithme 1).

Tout d'abord, pour chaque propriété de l'ensemble d'indiscernabilité Π, nous calculons son vecteur de représentation (voir ligne 1). Ensuite, nous calculons le vecteur moyen qui représente l'ensemble d'indiscernabilité (ligne 2). De même, nous considérons chaque propriété de la graine ou de ses entités similaires, et calculons leurs vecteurs de représentation. Par conséquent, d'une part, nous avons un vecteur qui représente l'ensemble d'indiscernabilité et, d'autre part, nous avons des vecteurs pour les propriétés qui sont candidates à la propagation. Les entités similaires (en ce qui concerne l'ensemble d'indiscernabilité Π) sont également considérées pour obtenir des propriétés candidates, puisque l'une d'entre elles peut éventuellement avoir une propriété de propagation que la graine n'a pas (voir ligne 3).

Ensuite, nous effectuons une boucle sur chaque propriété candidate (ligne 6) pour calculer une similarité cosinus ([22]) entre chaque vecteur candidat et le vecteur moyen représentant l'ensemble d'indiscernabilité Π. Si la similarité cosinus est suffisamment élevée (au-dessus d'un seuil spécifié, comme expliqué dans la section suivante), la propriété candidate est considérée comme une propriété de propagation (voir ligne 8).

Notre approche ayant été présentée, nous allons introduire les expériences pour valider notre travail.

5 Résultats expérimentaux

Nous présentons plusieurs requêtes SPARQL qui bénéficient de notre approche pour évaluer cette dernière. Mais tout d'abord, nous présentons succinctement notre implémentation. Il n'est pour l'instant pas possible d'évaluer quantitativement notre approche, puisque notre approche est la première à travailler sur cette problématique d'une part, et, d'autre part, il n'existe pas encore d'étalon d'or auquel nous comparer.

5.1 Implémentation et mise en place

Nous avons implémenté notre approche avec le langage Python. Dans un souci de reproductibilité, le code est mis à disposition sur un dépôt GitHub [4]. Comme mentionné précédemment, nous avons utilisé trois approches de plongement de phrases, à savoir *InferSent* [5], *GenSen* [6] et *Universal Sentence Encoder* [7]. Nous avons utilisé un fichier HDT (voir [15] et [8]) qui contient un dump de la dernière

4. https://github.com/PHParis/ConProKnow
5. https://github.com/facebookresearch/InferSent
6. https://github.com/Maluuba/gensen
7. https://tfhub.dev/google/universal-sentence-encoder/2

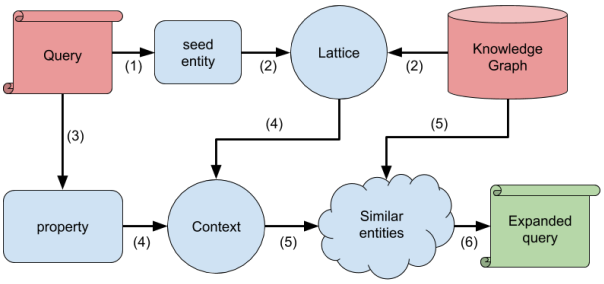

FIGURE 3 – Flux de travail de l'expérience qualitative : les éléments en rouge sont les entrées et l'élément en vert est la sortie. Pour simplifier le diagramme, nous ne considérons qu'une seule entité instanciée liée à une propriété instanciée dans la requête.

version de Wikidata [8]. HDT est un format de sérialisation compressé pour les graphes RDF qui permet une meilleure reproductibilité qu'un *endpoint* SPARQL. Contrairement à Turtle ou N-Triples, grâce à la compression, HDT facilite les manipulations nécessaires pour reproduire les expériences. L'ordinateur que nous avons utilisé est doté d'un processeur i7 et de 32 Go de mémoire vive. À titre indicatif, le calcul complet du treillis d'identité pour une entité telle que la ville de Paris prend environ 1 396 ms. Cette entité compte plus de 1 000 couples propriété-valeur et, dans Wikidata, le nombre moyen de couples propriété-valeur est d'environ 60. Il s'agit donc d'une entité assez importante et cette approche semble pouvoir être utilisée à grande échelle.

5.2 Étude qualitative

Dans cette section, nous introduisons trois requêtes différentes qui pourraient bénéficier de notre approche en étendant leurs résultats. Pour atteindre notre objectif, nous avons utilisé *InferSent* et une valeur de seuil égale à 0,9. Toutes ces requêtes sont des requêtes simplifiées testées sur Wikidata (pour la lecture humaine, nous avons adapté les noms d'entités et de propriétés). Les requêtes originales peuvent être trouvées sur le dépôt GitHub.

5.2.1 Objectif de la tâche

Pour chaque requête, l'objectif est de trouver un contexte d'identité qui permettra d'étendre la requête à des entités similaires. De cette façon, les utilisateurs peuvent bénéficier de résultats plus complets. Le flux de travail est le suivant (voir Figure 3) : dans un premier temps, à partir de la requête, nous extrayons l'entité (ou les entités) instanciée(s) qui sera (seront) la (les) graine(s) (étape 1). Ensuite, pour chaque graine, nous calculons son treillis d'identité (étape 2) qui contiendra dans chacun de ses nœuds un ensemble de propriétés indiscernables et de propagation (cf. algorithmes 1 et 3). Troisièmement, avec la propriété (ou l'ensemble de propriétés) instanciée liée à la graine dans la requête, nous sélectionnons dans le treillis le nœud ayant cette propriété dans son ensemble de propagation (étape 4). Ce nœud sera considéré comme le contexte d'identité de la requête. En effet, si plusieurs contextes d'identité sont possibles, l'utilisateur doit choisir celui qui convient le mieux à l'objet de sa tâche. Enfin, sur la base du contexte d'identité sélectionné, nous pouvons obtenir des entités similaires (étape 5) et réécrire la requête avec la graine et les entités similaires (étape 6).

8. http://gaia.infor.uva.es/hdt/wikidata/wikidata2018_09_11.hdt.gz

Requête	**Listing 2**	**Listing 4**	**Listing 5**
Graine	Paracetamol	France	Les Républicains
Ψ	research intervention	head of	member of political party
Π	condition treated, interacts with, legal status	capital, head of	country, political ideology
Entités similaires	Ibuprofen Aspirin	July Monarchy, French 3rd Republic, …	UMP, RPR, …
# résultats sans notre approche	586	12	2
# résultats avec notre approche	**860**	**99** (77)	**13**

TABLE 1 – Contribution du contexte d'identité sur les trois requêtes.

5.2.2 Requêtes

La première requête dans le Listing 2 concerne le médicament "Paracétamol". L'objectif de la requête est de récupérer tous les essais cliniques de ce médicament. Une extension intéressante de cette requête pourrait être de trouver tous les essais de médicaments légaux similaires en termes de conditions médicales traitées et d'interactions physiques.

```
SELECT DISTINCT ?clinicalTrial WHERE {
  ?clinicalTrial :researchIntervention
      :Paracetamol . }
```

Listing 2 – Requête SPARQL récupérant toutes les études sur l'analgésique nommé Paracétamol.

Le tableau 1 montre ce que notre approche peut apporter comme résultats supplémentaires. Les deuxième, troisième et quatrième colonnes correspondent respectivement aux Listing 2, 4 et 5. Pour la première requête (et aussi pour les suivantes), il n'y a qu'une seule graine "Paracétamol" (respectivement "France" et "les Républicains") car c'est la seule entité instanciée dans la requête. Pour remplir ce tableau, nous avons d'abord calculé le treillis de la graine, puis, sélectionné un contexte contenant la propriété "research intervention" dans son Ψ. Nous avons choisi comme contexte des médicaments légaux ayant les mêmes conditions médicales et les mêmes interactions physiques (évidemment, tout autre contexte pourrait être choisi en fonction des besoins des utilisateurs). Enfin, la requête est étendue avec des entités similaires comme indiqué dans Listing 3. Les résultats montrent une augmentation de 47 % du nombre d'essais cliniques pour le contexte considéré.

```
SELECT DISTINCT ?clinicalTrial WHERE {
  VALUES (?drug) {
    (:Paracetamol) (:Ibuprofen)
      (:Aspirin) }
  ?clinicalTrial :researchIntervention
      ?drug . }
```

Listing 3 – Expansion de la requête SPARQL en récupérant toutes les études sur les entités similaires au paracétamol dans le context d'identité choisi.

La deuxième requête, dans Listing 4, vise à retrouver les personnes qui ont autrefois dirigé la France. Cependant, la France a une histoire complexe et a changé de régime politique à plusieurs reprises (par exemple, pendant la Seconde Guerre mondiale, ou pendant la période napoléonienne). Ainsi, même si le territoire français a été presque toujours le même au cours des siècles passés, chaque régime politique a sa propre entité dans Wikidata. Il se peut donc que la requête ne donne pas tous les résultats escomptés. Mais si l'utilisateur choisit le bon contexte d'identité, c'est-à-dire $\mathcal{C}_{(\{capital,officialLanguage\},\{headOf\},\approx)}$ alors toutes les personnes attendues seront récupérées.

```
SELECT DISTINCT ?headOfState WHERE {
  ?headOfState :headOf :France . }
```

Listing 4 – Requête SPARQL récupérant toutes les personnes qui ont été à la tête de l'État français moderne.

Comme pour la requête sur "Paracétamol", nous avons calculé le treillis et cherché le contexte avec *headOf* dans les propriétés de propagation. Les résultats sont indiqués dans la troisième colonne de Table 1. L'expansion de la requête est réalisée comme pour la précédente. Il est à noter que parmi les 99 résultats, 22 personnes n'étaient pas à la tête de la France. 14 étaient en réalité à la tête du conseil municipal de Paris et 8 étaient Grand Maître d'une obédience maçonnique en France. Cela est dû au fait que le conseil et l'obéissance sont mal placés dans l'ontologie de Wikidata. Ces erreurs ne peuvent donc pas être attribuées à notre approche. Les résultats montrent une augmentation de 542 % du nombre de dirigeants français pour le contexte considéré.
Enfin, dans Listing 5, nous présentons une requête sur les politiciens français du parti Les Républicains qui ont été condamnés. La particularité ici est que cet important parti politique a changé plusieurs fois de nom, soit à cause de scandales politiques, soit à cause de défaites humiliantes. Par conséquent, si le graphe de connaissances n'est pas à jour ou n'est pas complet, certaines personnes qui ont été membres de plusieurs versions de ce parti dans le monde réel pourraient ne pas être effectivement liées à chacune de ces versions dans le graphe de connaissances. C'est le cas de Wikidata qui renvoie, pour la requête de Listing 5, seulement deux politiciens. Cependant, il y a plus d'une douzaine de politiciens de ce parti qui ont été condamnés pour divers crimes. En utilisant notre approche, il est possible de sélectionner un contexte composé de l'alignement politique et du pays pour lequel la propriété *memberOf* se propage, et, par conséquent, d'obtenir un résultat plus complet (bien sûr en fonction de l'exhaustivité des données sur les hommes politiques dans Wikidata).

```
SELECT DISTINCT ?politician ?crime
WHERE { ?politician :memberOf
      :TheRepublicans ;
      :convictedOf ?crime . }
```

Listing 5 – La requête SPARQL récupère tous les politiciens membres du parti français Les Républicains qui ont été condamnés.

Les mêmes étapes que pour les requêtes concernant le "Paracétamol" et la "France" ont été reproduites. Les résultats sont présentés dans la quatrième colonne de Table 1. Les résultats montrent une augmentation de 550 % du nombre de politiciens condamnés pour le contexte considéré.

5.3 Discussion

Comme nous l'avons vu, notre approche permet de découvrir des propriétés de propagation pour un ensemble donné de propriétés indiscernables Π. Un contexte d'identité avec ses ensembles d'indiscernabilité et de propagation peut fournir des réponses plus complètes aux requêtes grâce à l'expansion des requêtes. Les résultats sont très prometteurs, mais il faut les confronter à d'autres types de graphes de connaissances et à des combinaisons de graphes de connaissances distincts. En outre, notre approche ne fonctionne pas bien lorsque la propriété d'une entité manque de propriété la décrivant en langage naturel (comme *rdfs :comment* ou *schema :description*). Il s'agit d'une limitation puisque de nombreuses ontologies ne fournissent pas de descriptions textuelles de leurs propriétés. Par conséquent, une première étape pour les travaux futurs consiste à contourner cette faille par une approche à multiples composantes. De plus, dans une description textuelle, certains mots peuvent ne pas être pertinents (comme un identifiant Wikidata) et dégrader la qualité des résultats.

6 Conclusion et travaux futurs

Dans ce papier, nous avons proposé une approche fondée sur le plongement de phrases pour découvrir les propriétés propageables pour un ensemble de propriétés indiscernables données. Notre approche calcule, pour une entité, un treillis d'identité qui représente tous les contextes d'identité possibles de l'entité, c'est-à-dire les ensembles d'indiscernabilité et leurs ensembles de propagation respectifs.
Quelques limitations de notre approche nécessitent des investigations supplémentaires. En effet, seules les propriétés ayant une description textuelle peuvent être traitées. Utiliser d'autres caractéristiques, par exemple la valeur des propriétés, le nombre d'utilisations des propriétés ou leurs caractéristiques sémantiques, est donc essentiel pour améliorer les résultats. Cependant, capturer les informations ontologiques d'une propriété lors d'un plongement reste un problème ouvert. De plus, utiliser seulement une technique de plongement de phrases combinée avec l'intuition de la première loi de Tobler est peut-être trop naïf dans certains

cas. Par conséquent, il est aussi nécessaire de remettre en question notre travail en combinant aussi des graphes de connaissances distincts. Pour l'instant, nous ne considérons dans le treillis que le cas où l'entité est le sujet d'un triplet, il nous faudrait donc essayer aussi de traiter les triplets dans l'autre sens. Pour finir, nous souhaitons proposer un prototype plus complet, automatisant au maximum ce qui peut l'être, pour permettre à l'utilisateur de sélectionner facilement le contexte lui permettant d'obtenir de meilleurs résultats de requête. Par exemple, l'expansion de la requête est réalisée manuellement après le calcul automatique du contexte d'identité. Il serait aussi intéressant d'utiliser RDF* et SPARQL* ([12]) pour représenter le contexte d'identité tel que défini dans ce papier.

Références

[1] Manel Achichi, Zohra Bellahsene, and Konstantin Todorov. A survey on web data linking. *Revue des Sciences et Technologies de l'Information-Série ISI : Ingénierie des Systèmes d'Information*, 2015.

[2] Wouter Beek, Stefan Schlobach, and Frank van Harmelen. A contextualised semantics for owl : sameas. In *ESWC*, 2016.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5 :135–146, 2017.

[4] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder. *CoRR*, abs/1803.11175, 2018.

[5] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *EMNLP*, pages 670–680. Association for Computational Linguistics, 2017.

[6] Li Ding, Joshua Shinavier, Tim Finin, and Deborah L McGuinness. owl : sameas and linked data : An empirical study. 2010.

[7] Nick Drummond and Rob Shearer. The open world assumption. In *eSI Workshop : The Closed World of Databases meets the Open World of the Semantic Web*, volume 15, 2006.

[8] Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. Binary rdf representation for publication and exchange (hdt). *Web Semantics : Science, Services and Agents on the World Wide Web*, 19 :22–41, 2013.

[9] Alfio Ferrara, Andriy Nikolov, and François Scharffe. Data linking for the semantic web. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 7(3) :46–76, 2011.

[10] Nicola Guarino and Christopher A. Welty. Evaluating ontological decisions with ontoclean. *Commun. ACM*, 45(2) :61–65, 2002.

[11] Harry Halpin, Patrick J Hayes, James P McCusker, Deborah L McGuinness, and Henry S Thompson. When owl : sameas isn't the same : An analysis of identity in linked data. In *International Semantic Web Conference*, pages 305–320. Springer, 2010.

[12] Olaf Hartig and Bryan Thompson. Foundations of an alternative approach to reification in rdf. *ArXiv*, abs/1406.3399, 2014.

[13] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible sroiq. *Kr*, 6 :57–67, 2006.

[14] Al Koudous Idrissou, Rinke Hoekstra, Frank van Harmelen, Ali Khalili, and Peter Van den Besselaar. Is my : sameas the same as your : sameas ? : Lenticular lenses for context-specific identity. In *K-CAP*, 2017.

[15] Miguel A. Martínez-Prieto, Mario Arias, and Javier D. Fernández. Exchange and consumption of huge rdf data. In *The Semantic Web : Research and Applications*, pages 437–452. Springer, 2012.

[16] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[17] Markus Nentwig, Michael Hartung, Axel-Cyrille Ngonga Ngomo, and Erhard Rahm. A survey of current link discovery frameworks. *Semantic Web*, 8(3) :419–436, 2017.

[18] Natalya Fridman Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale knowledge graphs : lessons and challenges. *Commun. ACM*, 62(8) :36–43, 2019.

[19] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove : Global vectors for word representation. In *EMNLP*, 2014.

[20] Joe Raad, Nathalie Pernelle, and Fatiha Saïs. Detection of contextual identity links in a knowledge base. In *K-CAP*, 2017.

[21] Petar Ristoski and Heiko Paulheim. Rdf2vec : Rdf graph embeddings for data mining. In *International Semantic Web Conference*, 2016.

[22] Amit Singhal. Modern information retrieval : A brief overview. *IEEE Data Eng. Bull.*, 24(4) :35–43, 2001.

[23] Sandeep Subramanian, Adam Trischler, Yoshua Bengio, and Christopher J. Pal. Learning general purpose distributed sentence representations via large scale multi-task learning. *CoRR*, abs/1804.00079, 2018.

[24] Waldo R Tobler. A computer movie simulating urban growth in the detroit region. *Economic geography*, 46(sup1) :234–240, 1970.