# Découverte d'expressions référentielles dans les graphes de connaissances

Armita Khajeh Nassiri *, Nathalie Pernelle *
Fatiha Saïs *

* LRI, Université Paris Saclay, CNRS UMR 8623, France
nom.prenom@lri.fr,

**Résumé.** Dans un graphe de connaissance, une expression référentielle est une formule logique qui permet d'identifier de façon unique une entité. De telles expressions peuvent être exploitées pour répondre à des requêtes, lier des données, annoter des ressources textuelles, ou encore anonymiser des données. Il peut potentiellement exister de nombreuses expressions logiques pour identifier de maniére unique une entité. Nous proposons une approche permettant de découvrir efficacement certaines expressions référentielles en nous concentrant sur celles qui ne peuvent être trouvées en instanciant des clés. Les premières expérimentations montrent que cette approche passe à l'échelle de jeux de données de plusieurs millions de triplets RDF et que ces expressions peuvent permettre de lier efficacement les instances de classes de différents jeux de données.

## 1 Introduction

Une *expression référentielle* (ER) est une description en langage naturel ou une formule logique permettant d'identifier de manière unique une entité. Par exemple, le 44ème président des Etats-Unis caractérise sans ambiguité Barack Obama. Les expressions référentielles trouvent des applications en désambiguisation, en anonymisation des données, en réponse à une requête ou encore en liage de données. Il peut potentiellement exister de nombreuses expressions référentielles pour une entité. La génération de telles expressions est une tâche bien étudiée en génération de langage naturel, (Dale, 1992). Différents algorithmes avec différents objectifs ont été proposés pour découvrir automatiquement les ER. Ces approches varient en fonction de l'expressivité des formules logiques qu'elles peuvent générer. Par exemple, dans (Dale, 1992; Krahmer et al., 2003), les ER créées sont des conjonctions d'atomes. (Ren et al., 2010) découvrent des ER plus complexes, représentées en logique de description et pouvant comporter des quantificateurs universels. Pour être efficace et réduire l'espace de recherche, les méthodes mettent l'accent sur la minimalité des descriptions qu'elles découvrent et/ou sur des préférences définis sur les propriétés utilisées (Galárraga et al., 2019). Cependant, la plupart de ces méthodes ne sont pas capables de s'adapter à de grands graphes de connaissances tels que YAGO ou DBpedia, graphes qui comportent des millions d'instances.
Dans ce travail, nous proposons un algorithme de découverte automatique de ER pour chacune des instance d'une classe d'un graphe de connaissances. Nous nous focalisons sur les ER qui ne peuvent pas résulter de l'instanciation d'une clé associée à la classe considérée. En effet, les

clés sont des ensembles de propriétés dont les valeurs peuvent identifier de manière unique une instance de cette classe. Par conséquent, si les propriétés d'une clé sont instanciées, celles-ci peuvent alors être considérées comme des expressions référencielles. Par exemple, si le numéro ISBN est une clef déclarée pour la classe $Livre$, quelle que soit la valeur d'ISBN utilisée pour instancier cette propriété, l'expression pourra être considérée comme une expression référentielle. Des travaux récents permettent de découvrir des cles efficacement dans des graphes de données (Symeonidou et al., 2014, 2017; Soru et al., 2015) et l'approche proposée se base sur les résultats pouvant être obtenus par de tels travaux pour réduire l'espace de recherche.

Cet article est organisé de la façon suivante. En section 2, nous présentons le problème de découverte de ER dans les graphes de connaissances. La section 3 décrit l'approche de découverte de ER que nous avons dévéveloppée. Enfin, en section 4, nous présentons les résultats obtenus sur 10 classes du graphe de connaissance YAGO.

## 2 Définition du problème de découverte de ER

Un **graphe de connaissance** $\mathcal{G}$ peut être défini par le couple $(\mathcal{O}, \mathcal{F})$ où l' ontologie $\mathcal{O} = (\mathcal{C}, \mathcal{DP}, \mathcal{OP}, \mathcal{A})$ est définie par un ensemble de classes $\mathcal{C}$, un ensemble $\mathcal{DP}$ de *owl*:*DataTypeProperty*, un ensemble $\mathcal{OP}$ de *owl*:*ObjectProperty*, et un ensemble d'axiomes $\mathcal{A}$. $\mathcal{F}$ représente une collection de triplets *(subject, property, object)* $\in (\mathcal{I} \cup \mathcal{B}) \times (\mathcal{OP} \cup \mathcal{DP}) \times (\mathcal{I} \cup \mathcal{C} \cup \mathcal{L} \cup \mathcal{B})$ [1] où $\mathcal{I}$ est un ensemble d'individus, $\mathcal{B}$ est un ensemble de noeuds blancs, et $\mathcal{L}$ est un ensemble de littéraux.

Dans ce travail, nous considérons des ER qui sont valides pour une classe $C \in \mathcal{C}$ et qui peuvent être définies de la façon suivante :

**Definition (Expression référentielle).** Une expression référentielle ($ER$) définie pour une instance $u$ de la classe $C$ peut être représentée par une formule du premier ordre de la forme :

$$C(x) \bigwedge_{p_i \in \mathcal{OP} \cup \mathcal{DP}} p_i(w, y)$$

telle que la formule, fermée existentiellement, est restreinte aux conjonctions d'atomes formant un graphe connecté enraciné en $x$, et les feuilles de ce graphe représentent soit un individu de $\mathcal{I}$ soit un littéral de $\mathcal{L}$, les autres noeuds représentant des variables. Cette formule est une ER valide pour $u$ dans un jeu de données si la formule est vraie quand $x$ est instancié par $u$ mais fausse pour tout autre instance de $C$ du jeu de données.

Par exemple, une ER décrite pour Mozart peut être un musicien né à Salzbourg et décédé à Vienne.

L'objectif est de découvrir des ER **minimales**. Une ER est minimale si elle n'implique pas une autre ER qui respecterait la définition précédente. Cependant, dans une tâche telle que le liage des données, les ER qui comportent des IRIs sont souvent peu pertinentes. En effet, les individus décrits dans différents graphes de connaissances sont rarement représentés par la même IRI. C'est pourquoi, nous considérons aussi les *ER étendues* qui ne sont pas minimales mais où les individus sont remplacés par une description construite à partir de propriétés clés valuées.

1. Nous ne considérons pas les noeuds blancs dans ce travail.

**Definition (Expression référentielle étendue ).** Une expression référentielle étendue est une expression pour laquelle les feuilles qui représentent des individus (IRI) sont remplacées par des variables et sont récursivement développées par un sous-graphe qui instancie un ensemble de propriétés clés. Les extensions qui conduisent à un motif de graphe dont les feuilles ne sont que des littéraux sont selectionnés, quand ils existent.

# 3 Génération des Expressions Référentielles

Nous proposons une approche qui détecte automatiquement les ER pour chaque entité d'une classe du graphe de connaissance, en se concentrant sur celles qui ne peuvent pas être trouvées en instanciant un ensemble de clés.
Dans un graphe de connaissance, une clef OWL2 peut être définie de la façon suivante :

**Definition (Clé).** Une clé, déclarée pour une classe $C$, est un ensemble de propriétés $\{p_1, \ldots, p_n\} \subseteq \mathcal{OP} \cup \mathcal{DP}$, qui expriment le fait que :

$$\forall x \forall y \forall z_1 ... z_n (C(x) \wedge C(y) \wedge \bigwedge_{i=1}^{n} (p_i(x, z_i) \wedge p_i(y, z_i)) \rightarrow \ x = y)$$

Par définition, chaque instanciation des propriétés d'une clé déclarée pour une classe $C$ identifiera de manière unique un individu présent dans $C$. Cette instanciation donnera potentiellement de nombreuses ER. Néanmoins, cela ne représente pas l'ensemble complet des ER minimales possibles qui peuvent être découvertes. Nous souhaitons donc enrichir cet ensemble d'ER en découvrant des ER qui ne comportent que des propriétés qui ne sont pas des clés. Pour cela, nous utiliserons les propriétés apparaissant dans les non-clés maximales de la classe $C$.
**Definition (Non-Clé Maximale).** Une non-clé maximale pour une classe $C$ dans un graphe de connaissance $\mathcal{G}$ est un ensemble de propriétés $P$ tel que $P$ n'est pas une clé, mais l'ajout de toute propriété à $P$ fait que cet ensemble devient une clé pour cette classe.

Nous décrivons la procédure d'extraction des ER dans l'algorithme 1. Pour générer des ER pour un graphe de connaissance $\mathcal{G}$ donné, nous créons d'abord les données pour la classe $C$ et en conservons un clone qui servira d'espace de recherche (lignes 1-2). Ensuite, en utilisant SAKey (Symeonidou et al., 2014), nous recherchons les ensembles de non clés maximaux NK (ligne 3). Nous créons ensuite les parties non vides de chaque ensemble de propriétés NK et les regroupons en fonction de leur cardinalité (ligne 5). Par exemple, si $NK = [\{p1, p2\}, \{p3, p4, p5\}]$ ; alors le niveau 1 correspond à $[\{p1\}, \{p2\}, \{p3\}, \{p4\}, \{p5\}]$, le niveau 2 comporte $[\{p1, p2\}, \{p3, p4\}, \{p3, p5\}, \{p4, p5\}]$, etc. Puisque l'algorithme recherche des ER minimales, il débute à partir du niveau 1. Il génère des motifs de sous-graphe à partir de l'espace de recherche et des propriétés instanciées $P$ (e.g. $p_1(x, v)$ appartenant au niveau 1 et $p_1(x, y) \wedge p_2(x, z)$ pour le niveau 2) (ligne 9). Nous stockons les ER valides (ligne 10) et réitérons jusqu'à ce que nous ayons couvert tous les ensembles de ce niveau. Avant de passer au niveau suivant, nous supprimons toutes les ER de ce niveau de l'espace de recherche ; en effet l'ajout de propriétés à une ER pour l'instance $i$, identifiera toujours $i$ de façon unique, mais rendra l'expression plus complexe et non minimale (ligne 13).
Pour détecter des ER plus complexes comportant un quantificateur existentiel, nous devons augmenter la profondeur du sous-graphe (lignes 11 -15). Pour chaque ER comportant des propriétés $P$ découverte précédemment au niveau $l$, un ou plusieurs noeuds qui sont des IRIs

peuvent être remplacés par un motif de sous-graphe enraciné par une variable quantifiée existenciellement. Soit une ER $p_1(x, y) \wedge p_2(x, z)$ trouvée au niveau 2 avec $P = \{p_1, p_2\}$. La fonction existentialER essaie de remplacer l'individu $y$ par une variable. L'algorithme obtient le co-domaine de la propriété $p_1$ à partir du schéma de $\mathcal{G}$ et récupère les données pour cette classe $\mathcal{C}'$. Puis, comme précédemment, il construit les ensembles de non clés maximaux pour $\mathcal{C}'$ et regroupe les propriétés $P'$ en fonction des différents niveaux. Les ER candidates sont créées en remplaçant les individus $u$ instanciant $p_1(x, u)$ dans les ER basées sur les propriétés $P$, en utilisant les non-clés de $P'$. Les ER candidates qui sont valides sont alors ajoutées aux ER résultats. L'algorithme passe à l'individu suivant $z$ et réitère le processus.

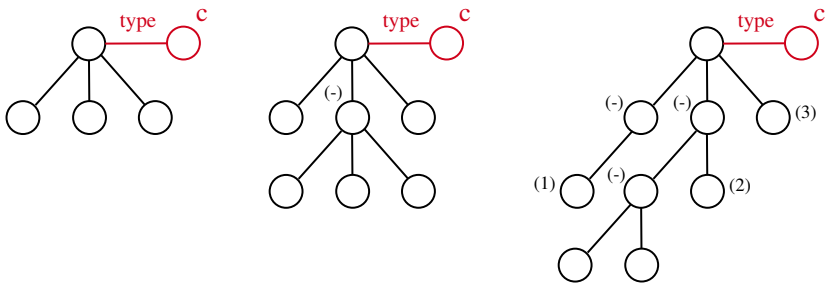
**Algorithme 1 :** ER-miner

**Input :** A knowledge graph $\mathcal{G}$, a class $\mathcal{C}$
**Output :** The set of all minimal ER for instances of type C

```
1  D ← createData (G, C)
2  SS ← clone (D) // sert d'espace de recherche
3  ERset ← ∅ // résultat de l'algorithme
4  NK ← generate maximal non-keys for the class C by running SAKey on the dataset D
5  A key/value pair (level, props) ← create the powerset of all nonkey sets in NK and
     group them based on their cardinality.
6  for level = 1 to level.max do
7      newER = ∅, newExistentialER = ∅
8      foreach P ∈ props.level do
9          ERcandidates ← constructSubgraphs (SS, P)
10         newER ← newER ∪ removeDuplicateSubgraphs (ERcandidates)
11         foreach ER ∈ newER do
12             if ER.child ∉ L then
13                 newExistentialER ← newExistentialER ∪ existentialER(ER, G)
14             end
15         end
16     end
17     ERset ← ERset ∪ newER ∪ newExistentialER
18     SS ← supress (SS, newER) // reduction de l'espace de
         recherche - monotonie
19 end
20 return ERset
```

De plus, nous avons développé une étape de post-traitement pour obtenir des expressions référentielles étendues telles que définies précédemment. Pour chaque IRI $u$ apparaissant dans les feuilles d'une $ER$ donnée, nous exploitons l'ensemble de clés minimales de la classe typant $u$ et étendons l'$ER$ en instanciant les propriétés de chaque clé minimale. Si les clés impliquent des object properties, cette étape est exécutée de manière récursive sur les IRI générées.

Certains élagages peuvent être appliqués pour limiter la taille et la complexité des ER qui peuvent être découvertes par notre approche. Tout d'abord, nous pouvons déclarer que pour la même variable, une propriété donnée ne peut être instanciée qu'une seule fois (i.e. une ER ne peut pas être construite en considérant un ensemble de valeurs pour des propriétés multivaluées). Par exemple, l'acteur George Clooney a joué dans de nombreux films et de




(a) $G_1$ (b) $G_2$ (c) $G_3$

FIG. 1 – *Exemples d'ER que l'algorithme ER-miner peut découvrir*

nombreuses ER minimales peuvent impliquer différents sous-ensembles de films dans lesquels il a joué. Nous considérons que ces ER ne sont pas significatives pour une tâche de liage de données. Enfin, la profondeur du motif de graphe et le nombre de variables autorisées peut être limité : nous nous concentrons sur les ER qui contiennent au plus un quantificateur existentiel. Ces ER peuvent être définies comme suit :

$$C(x) \wedge [p_i(x,y) \wedge p_j(y,v)] \bigwedge_{p_k \in \mathcal{OP} \cup \mathcal{DP}} p_k(\{x,y\},z)$$

où $x$, $y$ sont des variables, $v \in \mathcal{L}$, $z \in \mathcal{L} \cup \mathcal{I}$, et [ ] dénote l'optionalité de l'expression.
Nous présentons ci-dessous des exemples de ER choisies parmi les résultats des expérimentations menées avec les élagages décrits ci-dessus.

(1) Nous pouvons extraire des RE dont le motif de sous-graphe est de type de $G_1$, illustré sur la figure 1(a) : les propriétés appartiennent à une non-clé maximale de la classe $\mathcal{C}$ et les feuilles sont des individus ou des littéraux. Par exemple, Albert Einstein est un scientifique né à Ulm qui a travaillé à l'université de Zurich.
$RE_1 : scientist(x) \wedge wasbornin(x, ulm) \wedge worksat(x, university\_of\_zurich)$
(2) Nous pouvons obtenir des RE qui incluent un quantificateur existentiel comme le motif de sous-graphe $G_2$ en figure 1(b) où le symbole wildcard (_) est utilisé pour dénoter le quantificateur existentiel. Par exemple, glamour girl est un livre créé en 2008 qui a 320 pages et qui a été créé par quelqu'un (_) qui est marié à Brian Mcfadden.
$RE_2 : book(x) \wedge haspages(x, \text{“320”}) \wedge wascreatedonyear(x, \text{“2008”}) \wedge created\_inv(x,y) \wedge ismarriedto(y, brian\_mcfadden)$
(3) Nous pouvons obtenir des ER étendues où les individus sont remplacés par l'instanciation de leurs clés. Dans le motif de sous-graphe $G_3$ illustré en figure 1(c), la profondeur maximale est fixée à 3. Tous les chemins créés n'auront pas la longueur 3 car certains nœuds ($n_1$, $n_2$, $n_3$) peuvent atteindre un littéral en cours d'expansion.
Par exemple, Between wars est un film pour lequel Adrian Ford a écrit la musique. En développant Adrian Ford puis États-Unis par l'instanciation d'une clé de la classe personne et pays, nous obtenons l'ER étendue : Between wars est un film pour lequel une personne a écrit la musique. Cette personne est décédée le "1977-07-07" et a le prénom "Adrian". Elle est née le

"1904-01-01" et est citoyenne d'un pays, état voisin de Mexico et ayant un taux de chômage de "4,7%" et ayant participé à l'émeute de baltimore en 1861.
$RE_3 : film(x) \wedge wrotemusicfor\_inv(x, y) \wedge diedonyear(y, \text{“}1977-07-07\text{”})$
$\wedge hasgivenanme(y, \text{“}Adrian\text{”}) \wedge wasbornondate(y, \text{“}1904-01-01\text{”}) \wedge iscitizenof(y, w)$
$\wedge hasneighbor(w, Mexico) \wedge hasunemployment(w, \text{“}4.7\%\text{”}) \wedge participated(w, baltimore\_riot)$

# 4 Experimental Evaluation

Nous avons expérimenté notre approche sur un ensemble de classes du graphe de connaissances YAGO. Il s'agit du jeu de données utilisé dans VICKEY (Symeonidou et al., 2017) et constitué des descriptions RDF des instances de 10 classes. Les propriétés utilisées dans YAGO ont été alignées semi-automatiquement avec les propriétés utilisées pour les classes correspondantes dans DBpedia et les descriptions ont été réécrites en utilisant le vocabulaire de DBpedia. Un gold-standard recensant les liens d'identité entre instances de classes des deux graphes est également disponible. Notre méthode est implémentée en Python 3.6.5. Toutes les expérimentations ont été exécutées sur une seule machine Mac OS X 10.13 ayant un processeur de 2,7 GHz, 8 cœurs et 16 Go de RAM.
Dans le tableau 1 nous présentons les statistiques sur le jeu de données ainsi que les résultats obtenus pour les ER de profondeur un, sans quantificateur existentiel ni expansion. Pour chaque classe, nous avons indiqué le nombre d'ER découvertes, le temps d'exécution ainsi que le nombre maximal et moyen d'atomes dans les ER (i.e. nombre de nœuds dans le sous-graphe à l'exclusion de la racine). En moyenne, notre approche détecte de 1,5 à 31 ER par individu selon la classe. Le processus prend moins de 2 minutes par classe, sauf pour la plus grande, $Organisation$, qui nécessite plus de 3 heures.

TAB. 1 – *Détection des ER à profondeur 1 : classes, nombre de non-clé, nombre de ER, temps d'exécution, et taille des ER*

| Classe | Triplets | Inst. | #Prop | #NK | ER à prof. 1 | | max#atm | moy#atm |
|---|---|---|---|---|---|---|---|---|
| | | | | | #ER | Temps | | |
| Actor | 514.7k | 108.4k | 16 | 69 | 725.6k | 95.1s | 3 | 1.74 |
| Album | 381.1k | 137.1k | 5 | 2 | 212.1k | 14.7s | 3 | 1.30 |
| Book | 92.5k | 41.8k | 7 | 6 | 66.3k | 3.5s | 3 | 1.27 |
| Film | 533.5k | 123.9k | 9 | 7 | 690.9k | 102.3s | 4 | 1.77 |
| Mount. | 116.7k | 32.9k | 6 | 4 | 59.2k | 1.4s | 3 | 1.28 |
| Museum | 81.6k | 21.1k | 7 | 5 | 53.5k | 2.6s | 3 | 1.23 |
| Organiz. | 2.2M | 430.3k | 17 | 43 | 68.3M | 3.48h | 4 | 2.05 |
| Scientist | 335.6k | 93.1k | 18 | 92 | 309.9k | 64.0s | 4 | 1.58 |
| Univ. | 131.8k | 23.3k | 9 | 9 | 161.8k | 17.7s | 3 | 1.62 |
| City | 1.1M | 83.5k | 17 | 29 | 1.2M | 109.7s | 3 | 1.23 |

Dans le tableau 2, nous présentons les résultats obtenus quand nous considérons des ER avec un quantificateur existentiel pour 4 classes. Il faut noter que pour augmenter la profondeur du sous-graphe, nous n'avons utilisé que les instances directement typées par le co-domaine de la propriété. Ainsi, considérons l'ER $p_1(x, y) \wedge p_2(x, z)$ où l'individu $y$ peut être remplacé par

une description comportant le quantificateur existentiel. Nous requêtons le graphe pour obtenir les instances du co-domaine de la propriété $p_1$ (par exemple, Film pour la propriété actedIn). Comme le graphe de connaissance n'est pas saturé, il existe des instances typées par une sous-classe de film (e.g. romance _film) qui ne seront pas considérées et cela limite le nombre d'ER détectées. Cela explique pourquoi, pour la classe $Mountain$, si peu d'ER ont été trouvées.

TAB. 2 – *Détection d'ER comportant un quantifieur existenciel (sur 4 Classes) : nombre de ER détectées, temps, nombre maximum et moyen d'atomes.*

| Classe | Triplets | Inst. | #Prop | ER à prof. 2 | | max#atm | moy#atm |
|---|---|---|---|---|---|---|---|
| | | | | #ER | Temps | | |
| Album | 381.1k | 137.1k | 5 | 1.28M | 1.1h | 6 | 2.97 |
| Mount. | 116.7k | 32.9k | 6 | 77 | 2.78s | 2 | 2.0 |
| Museum | 81.6k | 21.1k | 7 | 1.7k | 17.3s | 3 | 2.23 |
| Scientist | 335.6k | 93.1k | 18 | 13.7M | 3.47h | 6 | 3.07 |

Nos expériences ont également montré que lorsque nous étendons des ER avec des clés (i.e. ER étendues), de nombreuses ER sont créées, mais très peu d'entre elles atteignent un état tel que les feuilles ne comportent que des littéraux.

Comme mentionné, les ER peuvent permettre de lier des entités décrites dans deux graphes de connaissances. Nous avons supposé que si une description identifie de manière unique une entité dans un GC, il est probable que la même description identifie la même entité dans l'autre GC. Aussi, pour chaque ER associée à une entité dans YAGO, si la description peut être associée à une seule entité dans DBPedia, nous lions les deux entités. Le tableau 3 montre le rappel, la précision et la F-mesure (F1) obtenus pour la tâche de liage en utilisant les clés, les clés et clés conditionnelles (Symeonidou et al. (2017)), et les ER extraites. Ces résultats montrent que l'exploitation des ER augmente considérablement le rappel quand ils sont comparés à ceux obtenus par les clés.

TAB. 3 – *Résultats du liage utilisant des clés (K), Clés Conditionnelles et clés (K+CK), et ER.*

| Classe | Rappel | | | Précision | | | F1 | | |
|---|---|---|---|---|---|---|---|---|---|
| | K | K+CK | ER | K | K+CK | ER | K | K+CK | ER |
| Actor | 0.27 | 0.60 | **0.66** | 0.99 | 0.99 | 0.99 | 0.43 | 0.75 | **0.79** |
| Album | 0.00 | 0.15 | **0.64** | 1.00 | 0.99 | 0.98 | 0.00 | 0.26 | **0.77** |
| Book | 0.03 | 0.13 | **0.77** | 1.00 | 0.99 | 0.97 | 0.06 | 0.23 | **0.86** |
| Film | 0.04 | 0.39 | **0.73** | 0.99 | 0.98 | 0.94 | 0.08 | 0.55 | **0.82** |
| Mountain | 0.00 | 0.29 | **0.77** | 1.00 | 0.99 | 0.98 | 0.00 | 0.45 | **0.86** |
| University | 0.09 | 0.25 | **0.65** | 0.99 | 0.99 | 0.98 | 0.16 | 0.40 | **0.78** |

# 5 Conclusion

Dans cet article, nous avons proposé une approche qui permet de découvrir efficacement un ensemble d'expressions référentielles (ER) pour chaque individu appartenant à une classe

de l'ontologie. Cette approche permet de se limiter à l'espace de recherche formé par les ensembles de propriétés apparaissant dans des non-clés maximales. Nous avons mené un ensemble d'expérimentations sur des données réelles issues de Yago et de DBPedia. Nos expérimentations montrent que nous pouvons découvrir des ER sur des classes comportant des millions de triplets et que celles-ci peuvent considérablement augmenter le rappel d'approches de liage basées sur des clés.

## Références

Dale, R. (1992). *Generating referring expressions - constructing descriptions in a domain of objects and processes*. ACL-MIT press series in natural language processing. MIT Press.

Galárraga, L., J. Delaunay, et J. Dessalles (2019). REMI : mining intuitive referring expressions on knowledge bases. *CoRR abs/1911.01157*.

Krahmer, E., S. v. Erk, et A. Verleg (2003). Graph-based generation of referring expressions. *Computational Linguistics 29*(1), 53–72.
English

Ren, Y., K. van Deemter, et J. Pan (2010). Generating referring expressions with owl2. In *Proceedings of the 23rd International Workshop on Description Logics (DL 201)*, pp. 428–438. CEUR-WS.

Soru, T., E. Marx, et A.-C. Ngonga Ngomo (2015). Rocker : A refinement operator for key discovery. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, Republic and Canton of Geneva, Switzerland, pp. 1025–1033. International World Wide Web Conferences Steering Committee.

Symeonidou, D., V. Armant, N. Pernelle, et F. Saïs (2014). SAKey : Scalable Almost Key Discovery in RDF Data. In S. Verlag (Ed.), *In proceedings of the 13th International Semantic Web Conference, ISWC 2014*, Volume Lecture Notes in Computer Science, Riva del Garda, Italy, pp. 33–49.

Symeonidou, D., L. Galàrraga, N. Pernelle, F. Saïs, et F. Suchanek (2017). VICKEY : Mining Conditional Keys on Knowledge Bases. In *International Semantic Web Conference (ISWC)*, Volume 10587 of *Lecture Notes in Computer Science*, Austria, pp. 661–677. Springer.

## Summary

In a knowledge graph, a referring expression is logical formula that can uniquely identify an entity. Referring expressions have applications in the fields of disambiguation, data anonymization, query response or data binding. There can potentially be many logical expressions to uniquely identify an entity. In this work, we discover referring expressions by focusing on those that cannot be found by instantiating the keys. The results of the first experiments showed that we can scale up to datasets of several millions of RDF triples and that the discovered ER can be used to discover identity links with a high recall and precision.