Concepts de voisins dans les graphes RDF : Une extension Jena et une interface graphique

Nicolas Fouqué, Sébastien Ferré, Peggy Cellier

Concepts de voisins dans les graphes RDF : Une extension Jena et une interface graphique

Nicolas Fouqué*, Sébastien Ferré*, Peggy Cellier*

*Univ Rennes, CNRS, INSA, IRISA
Campus de Beaulieu, 35042 Rennes cedex, France
Prenom.Nom@irisa.fr

Résumé. Les concepts de voisins définissent une forme symbolique de similarité entre les entités d'un graphe de connaissances. Partant d'une entité, chaque concept de voisins est un cluster d'entités voisines partageant un même motif de graphe centré sur l'entité. Dans ce papier démo, nous rappelons les définitions des concepts de voisins et nous présentons une extension de la librairie Jena dont l'API permet de calculer les concepts de voisins pour un modèle RDF(S) Jena. Nous présentons également une interface graphique permettant à un utilisateur d'effectuer ces calculs de façon simple et interactive.

1 Introduction

La méthode des k plus proches voisins (k-NN) (Mitchell, 1997) s'appuie sur la notion de *similarité* entre instances pour réaliser différentes tâches d'inférences, telles que la classification ou le raisonnement à partir de cas (De Mantaras et al., 2005). Par exemple, la classe d'une nouvelle instance x peut être décidée par un vote majoritaire parmi les k *plus proches voisins* de x, c'est-à-dire les k instances les plus similaires à x. Notre préoccupation est de pouvoir définir et calculer la similarité entre les nœuds d'un graphe de connaissances, typiquement un graphe RDF (Hitzler et al., 2009) mais pas seulement. Par similarité entre nœuds on entend ici la similarité des descriptions de ces nœuds, où la description d'un nœud X est constituée des relations arrivant et partant de X et récursivement des descriptions des nœuds adjacents. Par exemple, la similitude entre les pays France et Italie est que ce sont des républiques européennes parlant une langue latine et ayant les Alpes comme massif (entre autres choses).

Bisson (2000) distingue deux types de similarités : numériques et symboliques. Les similarités *numériques* ont l'avantage d'être faciles d'emploi mais elles n'offrent pas d'explications (*Pourquoi cette instance est plus similaire que telle autre ?*) et peuvent masquer des différences (*Deux instances ont la même similarité mais pour des raisons très différentes.*). De plus, peu de ces mesures ont été définies pour des données relationnelles (ex., RIBL (Horváth et al., 2001)). Les similarités *symboliques* évitent ces inconvénients en produisant des représentations symboliques généralisant plusieurs instances. Néanmoins, à notre connaissances, ces similarités ont seulement été utilisées pour la formation de concepts et de règles par généralisation (ex., Programmation Logique Inductive, PLI (Muggleton, 1995)), pas pour la recherche de plus proches voisins. De plus, les travaux existants considèrent généralement comme instances des

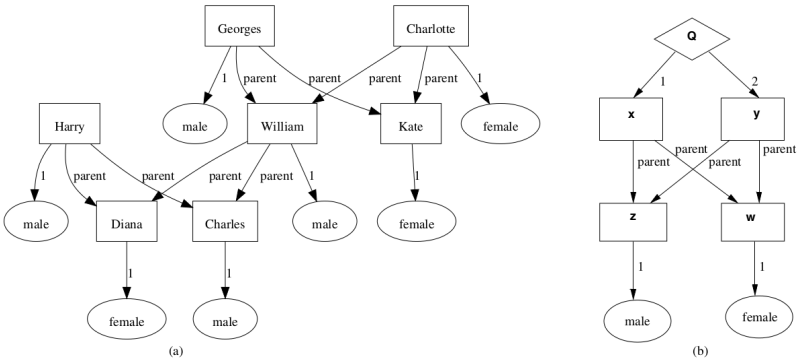

FIG. 1 – *(a) Graphe de connaissances sur la famille royale. Les rectangles sont les entités, les étiquettes d'ovales sont les relations unaires et les étiquettes d'arcs sont les relations binaires. (b) Requête définissant la relation frère-et-sœur.*

petits graphes (ex. molécules (Kuznetsov, 2013)) plutôt que les nœuds d'un grand graphe. Un inconvénient des similarités symboliques est leur coût de calcul.

Dans un travail antérieur (Ferré, 2017), nous avons introduit les *concepts de plus proches voisins* (*Concepts of Nearest Neighbours*, CNN) comme forme symbolique de similarité entre les nœuds d'un graphe de connaissances. L'approche s'enracine dans la théorie Graph-FCA (Ferré et Cellier, 2019), une généralisation de l'analyse de concepts formels (Ganter et Wille, 1999) aux graphes de connaissances ; et a été appliquée avec succès à l'évaluation approchée de requêtes conjonctives (Ferré, 2018) et à la prédiction de liens (Ferré, 2019).

La première contribution de ce papier démo est de présenter une ré-implémentation de nos algorithmes de calcul des CNN sous la forme d'une extension Jena [1]. En effet, Jena est le cadriciel le plus populaire du web sémantique. L'objectif est de permettre aux applications basées sur Jena de pouvoir facilement intégrer les CNN dans leurs raisonnements sur les données. La deuxième contribution est une interface graphique qui permet de facilement explorer les CNN à partir d'un fichier de données RDF. Le dépôt Git [2] contient les sources, la javadoc et deux scripts, un pour lancer l'interface graphique et l'autre pour une interface en ligne de commande. Une vidéo de démonstration est accessible en ligne [3].

La section 2 donne les définitions des concepts de voisins. La section 3 présente l'API de notre extension Jena. La section 4 présente l'interface graphique du point de vue utilisateur.

2 Concepts de voisins (CNN)

Dans cette section, nous rappelons brièvement les définitions relatives aux CNN, ainsi que les aspects algorithmiques et pratiques du calcul de leur approximation dans un temps donné. Plus de détails sont disponibles dans les publications précédentes (Ferré et Cellier, 2019; Ferré, 2017, 2018). Ces définitions sont illustrées par un petit exemple sur la famille royale anglaise.

1. `https://jena.apache.org/`
2. `https://bitbucket.org/sebferre/conceptsofneighbours/`
3. `https://youtu.be/6XsiSyU-l9s` (meilleure résolution que la vidéo soumise sur EasyChair)

2.1 Graphes de connaissances, requêtes et concepts de graphe

Un *graphe de connaissances* (GC) est une structure $K = \langle E, R, T \rangle$, où E est l'ensemble des *entités*, $R = \bigcup_{k \geq 1} R_k$ est l'ensemble des *relations* d'arité $k \geq 1$ et $T \subseteq \bigcup_{k \geq 1} R_k \times E^k$ est l'ensemble des *faits* considérés comme vrais. La figure 1 est une représentation graphique d'un petit GC sur la famille royale brittanique. Des exemples de faits sont $male(William)$ et $parent(William, Diana)$, où $male$ est une relation unaire ($k = 1$) et $parent$ est une relation binaire ($k = 2$). Les graphes RDF sont des GC où les entités sont appelées *ressources* (des URIs, des littéraux ou des *blank nodes*), les relations unaires sont appelées *classes*, les relations binaires sont appelées *propriétés* et les faits sont appelés *triplets*.

Un *motif de fait* $r(x_1, \ldots, x_k)$ a la forme d'un fait avec des variables à la place des entités. Un *filtre* exprime une condition booléenne sur des variables. On ne considère ici que les égalités entre une variable et une entité : $x = e$. On appelle, *éléments de requêtes*, l'ensemble des motifs de faits et des filtres qui peuvent être composés à partir d'entités, relations et variables. Un *motif de graphe* P est un ensemble d'éléments de requêtes. Une *requête* $Q = (x_1, ..., x_n) \leftarrow P$ est la projection d'un motif de graphe sur un sous-ensemble de ses variables. Une requête est dite d'arité n si elle est projetée sur n variables. Les requêtes, basées sur des motifs de graphe, jouent un rôle central dans notre approche car elles sont utilisées pour caractériser les CNN. De telles requêtes ont une forme concrète en SPARQL avec la syntaxe `SELECT ?x1...?xn WHERE { graph pattern }`. Les requêtes peuvent être vues comme des règles anonymes, c-à-d. des règles similaires à celles d'AMIE (Galárraga et al., 2015) mais sans la relation dans la tête de la règle. Ces requêtes sont également appelées PGP (*Projected Graph Pattern*) en Graph-FCA. Par exemple, la requête $(x, y) \leftarrow parent(x, z), parent(y, z), male(z), parent(x, w), parent(y, w), female(w)$ sélectionne les couples de personnes ayant le même père et la même mère, c-à-d. des frères et sœurs. La figure 1.(b) en donne une représentation graphique.

La *description* d'une entité $e \in E$ est la requête $Q(e) = e \leftarrow T$ où T est l'ensemble des faits du GC dans lequel chaque entité est considérée comme une variable. Ainsi, dans l'exemple de la famille royale, chaque individu est décrit par le graphe entier, mais chacun d'un point de vue différent. Par exemple, William est un homme qui a un garçon (Georges) et une fille (Charlotte) qui ont tous deux une même mère (Kate), et qui a un père (Charles) et une mère (Diana) qui ont un fils en commun (Harry).

Nous définissons maintenant les *réponses* à une requête. Un *matching* d'un motif P sur le GC $K = \langle E, R, T \rangle$ est une fonction μ des variables de P vers des entités de E telle que $\mu(t) \in T$ pour tout motif $t \in P$ et $\mu(f)$ s'évalue à vrai pour tout filtre $f \in P$, où $\mu(t)$ et $\mu(f)$ sont obtenus à partir de t et f en remplaçant chaque variable x par $\mu(x)$. L'ensemble des *réponses* $ans(Q, K)$ d'une requête $Q = (x_1, ..., x_n) \leftarrow P$ est l'ensemble des n-uplets $(\mu(x_1), ..., \mu(x_n))$ pour tout matching μ de P dans K. On notera que plusieurs matchings peuvent produire la même réponse et que les doublons sont ignorés. Dans la suite, nous considérons des requêtes avec une seule variable projetée, dont les réponses sont assimilables à un ensemble d'entités $A \subseteq E$.

Dans Graph-FCA (Ferré et Cellier, 2019), un *concept de graphe* est defini comme une paire $C = (A, Q)$ telle que $A = ans(Q)$ et $Q = msq(A)$; $msq(A)$ est la requête la plus spécifique qui vérifie $A = ans(Q)$. Cette requête la plus spécifique peut être calculée à partir de A avec le produit catégoriel de graphes (voir intersection de PGP dans (Ferré et Cellier, 2019)), ou avec l'anti-unification de Plotkin (1971). A est appelé l'*extension* du

concept et Q l'*intension* du concept. Un concept est dit d'arité n si son intension est une requête d'arité n. Un concept $C_1 = (A_1, Q_1)$ est plus spécifique qu'un concept $C_2 = (A_2, Q_2)$ si $A_1 \subseteq A_2$. Un résultat important est que l'ensemble des concepts de même arité forment un treillis, si bien que pour toute paire de concepts il existe un supremum $\vee$ (union des concepts) et un infimum $\wedge$ (intersection des concepts). Un exemple de concept est le couple $C_{ex} = (\{Charlotte, George, Harry, William\}, x \leftarrow parent(x, f), male(f), parent(x, m), female(m), parent(y, f), parent(y, m), male(y)\}))$. Il représente le concept des "enfants", lesquels, dans ce contexte, ont tous un père f et une mère m connus, lesquels ont ici toujours un fils y.

2.2 Distance conceptuelle et concepts de voisins

La *distance conceptuelle* entre deux entités $e, e' \in E$ est le plus petit concept (unaire) de graphe qui contient ces deux entités, c'est-à-dire le concept $\delta(e, e') = (A, Q)$ où l'intension $Q = msq(\{e, e'\})$ représente tout ce que e et e' ont en commun et où l'extension A englobe tous les entités qui partagent cette description commune. La distance conceptuelle vérifie les propriétés d'une distance si on prend l'ordre partiel $\leq$ sur les concepts et le supremum $\vee$ de concepts comme addition. Les "valeurs de distances" ont donc une représentation symbolique via l'intension de concept Q. Les distances conceptuelles sont organisées en ordre partiel plutôt qu'en ordre total, contrairement aux mesures de distance classiques. Une distance numérique $dist(e, e') = |ext(\delta(e, e'))|$ peut être dérivée de la taille de l'extension parce que plus e et e' sont proches, plus leur distance conceptuelle est spécifique et plus l'extension est petite. De façon duale, une similarité numérique $sim(e, e') = |int(\delta(e, e'))|$ peut être dérivée de la taille de l'intension (nombre d'éléments de requête) parce que plus e et e' sont similaires, plus leur distance conceptuelle est spécifique et plus l'intension est grande.

Les *concepts de voisins* $CNN(e, K)$ d'une entité choisie e dans le GC K sont les distances conceptuelles partant de e, c'est-à-dire

$$CNN(e, K) := \{\delta(e, e') \mid e' \in E\}.$$

L'extension propre d'un concept de voisins $\delta \in CNN(e, K)$ est l'ensemble des entités qui se trouvent exactement à la distance δ de e et est défini par $\delta.proper := \{e' \in E \mid \delta(e, e') = \delta\}$. L'ensemble des extensions propres constitue une partition de l'ensemble des entités du GC. Le concept de voisins à distance zéro $\delta(e, e)$ englobe, en plus de e, les éventuels objets e' dont la description contient celle de e. Comme il est inférieur à tous les autres concepts de voisins (positivité), on peut lui attribuer le rang 0, puis définir le rang de tout autre concept de voisins $\delta \in CNN(e, K)$ par $rank(\delta) := 1 + max\{rank(\delta') \mid \delta' \in CNN(e, K), \delta' < \delta\}$. On peut ensuite définir les k plus proches concepts de voisins comme les k premiers concepts de voisins dans l'ordre topologique induit par le rang.

Par exemple, le concept C_{ex} de la section précédente est la distance conceptuelle entre $Charlotte$ et $William$ (ils ont en commun d'avoir un père et une mère partageant un fils) et fait donc partie des concepts de voisins de chacun des deux individus. Pour $e = William$, l'extension propre est réduite à $\{Charlotte\}$ car $George$ et $Harry$ ont aussi en commun avec $William$ d'être des garçons. Pour $e = Charlotte$, l'extension propre est $\{Harry, William\}$ car $George$ a aussi en commun avec $Charlotte$ d'avoir des grands-parents dans le GC. Les voisins au rang 1 de $William$ sont : (1) $Kate$ (est parent d'un garçon et d'une fille), (2) $Charles$ (est un père), et (3) $Harry$ et $George$ (est un fils).

Discussion. Comme $CNN(e, K)$ induit une partition de l'ensemble des entités, le nombre de concepts de voisins ne peut qu'être inférieur au nombre d'entités, et en pratique il est très inférieur. C'est intéressant parce qu'en comparaison le nombre de concepts de graphe est exponentiel dans le nombre d'entités dans le pire cas. L'espace de recherche des approches à base de PLI est l'ensemble des règles, lequel est encore plus large que l'ensemble des concepts de graphe. Calculer les CNN pour une entité donnée est donc une tâche nettement plus abordable que la fouille de règles, bien que l'espace de représentations soit le même.

Comparé aux mesures numériques utilisées dans les approches de plus proches voisins, les CNN définissent un ordre plus subtil sur les entités. Tout d'abord, parce que les distances conceptuelles sont seulement partiellement ordonnées, il se peut que parmi deux entités aucune ne soit plus similaire à e que l'autre. Cela reflète le fait qu'il peut y avoir plusieurs façons d'être similaire à quelque chose, sans qu'une soit nécessairement préférée à l'autre. Par exemple, laquelle est la plus similaire à une "grande maison ancienne" ? une "petite maison ancienne" ou une "grande maison neuve" ? Ensuite, il se peut que deux entités soit exactement à la même distance et donc soit indiscernables en terme de similarité. Enfin, l'intension de concept fournit une explication intelligible de la similarité avec e.

2.3 Aspects algorithmiques et pratiques

Nous esquissons ici les aspects algorithmiques et pratiques du calcul de $CNN(e, K)$. Plus de détails sont disponibles dans (Ferré, 2018). Le principe de l'algorithme est de raffiner de façon itérative une partition de l'ensemble des entités en *clusters*, convergeant vers la partition induite par les extensions propres des concepts de voisins. Avant convergence, un cluster peut ainsi représenter l'union des extensions propres de plusieurs concepts de voisins.

Bien que notre algorithme termine, dans le cas de grandes descriptions ou de grands GC, le temps de calcul peut être trop long pour une utilisation pratique. Nous pouvons facilement contrôler ce temps d'exécution avec un *timeout* car l'algorithme est *any-time*. En effet, il peut produire à tout moment une partition des entités, avec une surestimation de la distance conceptuelle pour chaque cluster d'entités. Des expériences passées (Ferré, 2018) ont montré que l'algorithme a la bonne propriété de produire la moitié des concepts dans une petite fraction du temps total de calcul.

Des expériences dans (Ferré, 2018) sur des GC ayant jusqu'à un million de triplets ont montré que l'algorithme peut calculer tous les clusters pour des descriptions de centaines d'arcs en quelques secondes ou minutes. En comparaison, dans le même délai, les approches par relâchement de requêtes ne parviennent pas à faire plus de 3 relâchements, ce qui est largement insuffisant pour identifier des entités similaires dans la plupart des cas ; et les approches calculant les similarités symboliques avec chaque entité ne passent pas à l'échelle des GC qui ont des dizaines de milliers de nœuds.

3 Extension Jena

L'extension Jena permet de calculer les concepts de voisins à partir d'un modèle Jena. Normalement la seule classe avec laquelle l'utilisateur a besoin d'intéragir est la classe *Partition*. On construit un objet de cette classe à partir d'un modèle Jena (classe *Model*) représentant le

GC K, de l'URI de l'entité e dont on veut trouver les concepts de voisins et enfin de la profondeur maximale de description de l'entité. Le calcul des concepts de voisins $CNN(e, K)$ est déclenché par l'appel à la méthode *completePartitioning*. La méthode *cut* permet d'interrompre ce calcul et la méthode *oneStepPartitioning* permet d'exécuter le calcul pas à pas. À tout moment, la méthode *getClusters* retourne une liste des clusters (classe *Cluster*) formant la partition, de plus en plus fine au cours de l'exécution. La classe *Cluster* a de nombreuses méthodes permettant d'accéder à ses composantes, notamment les éléments de sa requête (méthode *getRelaxQueryElements*), les réponses à cette requête (méthode *getAnswers*) et la distance extensionelle (méthode *getExtensionDistance*). Un exemple d'utilisation de l'API est présenté dans la classe *ImplementationExample*.

Dépendances logicielles. L'application est développée en Java 12 donc elle nécessite un JDK compatible. Elle est prévue comme une extension de Jena 3.12 donc elle nécessite cette version de Jena. Les classes Jena utilisées sont :

- *Model* : représentation du graphe de connaissances ;
- *Element* : représentation des éléments de requêtes, augmentée dans notre classe *ElementUtils* ;
- *Table* : représentation des ensembles de réponses et opérations algébriques sur les relations, augmentée par notre classe *TableUtils*.

Pour l'instant le package contient encore la partie GUI donc elle nécessite la même version d'openjfx que l'interface graphique, c'est à dire JavaFX 12.0.1. L'application utilise le logging log4j, cette librairie est intégrée à Jena donc elle n'impose pas de téléchargements supplémentaires, cependant elle implique de configurer les loggers log4j.

Efficacité. À titre d'illustration, nous donnons quelques durées de partitionnement complet avec les quelques jeux de données RDF inclus dans le dépôt Git, et ce pour une profondeur de description de 1. Dans un jeu de données aussi réduit que *royal* (22 triplets), toutes les recherches sont terminées en moins d'une seconde, même avec une profondeur de description maximale, le graphe est vite parcouru dans son intégralité. Dans le jeu de données *mondial-europe* (62533 triplets), la recherche de similarité d'une ville ou d'une province prend une durée de l'ordre de 10s. Pour les pays (les objets les plus complexes du jeu de données) le partitionnement exhaustif peut prendre de 20min à 1h, mais les partitions obtenues au bout de 2min sont souvent bien assez fines pour être utiles.

Limites. L'API actuelle ne permet pas de contrôler les stratégies de choix du prochain cluster à raffiner et de choix de l'élément de requête à utiliser pour effectuer ce raffinement. Dans l'implémentation courante, les clusters sont générés selon une recherche en largeur d'abord (utilisation d'une file) ; et les éléments de requêtes choisis sont en priorité ceux situés à la plus faible profondeur de description de l'entité choisie.

4 Interface graphique

L'interface graphique permet à un utilisateur d'explorer les concepts de voisins de son graphe de connaissances, sans connaissance de l'API ni vraiment de RDF. Le scénario d'utilisation typique est le suivant (voir la figure 2 et la vidéo associée `https://youtu.be/6XsiSyU-l9s`).

1. Charger un fichier RDF du GC (ex. `mondial-europe.n3`), en précisant le format (ex. N3). Une liste partielle des entités s'affiche à gauche.

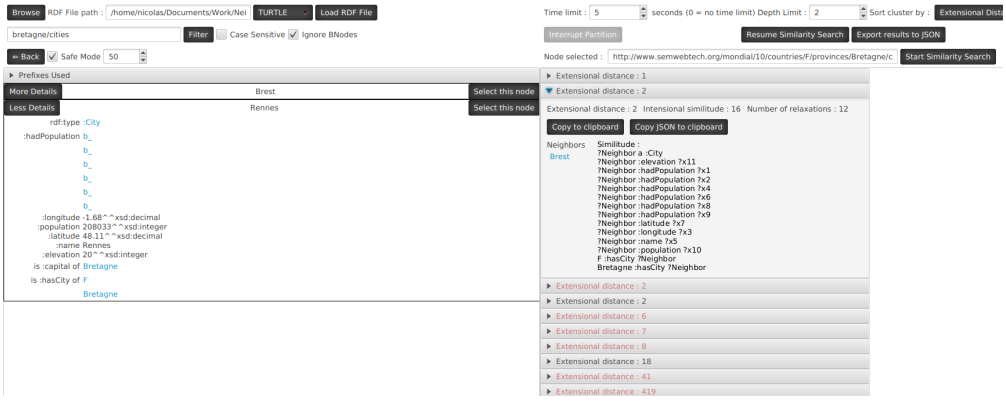

FIG. 2 – *Capture d'écran montrant la recherche de concepts de voisins de la ville de Rennes dans les données Mondial Europe.*

2. Entrer un mot-clé (ex. "bretagne/cities") dans la barre de recherche au-dessus de la liste d'entités et cliquer sur "Filter" pour filtrer la liste.
3. Sélectionner dans la liste filtrée l'entité (ex. `Rennes`) pour laquelle on veut calculer les concepts de voisins, en cliquant sur le bouton "Select this node".
4. Définir les paramètres de calcul des concepts de voisins dans la partie droite : limite de temps (ex. 5s), profondeur de description (ex. 2), critère de tri des clusters (ex. distance extensionnelle).
5. Cliquer sur "Start Similarity Search" pour démarrer le calcul.
6. Attendre que le partitionnement se termine ou l'interrompre à la main.
7. Consulter la liste des clusters à droite, triés par distance croissante. Pour chaque cluster, on peut afficher la requête, les réponses (c'est-à-dire les entités voisines), la distance extensionnelle, etc. Par exemple, le 2ème cluster contient la ville de Brest qui, comme Rennes, est une ville de Bretagne avec populations, coordonnées et altitude.
8. Exporter les résultats au format JSON pour une utilisation programmatique.

L'interface graphique offre des fonctionnalités en plus du scénario ci-dessus :

— Navigation dans le graphe via l'affichage des descriptions des entités dans la liste à gauche et suivi des liens vers les entités adjacentes. Cette navigation fonctionne également avec les blank nodes.
— Reprendre le partitionnement de façon globale ou pour un cluster en particulier.

En conclusion, nous avons proposé une extension pour la bibliothèque Jena permettant à toutes les applications utilisant Jena de calculer les concepts voisins dans un graphe de connaissances. De plus, nous avons développé une interface graphique facilitant l'exploration des concepts voisins à partir d'un fichier de données RDF.

Références

Bisson, G. (2000). La similarité : une notion symbolique/numérique. *Apprentissage symbolique-numérique 2*, 169–201.

De Mantaras, R. L., D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. L. Maher, M. T. Cox, K. Forbus, et al. (2005). Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review 20*(03), 215–240.

Ferré, S. (2017). Concepts de plus proches voisins dans des graphes de connaissances. In *Ingénierie des Connaissances (IC)*, pp. 163–174.

Ferré, S. (2018). Answers partitioning and lazy joins for efficient query relaxation and application to similarity search. In A. Gangemi et al. (Eds.), *Int. Conf. The Semantic Web (ESWC)*, LNCS 10843, pp. 209–224. Springer.

Ferré, S. (2019). Link prediction in knowledge graphs with concepts of nearest neighbours. In P. Hitzler et al. (Eds.), *The Semantic Web (ESWC)*, LNCS 11503, pp. 84–100. Springer.

Ferré, S. et P. Cellier (2019). Graph-FCA : An extension of formal concept analysis to knowledge graphs. *Discrete Applied Mathematics*. https ://doi.org/10.1016/j.dam.2019.03.003.

Galárraga, L., C. Teflioudi, K. Hose, et F. Suchanek (2015). Fast rule mining in ontological knowledge bases with AMIE+. *Int. J. Very Large Data Bases 24*(6), 707–730.

Ganter, B. et R. Wille (1999). *Formal Concept Analysis — Mathematical Foundations*. Springer.

Hitzler, P., M. Krötzsch, et S. Rudolph (2009). *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC.

Horváth, T., S. Wrobel, et U. Bohnebeck (2001). Relational instance-based learning with lists and terms. *Machine Learning 43*(1-2), 53–80.

Kuznetsov, S. (2013). Fitting pattern structures to knowledge discovery in big data. In P. Cellier, F. Distel, et B. Ganter (Eds.), *Int. Conf. Formal Concept Analysis*, LNAI 7880. Springer.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computation 13*.

Plotkin, G. (1971). *Automatic Methods of Inductive Inference*. Ph. D. thesis, Edinburgh Univ.

Summary

Concepts of neighbors define a symbolic similarity between the entities of a knowledge graph. Starting with an entity, each concept of neighbors is a cluster of neighbor entities that share a common graph pattern centered on the entity. In this demo paper, we recall the definitions of concepts of neighbors, and we present a Jena library extension whose API enables to compute concepts of neighbors for an RDF(S) Jena model. We also present a graphical interface that enables a user to perform those computations in a simple and interactive way.