Identifying Equivalent Relation Paths in Knowledge Graphs

Sameh K. Mohamed[1], Emir Muñoz[1,2], Vít Nováček[1], and Pierre-Yves Vandenbussche[2] ⋆ †

[1] Insight Centre for Data Analytics at NUI Galway, Ireland
[2] Fujitsu Ireland Limited

Abstract. Relation paths are sequences of relations with inverse that allow for complete exploration of knowledge graphs in a two-way unconstrained manner. They are powerful enough to encode complex relationships between entities and are crucial in several contexts, such as knowledge base verification, rule mining, and link prediction. However, fundamental forms of reasoning such as containment and equivalence of relation paths have hitherto been ignored. Intuitively, two relation paths are equivalent if they share the same extension, i.e., set of source and target entity pairs. In this paper, we study the problem of containment as a means to find equivalent relation paths and show that it is very expensive in practice to enumerate paths between entities. We characterize the complexity of containment and equivalence of relation paths and propose a domain-independent and unsupervised method to obtain approximate equivalences ranked by a tri-criteria ranking function. We evaluate our algorithm using test cases over real-world data and show that we are able to find semantically meaningful equivalences efficiently.

1 Introduction

Knowledge graphs (KGs) are graph-structured knowledge bases (KBs), consisting of facts encoded in the form of (subject, relation, object) triples, indicating that *subject* and *object* entities hold the relationship *relation*, e.g., (*Bob*, hasPartner, *Alice*). Popular KGs such as NELL, DBpedia, Freebase, or YAGO, have been developed in both academia and industry environments, attracting lot of attention due to their usefulness for many applications such as search, analytics, recommendations, and data integration. Relation paths with inverse are convenient means for complete exploration of KGs as they allow for unconstrained navigation that will not get stuck in sink nodes with no outgoing edges. However, there has been little study of their reasoning in comparison to single relations. In this paper we study two fundamental forms of reasoning on relation paths,

⋆ The first two authors contributed equally to this work.

† This work has been supported by the TOMOE project funded by Fujitsu Laboratories Ltd., Japan and Insight Centre for Data Analytics at National University of Ireland Galway, Ireland (supported by the Science Foundation Ireland grant 12/RC/2289).

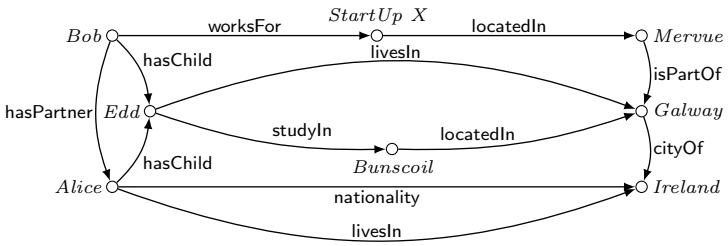

Fig. 1: Example knowledge graph of a family living in Ireland.

namely, containment and equivalence, and show how they can be used in the analysis of selected popular KGs. Intuitively, two relation paths P and Q are equivalent if they share the same extension, i.e., set of pairs (u, v), where nodes u and v are connected by P and Q. Recently, [23,20] have addressed the problem of determining whether two relations are equivalent or synonymous, using inductive approaches based on similarity and frequent itemsets, respectively. We generalize this task and compute equivalence between relation paths instead of single relations, which is a computationally more challenging problem. Relation paths in KGs have been mainly used in inference systems [11,12], and the link prediction task [15,14]. For instance, a relation path is able to encode complex relationships between entities, e.g., $Bob \xrightarrow{\mathsf{hasChild}} Edd \xrightarrow{\mathsf{hasChild}^{-1}} Alice$ in Fig. 1, can guide the inference that Bob is partner of Alice because they have a child in common, even if there was no direct relation between them. Also, $Bob \xrightarrow{\mathsf{livesIn}} Galway$ does not exists in Fig. 1, indicating possible missing information, which can be inferred by knowing that *Bob* works for a company located in *Mervue* which is part of *Galway* city, and he has a child living and studying in that city.

Reasoning on relation paths can be a computational challenge, especially when dealing with large KGs and considering inverse relations as well, and has hitherto been ignored. Given a fixed relation path, finding paths equivalent to it requires testing equivalence with all possible paths in the KG. Enumerating all simple paths between two entities is generally intractable, as their number can be in $\mathcal{O}(n!)$ for complete graphs (with n nodes) and tends to be quite large even for relatively sparse undirected graphs. We argue that despite of the challenges, a proper analysis of their reasoning is needed to unlock further applications in tasks such as knowledge graph completion, rule mining, and link prediction.

We propose to reduce the problem of finding equivalences to path containment. We map the problem to the well-studied regular path queries with inverse (2RPQs) in graph databases (see [3,4,2], among others). In graph databases, the notion of equivalence is defined in terms of containment, more specifically, query containment, which consists of determining whether the evaluation of a query Q_1 is a subset of the evaluation of a query Q_2. Checking containment of queries in databases is crucial in several contexts, such as query optimization, query reformulation, knowledge base verification, information integration, integrity checking, and cooperative answering [1]. Whether a relation paths and its reasoning in KGs can play a similar crucial role, and bring similar benefits is an open question. We hope that this work shed light on this topic.

More formally, the problem of interest in this paper is as follows:

Search of equivalent relation paths problem
Given: a knowledge graph $\mathcal{G}$, relation path query Q, integer k, depth d **Find:** top-k equivalent relation paths of max. length $2d$ for Q in $\mathcal{G}$ according to a ranking function $Rank_Q(P)$, for all $P \in \mathcal{C}$ set of candidates.

In this paper, we formulate an approximate solution to this problem, and illustrate it using four real-world knowledge graphs under different test cases from different domains of knowledge.

Example 1. Let us consider the KG in Fig. 1 with eight entities and nine different relationships. A query $Q = \langle \mathsf{livesIn}, \mathsf{cityOf}^{-1} \rangle$ asks for equivalent relation paths to *"a person living in a country which has a city"*. Valid answers will contain: (a) $\langle \mathsf{nationality}, \mathsf{cityOf}^{-1} \rangle$, (b) $\langle \mathsf{hasChild}, \mathsf{studyIn}, \mathsf{locatedIn} \rangle$, (c) $\langle \mathsf{hasChild}, \mathsf{livesIn} \rangle$, (d) $\langle \mathsf{hasPartner}^{-1}, \mathsf{hasChild}, \mathsf{livesIn} \rangle$. Interestingly, these results can be translated into insights about the person: (a) she has the *nationality* of the *country* where *city* is, (b) she has a *child* who study in a *school* located in *city*, (c) she has a *child* who lives in *city*, and (d) she has a *partner* whose child lives in *city*.

Organization. The rest of the paper is organized as follows. Section 2 presents background definitions for knowledge graphs, subgraphs, and connecting paths. In Section 3 we study the properties of containment and equivalence of relation paths. Section 4 presents our algorithm for efficient search for equivalent relation paths in a knowledge graph. Experimental results of our approach over four real-world knowledge graphs are presented in Section 5. We discuss related work in Section 6. Conclusions and future work are summarised in Section 7.

2 Preliminaries

In the following, we give definitions and examples supporting the two key notions of our approach – knowledge graph and query relation path.

Knowledge Graphs. We define a *knowledge graph* $\mathcal{G} = (V, E, \Sigma_V, \Sigma_E, \ell)$ as an edge-labeled directed multigraph, where V is the set of nodes, $E \subseteq V \times \Sigma_E \times V$ is the set of directed, labeled edges between two nodes, $\Sigma_V \subseteq \Sigma^*$ is a finite set denoted as the *node vocabulary*, $\Sigma_E \subseteq \Sigma^*$ is a finite set denoted as the *edge vocabulary*, and ℓ is a labeling function that assigns a label in Σ_V to a node in V, and a label in Σ_E to an edge in E. Each node $v \in V$ represents an entity, and each edge $e = (v, a, v') \in E$ represents a a-labeled relationship between entities v and v' (endpoints), where v is the domain of e, and v' is the range of e, denoted $Dom(e)$ and $Ran(e)$, respectively. Here, for simplicity, we omit the node and edge vocabularies, and use $\mathcal{G} = (V, E, \ell)$ to refer to a knowledge graph.

Example 2. Fig. 1 shows a fragment of a knowledge graph. Each entity (e.g., *Edd*) has a label name, and connects to other entities (e.g., *Galway*) via labeled relationships (e.g., $\mathsf{livesIn}$).

Paths. A *path* P in graph $\mathcal{G}$ is a sequence of edges $\langle e_1, e_2, \ldots, e_k \rangle$, where for $1 \leq i \leq k$, $e_i = (v_{i-1}, a_i, v_i)$ is an edge in $\mathcal{G}$ with label a_i and endpoints v_{i-1} and v_i. The integer k is called the *length* of P, denoted by $|P|$. We use $P(u \rightsquigarrow v)$ to denote a variable-length path with first node u and last node v, where u and v are known as the endpoints. We define the functions domain and range of a path P, $Dom(P) = Dom(e_1)$ and $Ran(P) = Ran(e_k)$. We define the binary associative operator $\oplus$ that concatenates two paths, $P_1 \oplus P_2$, by adding the edges in P_2 to the back of P_1 iff $Ran(P_1) = Dom(P_2)$. A *path label* $\ell(P)$ is defined as $\ell(v_0)\ell(e_1)\ldots\ell(v_{k-1})\ell(e_k)\ell(v_k)$, i.e., the concatenation of all node and edge labels on the path P. For instance, in our running example, we can have the paths $P_1 = \langle Edd, \mathsf{livesIn}, Galway \rangle$, and $P_2 = \langle Ireland, \mathsf{cityOf}^{-1}, Galway \rangle$. Notice that $\mathcal{G}$ is a directed graph, and we would like to walk through the graph in a *two-way unconstrained* manner, meaning that an edge can be walked in its opposite direction. This is to avoid the problem of getting stuck in sink nodes with no outgoing edges. This allows for complete exploration of the graph structure. To preserve the semantics of the original directed relations, we consider auxiliary *inverse* relations in the traversal, which is considered as a different relation, and equivalent to following a directed edge in its opposite direction. We use the '-1' superscript to denote the inverse relations.

Relation Paths. We introduce the use of the wildcard symbol '_' in paths to replace any node in the path. The use of a wildcard or "don't care" character in a path means that the position where the wildcard is used can take any value, i.e., any node label in Σ_V can be put in that place. For instance, we can have $P_1' = \langle \_, \mathsf{livesIn}, Galway \rangle$, which can serve to match all people (nodes) that live in Galway. We call a path *relation path* when all its nodes are replaced by wildcards, and wildcard symbols are omitted from the path.

Subgraph. Let $\mathsf{Graph}^d(\mathcal{G}, v)$ be the subgraph function that extracts $\mathcal{G}' = (V', E', \ell')$ around a node $v \in V$ from $\mathcal{G}$ up to depth d. The set of nodes $V' \subseteq V$ in $\mathcal{G}'$ contains all nodes that are reachable in $\mathcal{G}$ from v following paths of length $\leq d$. Similarly, the set of edges $E' \subseteq E$, where if $(v_1, a, v_2) \in E'$ then $v_1, v_2 \in V'$. Note that when $d = 0$, $\mathsf{Graph}^d(\mathcal{G}, v)$ extracts only the node v with no edges.

Intuitively, a subgraph allows us to generate a neighborhood for node v, which contains all reachable nodes from v following paths of a variable length.

Connecting Paths. If we relax the definition of path in such a way that the edges still carry a label, but the non-endpoint nodes do not (i.e., they are replaced by wildcards), then the path degenerates into a so-called *connecting path* anchored by its starting and ending nodes (endpoints). Formally, a node u is said to be *connected* to a node v in $\mathcal{G}$ via P if there is a $P(u \rightsquigarrow v)$ path.

Example 3. Considering the KG in Fig. 1, one can extract $\mathsf{Graph}^1(\mathcal{G}, Alice)$ and $\mathsf{Graph}^1(\mathcal{G}, Galway)$, which are the subgraphs of depth 1 around nodes *Alice* and *Galway*, respectively. Considering the overlapping nodes, i.e., *Edd*, and *Ireland*, we can identify the following connecting paths:

(a) $\langle Alice, \mathsf{hasChild}, \mathsf{livesIn}, Galway\rangle$ (c) $\langle Alice, \mathsf{nationality}, \mathsf{cityOf}^{-1}, Galway\rangle$
(b) $\langle Alice, \mathsf{livesIn}, \mathsf{cityOf}^{-1}, Galway\rangle$ (d) $\langle Alice, \mathsf{hasChild}, \mathsf{studyIn}, \mathsf{locatedIn}, Galway\rangle$.

Moreover, by replacing the endpoint nodes in connecting path (a) by wildcards, we get a relation path $P_k = \langle \mathsf{hasChild}, \mathsf{livesIn}\rangle$ of length 2.

3 Equivalence of Relation Paths

With the definitions of knowledge graph and relation paths, in this section, we present our study on the reasoning of relation paths.

Every relation path has an associated path extension in $\mathcal{G}$, that consists of pairs of entities: from node u we can reach node v following a relation path P if the pair (u, v) is a member of the *relation path extension* $\mathsf{PEXT}_{\mathcal{G}}(P)$. Therefore, the size of a relation path extension, correspond to the number of valid connecting paths we can generate for any pair of nodes in the graph. Formally, given $\mathcal{G} = (V, E, \ell)$ we define the relation path extension of a relation path P as:

$$\mathsf{PEXT}_{\mathcal{G}}(P) = \{(u, v) \mid u, v \in V \wedge \mathbf{I}_{\mathcal{G}}(P(u \rightsquigarrow v))\}, \tag{1}$$

where $\mathbf{I}_{\mathcal{G}}(P(u \rightsquigarrow v))$ is an indicator function that takes value True if there is a $P(u \rightsquigarrow v)$ path in $\mathcal{G}$, and False otherwise. Syntactically, relation paths can be seen as *2-way regular path queries* (2RPQs) [3], the default core navigational language for graph databases, where inverse relations are allowed. Then, the problem of deciding whether a pair of nodes is connected via a relation path P over a knowledge graph $\mathcal{G}$ (i.e., $P(u \rightsquigarrow v)$?) can be translated into computing the answer of 2RPQs, which is in low polynomial time [17,2]. The problem of determining whether a path P with inverse relations exists between nodes $u, v \in V$ can be easily reduced to the evaluation of *regular path queries* (RPQs) by extending the underlying KG with inverse edges (cf. [2]). Next, we define the notion of *symmetric closure* of a knowledge graph $\mathcal{G}$, denoted by $\mathcal{G}^{\pm}$. Let $\mathcal{G}^{\pm}$ be the knowledge graph obtained from $\mathcal{G} = (V, E, \ell)$ by adding the edge (u, a^{-1}, v), for each $(v, a, u) \in E$.

Proposition 1. *For every relation path P and knowledge graph $\mathcal{G}$ the problem of deciding whether a pair (u, v) of nodes belongs to $\mathsf{PEXT}_{\mathcal{G}}(P)$ can be solved in time $\mathcal{O}(|E| \cdot |P|)$.*

Proposition 2. *(See e.g., [6]). Let P be a relation path. There is a* NLogspace *procedure that computes $\mathsf{PEXT}_{\mathcal{G}}(P)$ for each knowledge graph $\mathcal{G}$.*

Proof (Sketch). Given a knowledge graph $\mathcal{G}$ and its symmetric closure $\mathcal{G}^{\pm}$. For each pair (u, v) of nodes in V, we check whether it belongs to $\mathsf{PEXT}_{\mathcal{G}}(P)$ over $\mathcal{G}$ as a simple evaluation of a regular path query over $\mathcal{G}^{\pm}$ [17]. Clearly, $\mathcal{G}^{\pm}$ can be constructed in Logspace from $\mathcal{G}$[1]. Non-emptiness of $P(u \rightsquigarrow v)$ in $\mathcal{G}^{\pm}$ can be checked in NLogspace in $|E|$ using a standard "on-the-fly" algorithm. We conclude that the whole process can be computed in NLogspace for each relation path P.

[1] Given a two-way automaton with n states, we can construct a one-way automaton with $\mathcal{O}(2^{n \log n})$ states accepting the same language [22].

With the relation path extension definition in place, two interesting and basic analysis tasks that arise are the containment and equivalence problem, defined in databases query languages (see, e.g.,[5] for details of a relational database context, and [3,21,10] for a graph database context). In a query scenario, if a relation path (i.e., 2RPQ) is equivalent to other with better computational properties (e.g., shorter, faster evaluation), then the initial relation path can be replaced for optimization purposes. In other words, we can search for a simpler relation path that is *contained* in the original one. Indeed, the containment and equivalence problems have been always very related and played a prominent role in the analysis of query languages in databases. In the following, we build upon these ideas, and use the containment problem as a building block for the equivalence of relation paths.

Definition 1 (Path Extension Containment). *Let P_1 and P_2 be two relation paths, and $\mathcal{G}$ a knowledge graph. We say that P_1 is* contained *in P_2, denoted $P_1 \sqsubseteq P_2$, if $\mathit{PEXT}_{\mathcal{G}}(P_1) \subseteq \mathit{PEXT}_{\mathcal{G}}(P_2)$.*

Theorem 1 ([4] Theorem 5). *The path extension containment problem is* PSPACE*-complete.*

We can reduce the path extension containment problem to the well-studied 2RPQ containment. A proof of Theorem 1 considering 2RPQs is given in Calvanese et al. [4], where is shown that for 2RPQs the problem of containment has a PSPACE-complete upper bound. The proof uses two-way automata techniques, where the containment problem is reduced to determine whether there is a path from start state to final state in a two-way automata.

Definition 2 (Path Extension Equivalence). *Let P_1 and P_2 be two relation paths. We say that P_1 and P_2 are* extension equivalent *(or* PEXT-equivalent*), denoted by $P_1 \equiv P_2$, iff $P_1 \sqsubseteq P_2$ and $P_2 \sqsubseteq P_1$. While when only $P_1 \sqsubseteq P_2$ (resp. $P_2 \sqsubseteq P_1$) is true, we say that P_1 and P_2 are* approximate equivalent *(or Δ-*equivalent*), denoted as $P_1 \preceq_\Delta P_2$ (resp. $P_2 \preceq_\Delta P_1$).*

It is easy to see, that path extension equivalences are in PSPACE-complete complexity given Theorem 1 and Definition 2.

Corollary 1. *The PEXT-equivalence problem is* PSPACE*-complete.*

Example 4. Using the KG in Figure 1, the following are two PEXT-equivalences: (a) $\langle \mathsf{livesIn} \rangle \equiv \langle \mathsf{studyIn}, \mathsf{locatedIn} \rangle$, and (b) $\langle \mathsf{hasPartner} \rangle \equiv \langle \mathsf{hasChild}, \mathsf{hasChild}^{-1} \rangle$, which could be interpreted as: (a) people who lives in *city* also study in *school* located in *city*, and (b) people that has *partner* also has a *child* who is children of *partner*. Whilst one can consider $\langle \mathsf{hasChild}, \mathsf{livesIn}, \mathsf{isPartOf}^{-1} \rangle \preceq_\Delta \langle \mathsf{worksFor}, \mathsf{locatedIn} \rangle$ to be a Δ-equivalence, because the PEXT of the r.h.s. path is a superset of the l.h.s. one. In other words, not everybody works for a company located in the same neighborhood where their children live.

Theorem 2. *PEXT-equivalence is an equivalence relation.*

Proof. We need to show that PEXT-equivalence is reflexive, symmetric, and transitive. This follows from the $\sqsubseteq$ relation (path extension containment) definition. Two paths P_1, P_2 are PEXT-equivalent iff $P_1 \sqsubseteq P_2 \wedge P_2 \sqsubseteq P_1$ where $\sqsubseteq$ is defined using a standard set inclusion operation $\subseteq$ on the path extensions. *(Reflexivity)* $P \equiv P$ for all P, because $\mathsf{PEXT}(P) \subseteq \mathsf{PEXT}(P)$. *(Symmetry)* If $P_1 \equiv P_2$, then $\mathsf{PEXT}(P_1) \subseteq \mathsf{PEXT}(P_2) \wedge \mathsf{PEXT}(P_2) \subseteq \mathsf{PEXT}(P_1)$, therefore also $P_2 \equiv P_1$ for all P_1, P_2. *(Transitivity)* If $P_1 \equiv P_2$ and $P_2 \equiv P_3$ then $\mathsf{PEXT}(P_1) \subseteq \mathsf{PEXT}(P_2) \wedge \mathsf{PEXT}(P_2) \subseteq \mathsf{PEXT}(P_1) \wedge \mathsf{PEXT}(P_2) \subseteq \mathsf{PEXT}(P_3) \wedge \mathsf{PEXT}(P_3) \subseteq \mathsf{PEXT}(P_2)$, which means that $\mathsf{PEXT}(P_1) = \mathsf{PEXT}(P_2) = \mathsf{PEXT}(P_3)$, i.e., $\mathsf{PEXT}(P_1) \subseteq \mathsf{PEXT}(P_3) \wedge \mathsf{PEXT}(P_3) \subseteq \mathsf{PEXT}(P_1)$, and therefore also $P_1 \equiv P_3$ for all P_1, P_2, and P_3. □

Theorem 3. *Δ-equivalence is a partial order.*

Proof. This comes as straightforward from Definition 2 – the $\subseteq$ relation used in the definition is a partial order (on path extensions) and therefore $\sqsubseteq$ is also a partial order (on relation paths).

4 Search of Δ-Equivalences

In the following, we mainly focus on the extraction of Δ-equivalences for two reasons: (a) extracting Δ-equivalences from large-scale KGs is cheaper than PEXT-equivalences, and (b) we argue that because of the incompleteness of knowledge graphs[2], PEXT-equivalences are harder to find in real-world KGs, than instances of its more relaxed version (Δ-equivalence), which we believe are more abundant. Point (a) is justified by our theoretical discussion in Section 3, while point (b) will be later justified for our experimental results in Section 5.

Algorithm 1 (SearchEquiv: Search of Δ-Equivalent Paths)

Input: Knowledge graph $\mathcal{G}$, query relation path Q, depth d
Output: Ranked list of Δ-equivalent paths of Q

1: $\mathcal{E} \leftarrow \mathsf{PEXT}_{\mathcal{G}}(Q)$
2: $\mathcal{C} \leftarrow set()$
3: **for** $(v_1, v_2) \in \mathcal{E}$ **do**
4: $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \mathsf{ConnectingPaths}(v_1, v_2, d, \mathcal{G})$
5: **return** $\mathsf{RankFunc}(\mathcal{C})$

We present Algorithm 1 as an implementation of our method for finding Δ-equivalences of a query path relying on connecting paths between pairs of entities in $\mathsf{PEXT}_{\mathcal{G}}(Q)$. All steps of our method are shown in Figure 2, along the data structures that each step takes as input an returns as output.

[2] Dong et al. (2014) [7] report that 71% of the people described in Freebase have unknown place of birth, 75% have unknown nationality, and the coverage for less used relations can be even lower.

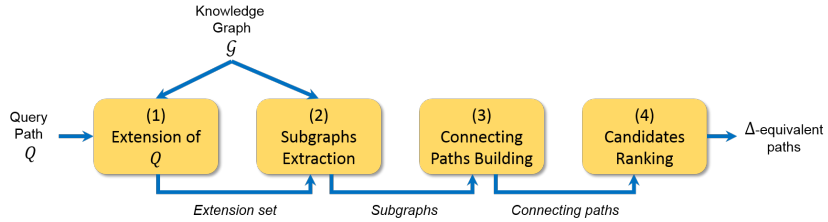

Fig. 2: Flow diagram of the process of retrieving Δ-equivalent paths for Q.

Next, we describe in details the four steps of our method:

(1) Extracting the extension of query path Q. This step corresponds to line 1 in Algorithm 1. In this step we extract the pairs of nodes in $\mathsf{PEXT}_{\mathcal{G}}(Q)$. Despite being a NLOGSPACE procedure, in practice this computation is very expensive for large-scale KGs, therefore, here we take a divide-and-conquer strategy to obtain this set. Let $Q = \langle \mathsf{e}_1,\mathsf{e}_2,\ldots,\mathsf{e}_k \rangle$ be the query relation path. First, we find the m-th relation with $m = \lfloor k/2 \rfloor$, and divide Q into two relation paths, $Q_1 = \langle \mathsf{e}_1,\mathsf{e}_2,\ldots,\mathsf{e}_m \rangle$ and $Q_2 = \langle \mathsf{e}_{m+1},\mathsf{e}_{m+2},\ldots,\mathsf{e}_k \rangle$. Second, we compute the set $\mathcal{Q} = \{u \mid Ran(\mathsf{e}_m) = u \wedge Dom(\mathsf{e}_{m+1}) = u\}$ of *witness nodes*, which are nodes that are range of Q_1 and domain of Q_2. For each node $u \in \mathcal{Q}$, we walk backwards through Q_1 and forwards through Q_2 in $\mathcal{G}$, saving all final endpoint nodes on both sides, i.e., all nodes which are $Dom(\mathsf{e}_1)$ and $Ran(\mathsf{e}_k)$. Considering that we followed Q_1 and Q_2 from the witness nodes, we know that the destination nodes which are $Dom(\mathsf{e}_1)$ and/or $Ran(\mathsf{e}_k)$ are indeed connected by Q. Thus, the $\mathsf{PEXT}_{\mathcal{G}}(Q)$ is obtained from generating pairs between the found endpoints.

It is worth to mention, that some nodes in $\mathcal{G}$ can be densely connected, and paths passing through them are prone to outnumber the true facts in $\mathcal{G}$, because they usually contain many-to-many relations. For example, the relation path $\langle \mathsf{hasGender},\mathsf{hasGender}^{-1} \rangle$ in most knowledge graphs will have more instances than the total count of fact triples, as it represents all possible combinations of instances of people with same gender in the graph. Because of that, we consider here a random sample of 1000 path instances, thus $\mathsf{PEXT}_{\mathcal{G}}(Q)$ is a sample rather than the full set. This constraint can be seen as a limitation of our method; however, it is a parameter that users can set accordingly, at a cost in the throughput. Further study and optimization of this part are left for future work.

(2) Subgraphs extraction. This step correspond to line 2 in Algorithm 2, where the subgraph generation routine is called twice. For each pair of nodes (u, v) in $\mathsf{PEXT}_{\mathcal{G}}(Q)$, we generate $\mathsf{Graph}^d(\mathcal{G}, u)$ and $\mathsf{Graph}^d(\mathcal{G}, v)$. A subgraph of depth d is generated using a mix between Depth-first search (DFS) and Breath-first search (BFS) from a node. It is easy to see that the subgraph generation step also suffers of scalability issues on large-scale KGs, where a simple DFS or BFS search can become very expensive, and return non-representative subgraphs if taken separately. Applying only DFS would lead to very deep subgraphs which might not consider all neighbor relations; and applying only BFS would lead to very wide subgraphs with not enough depth. To cope with this, we apply a

Algorithm 2 (ConnectingPaths: Connecting Paths Extraction)

Input: v_1, v_2 nodes, depth d, knowledge graph $\mathcal{G}$
Output: $\mathcal{C}$ list of connecting paths between v_1 and v_2
1: $\mathcal{G}_1, \mathcal{G}_2 \leftarrow \mathsf{Graph}^d(\mathcal{G}, v_1)$, $\mathsf{Graph}^d(\mathcal{G}, v_2)$
2: $T_1, T_2 \leftarrow \{v \mid u, v \in V_{\mathcal{G}_1} \wedge \mathbf{I}_{\mathcal{G}_1}(P(u \rightsquigarrow v))\}, \{w \mid y, w \in V_{\mathcal{G}_2} \wedge \mathbf{I}_{\mathcal{G}_2}(P(y \rightsquigarrow w))\}$
3: **for** $t \in T_1 \cap T_2$ **do**
4: **for** $P_1 \in \{P \mid u \in V_{\mathcal{G}_1} \wedge \mathbf{I}_{\mathcal{G}_1}(P(u \rightsquigarrow t))\}$ **do**
5: **for** $P_2 \in \{P \mid v \in V_{\mathcal{G}_2} \wedge \mathbf{I}_{\mathcal{G}_2}(P(v \rightsquigarrow t))\}$ **do**
6: $\mathcal{C}.append(P_1 \oplus \mathsf{Inverse}(P_2))$
7: **return** $\mathcal{C}$

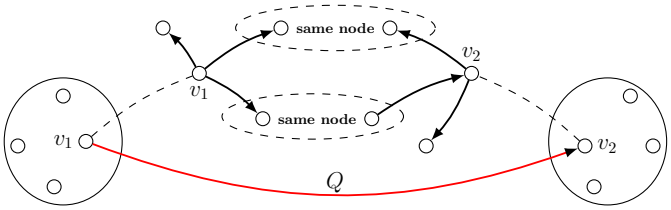

Fig. 3: Generation of connecting paths from the path extension of query Q.

DFS with BFS flavor by considering the following two restrictions: (1) from a given node, we extract a maximum of 50 instances of a same relation, to avoid neglecting under represented relations in nodes with highly common relations; and (2) in each iteration of DFS, we take a sample of 200 edges, to keep a manageable final size for a subgraph. Again, such decisions are implementation parameters that can be tuned by users. In this way, we try to keep instances for all neighbor relations (even the underrepresented ones, such as one-to-one relations that otherwise could be discarded), and we try to keep a representative enough subgraph while keeping an adequate size.

(3) Connecting paths building. This step is used in line 4 of Algorithm 1, and fully expanded in Algorithm 2. In this step we use subgraphs extracted in step (2) to build connecting paths, which are considered as Δ-equivalent to Q. For clarity, we depict the process in Fig. 3. For a given pair of nodes (v_1, v_2) in $\mathsf{PEXT}_{\mathcal{G}}(Q)$, we consider their corresponding subgraphs (line 1, Algorithm 2), and identify those nodes that are endpoints of paths in both subgraphs (lines 2-3, Algorithm 2). In other words, we find the so-called *target nodes* in which the subgraphs intersect. A target node is a connecting point which allows us to connect nodes from the domain and range of a query Q. Thus, we build connecting path around them as shown in Fig. 3. Notice that when creating a connecting path from a target node, we must append the inverse of the path from the right subgraph to the one on the left (line 6, Algorithm 2). This is because paths on the right subgraph originally considers v_2 as starting point, while now v_2 becomes the final point. The $\mathsf{Inverse}(\cdot)$ function in line 6 of Algorithm 2 inverts a path walking backwards: Let $P = \langle v_0, \mathsf{e}_1, \ldots, v_{k-1}, \mathsf{e}_k, v_k \rangle$, then $\mathsf{Inverse}(P) = \langle v_k, \mathsf{e}_k{}^{-1}, v_{k-1}, \ldots, \mathsf{e}_1{}^{-1}, v_0 \rangle$, and $(\mathsf{e}_i{}^{-1})^{-1} = \mathsf{e}_i$. The output of this step is a set of connecting paths which are ranked in the next step.

(4) Candidate Path Ranking. This is the last step and line 5 of Algorithm 1. After extracting connecting paths between nodes of the $\mathsf{PEXT}_{\mathcal{G}}(Q)$, we can design a ranking function in order to get the most relevant Δ-equivalences. We argue that such a ranking function should consider three criteria to rank candidates $P \in \mathcal{C}$: (Cr-1) if $P \preceq_{\Delta} Q$, then the size of the $\mathsf{PEXT}_{\mathcal{G}}(P)$ set could be big but always smaller than the size of $\mathsf{PEXT}_{\mathcal{G}}(Q)$; (Cr-2) the number of instances of P among the candidates should be normalized to avoid ranking first the more frequent paths; and (Cr-3) users can specify whether they would like longer or shorter paths on top of the ranking. To satisfy all requirements enumerated above, we designed the following ranking function:

$$Rank_Q(P) = \alpha \underbrace{\frac{|\mathsf{PEXT}_{\mathcal{G}}(P)|}{|\mathsf{PEXT}_{\mathcal{G}}(Q)|}}_{\text{Cr-1}} + \beta \underbrace{\frac{\sigma(P)}{max\{\sigma(P_i) : P_i \in \mathcal{C}\}}}_{\text{Cr-2}} + \gamma \underbrace{\frac{|Q| - |P|}{max\{|Q|, |P|\}}}_{\text{Cr3}}, \quad (2)$$

where each criterion is weighted using constants α, β, and γ; and σ(P) returns the total count of instances of path P in $\mathcal{C}$. The parameters α, β, and γ can be defined by a user to match her expectations of the rank, and are set to 1 in our experiments. In Equation 2, we implement Cr-1 using the Jaccard index to find those paths whose PEXT is similar enough to the query one. Remember that we only compute $\mathsf{PEXT}(Q)$ extensively, while $\mathsf{PEXT}(P)$ is obtained from the former, and will always be a subset of it. Therefore, the Jaccard index is reduced to the simple ratio in Cr-1.

Using this tri-criteria function users are allowed to tune the ranking in order to favor, e.g., shorter paths setting $\gamma = +1$, or longer ones using $\gamma = -1$. Similarly, the other two criteria can be highlighted or decreased tuning α and β.

5 Experimental Evaluation

In this section, we present the set up of our experiments considering four different KGs, and discuss results that are encouraging for further research in this area.

5.1 Experimental Setup

We design a set of test cases using four large and well-known KGs, generated from different sources and tailored to different domains. To the best of our knowledge, there is no benchmark to evaluate equivalence between relation paths. Zhang et al. [23] have a crowd-sourced gold standard for single relations not covering complex relation paths, so it does not meet our requirements. We consider the generation of a gold standard for relation paths as future work, and here our goal is to examine the capabilities of our proposed approach for finding Δ-equivalences. For each dataset we propose two queries that are evaluated and the Δ-equivalent paths are ranked. Furthermore, for the top-5 results of each query, we present the values of the three criteria, and the ranking score.

Dataset. Our proposed method relies on KG exploration to search for equivalences. We noticed that more interesting results arise from KGs with a rich set of

Table 1: Statistics of knowledge graphs used in our experiments.

	NELL	DBpedia	YAGO3	WordNet
#Entities - $\|V\|$	1.2M	1.2M	2.6M	10K
#Relations - Σ_E	520	644	36	18
#Triples - $\|E\|$	3.8M	4M	5.5M	141K

interconnected entities and relation paths. A possible reason is that in KGs with low depth relation paths are rather short and more prone to contain loops with inverse relations. Here, we use four commonly used and general human knowledge KGs of different size and domain (Table 1 shows their characteristics): (A) Never Ending Learning (NELL) [19] is a knowledge base generated continuously by a never-ending machine learning system that crawls web pages and extracts facts from a pre-defined set of categories and relations; (B) YAGO3 [16] applies similar techniques to NELL, but it limits its sources to Wikipedia and WordNet [18]; (C) DBpedia [13] is a crowd-sourced knowledge base extracted from Wikipedia and Geonames as sources; and (D) WordNet [18] is a large lexical database of English, linking words and concepts using cognitive synonyms.

Queries. To date, there is no publicly available gold standard for relation paths equivalence. The generation of such a gold standard requires a deep understanding of the structure and semantics of a knowledge graph, which is not always available, showing the importance of having methods as the one described here. Therefore, we evaluate our method by using a set of eight manually generated queries, and leave as future work the generation of such a gold standard. For each knowledge graph in Table 1, we manually generated two relation paths that are used as queries. The queries are of different length, include inverse relations when possible, and cover different topics according to the domain of the KGs.

Implementation. In general, we implemented our approach using *Python* 3.5. We use *adjacency lists* (which provides a time complexity $\mathcal{O}(1)$ to find neighbors of a node) as data structure for storing the knowledge graphs in memory. Because our method considers inverse relations, we include the symmetric closure $\mathcal{G}^{\pm}$ of the KG for easy traversal. As an optimization, the discovery of connecting paths can be performed in parallel, giving one pair of nodes to a thread/process which computes connecting paths between them, and stores the results in a shared variable among threads. All experiments are executed in a Linux virtual machine configured with 40GB of RAM and 10 processing cores of 2.20GHz.

5.2 Results

As shown in Table 2, the running time of the algorithm varies between 35 and 193 minutes when considering 1000 elements in the relation path extensions. We noticed that such limit is reached in queries B.1, B.2, and C.1 only, while for all other cases a size of 500 should be sufficient. To put the running time in contrast, we also ran the same experiments using a limit size of 100 in the relation path extensions, and got running times between 4 and 60 minutes. Running time values depend on the size of the KG (see Table 1) and on the cardinality of the

relations included in the relation paths. Table 2 shows the queries for the first three dataset with the ranked equivalences. For each relation path answer, we report the values of each criterion, and the final ranking score w.r.t. Equation 2.

Remember that what we extract are Δ-equivalences, and in order to identify PEXT-equivalences from them, we try to determine whether $Q \equiv P$ for a given path P in the results of Q. We capture this comparison by using the Jaccard similarity between the full PEXT of a query and its Δ-equivalent paths. If the Jaccard similarity is 1, we can safely state that $Q \equiv P$; otherwise, the similarity value gives us a relative indication of how similar they are. We noticed that checking whether $Q \equiv P$ for the top results of queries over NELL, DBpedia and YAGO3 quickly becomes infeasible, and we stopped our processes after 24 hours. We associate this high computation time to the existence of relations with many-to-many cardinality and large number of instances which, as mentioned earlier, directly affects the computation of PEXT.

On the other hand, we have WordNet queries $\langle$_hypernym$\rangle$and $\langle$_part_of$\rangle$, which are composed of single relations that usually do not contain many range values for a given entity. In these cases, it was easy to compute the full PEXT and generate a score for the equivalences. We found that $\langle$_hypernym$\rangle \equiv \langle$_hyponym$^{-1}\rangle$ and $\langle$_part_of$\rangle \equiv \langle$_has_part$^{-1}\rangle$ are 87.7% and 87.5% similar, respectively, according to the KG. In theory they are strictly equivalent, indicating the effectiveness of our ranking function. Because of the characteristics of our method, we cannot tell if these numbers should be higher, but could be considered as a sign that the KG is incomplete.

We also observed that some query results differ from the query in length and building relations. A clear example of it is given by the path $P = \langle$cityLiesOnRiver^{-1},generalizations,generalizations$^{-1}\rangle$, which ranks higher for query A.1, and contains new and different relations from the query. Also, its semantics "*a city has a river, and city has a generalization*" is different from the original relation path query semantics. In NELL [19], the relation generalizations is common and acts as a meta relation for classes of entities, e.g., the generalization of a city is *Location*. So, the second part of P is a loop decoded as "*a city is a location, and location is a generalization of a city*". A deeper study is required to analyze other possible equivalences between individual relations that appear in the results, such as cityLiesOnRiver and riverFlowsThroughCity, or athletePlaysForTeam and teamMember, or citySportsTeams and teamPlaysInCity.

YAGO query results are also interesting. First result for query B.1 is a loop around wasBornIn, while the 2nd, 3rd and 4th results show Δ-equivalence to a single relation, having a high CR-3. Interestingly, for query B.2, we get an empty path as result, which indicates that actors are also directors of movies where they perform. And that some actors are married to the directors of the movies.

Results for DBpedia queries C.1 and C.2 are usually high in CR-2, meaning that the data are relatively complete. For C.1 we get an interesting results using a relation path between songs and their artists (band members) as query. Equivalent paths usually include bands associated to the artists, and even genres and instruments of the artists. In query C.2, we can see that the relation

Table 2: Top-5 best ranked Δ-equivalent paths for example queries.

Query	$Rank_Q(P)$	Cr-1	Cr-2	Cr-3
Query A.1: ⟨riverEmptiesIntoRiver,riverFlowsThroughCity⟩	Time ca. 53.2 min.		519 instances	
⟨cityLiesOnRiver^{-1},generalizations,generalizations^{-1}⟩	1.342	0.688	0.987	-0.333
⟨riverFlowsThroughCity,generalizations,generalizations^{-1}⟩	1.337	0.688	0.982	-0.333
⟨cityLiesOnRiver^{-1},generalizations,generalizations^{-1},generalizations^{-1}⟩	1.192	0.692	1.000	-0.500
⟨riverFlowsThroughCity,generalizations,generalizations^{-1},generalizations^{-1}⟩	1.189	0.692	0.997	-0.500
⟨riverEmptiesIntoRiver,cityLiesOnRiver^{-1}⟩	1.015	0.996	0.019	0.000
Query A.2: ⟨athletePlaysForTeam,teamHomeStadium,stadiumLocatedInCity⟩	Time ca. 35 min.		326 instances	
⟨athletePlaysForTeam,generalizations,generalizations^{-1},citySportsTeams^{-1}⟩	1.404	0.770	0.884	-0.250
⟨athletePlaysForTeam,generalizations,generalizations^{-1},teamPlaysInCity⟩	1.353	0.764	0.839	-0.250
⟨teamMember^{-1},generalizations,generalizations^{-1},citySportsTeams^{-1}⟩	1.264	0.739	0.775	-0.250
⟨athletePlaysForTeam,teamPlaysAgainstTeam^{-1},teamPlaysAgainstTeam^{-1}, citySportsTeams^{-1}⟩	1.249	0.531	0.969	-0.250
⟨teamMember^{-1},generalizations,generalizations^{-1},teamPlaysInCity⟩	1.244	0.733	0.761	-0.250
Query B.1: ⟨wasBornIn,isLocatedIn⟩	Time ca. 118 min.		1000^b instances	
⟨wasBornIn,isLocatedIn,isLocatedIn^{-1},isLocatedIn⟩	1.276	0.776	1.000	-0.500
⟨isCitizenOf⟩	1.025	0.016	0.001	0.500
⟨isPoliticianOf⟩	0.517	0.013	0.001	0.500
⟨livesIn⟩	0.514	0.007	0.000	0.500
⟨hasGender,hasGender^{-1},isPoliticianOf⟩	0.507	0.332	0.243	-0.333
Query B.2: ⟨actedIn,directed^{-1}⟩	Time ca. 193 min.		1000^b instances	
⟨actedIn,isLocatedIn,isLocatedIn^{-1},directed^{-1}⟩	1.360	0.860	1.000	-0.500
⟨hasGender,hasGender^{-1}⟩	0.587	0.583	0.004	0.000
⟨actedIn,actedIn^{-1},actedIn,directed^{-1}⟩	0.524	0.674	0.350	-0.500
⟨ϵ⟩^{a}	0.518	0.018	0.000	0.500
⟨isMarriedTo⟩	0.503	0.003	0.000	0.500
Query C.1: ⟨artist,bandMember⟩	Time ca. 58 min.		1000^b instances	
⟨artist,associatedMusicalArtist^{-1},associatedBand,bandMember⟩	1.435	0.935	1.000	-0.500
⟨artist,associatedBand^{-1},associatedMusicalArtist,bandMember⟩	1.429	0.935	0.994	-0.500
⟨artist,associatedBand^{-1},associatedMusicalArtist,associatedBand^{-1}⟩	0.953	0.524	0.929	-0.500
⟨artist,associatedMusicalArtist^{-1},associatedBand,associatedMusicalArtist^{-1}⟩	0.952	0.524	0.928	-0.500
⟨genre,instrument,instrument^{-1},genre^{-1}⟩	0.736	0.432	0.804	-0.500
Query C.2: ⟨academicAdvisor,almaMater⟩	Time ca. 80 min.		335 instances	
⟨academicAdvisor,birthPlace,birthPlace^{-1},almaMater⟩	1.080	0.580	1.000	-0.500
⟨academicAdvisor,deathPlace,birthPlace^{-1},almaMater⟩	0.846	0.575	0.771	-0.500
⟨almaMater⟩	0.641	0.121	0.020	0.500
⟨academicAdvisor,deathPlace,deathPlace^{-1},almaMater⟩	0.587	0.620	0.467	-0.500
⟨notableStudent^{-1},almaMater⟩	0.540	0.459	0.081	0.000

(a) ϵ denotes the empty path
(b) maximum size of the PEXT set reached

path ⟨academicAdvisor,almaMater⟩ can be Δ-equivalent to ⟨almaMater⟩, i.e., people usually graduate from the university where their supervisor studied. We also get that they usually were a notableStudent for their supervisor.

6 Related Work

Research on equivalences in knowledge graphs (bases) has been mainly focused on single relations, which are emulated by our approach as paths of unitary length. In [23], authors address the problem of mining equivalent relations from Linked Data datasets as a clustering problem using an equivalence score. Similarly, [20] deals with the problem of finding synonymous relations in Linked Data using an itemset mining approach on the domain and range types of a relation

instance. Both [23,20] require the notion of typed entities and do not consider inverse relations, which are significant shortcomings for mining knowledge graphs. Here, we consider variable-length relation paths with inverse relations, and although we do not require any kind of schema knowledge, our method can benefit from it, e.g., in the interpretation of paths. [8] used distributional semantics to find semantically related class type paths, i.e. meta paths, using latent feature space. Our work is also related to AMIE+ [9], a system for mining Horn rules in knowledge bases, where in each rule the body is a path and the head is a relation. Our work is orthogonal to [9] since we do not focus on mining rules, but rather on ranking the most prominent equivalences for a given relation path. The application of our method for rule mining, considering more generic rules than Horn rules, is part of our future work. Last but not least, [23] describes an annotation process that could be used for generating a gold standard applicable to evaluation of the presented approach and other similar experiments.

7 Conclusions and Future Work

We explored the problem of identifying relation path equivalences in KGs using a data-driven, unsupervised and domain independent method. We addressed several practical and theoretical issues regarding finding strict equivalences, and proposed a more efficient, approximate approach that is still able to bring valuable insights. Using different test queries, we show that our approach can efficiently rank candidates that are Δ-equivalent. In our experiments, we achieved results consistent with our initial assumptions, as our method retrieved intuitively similar relation paths that were, however, of different length and contained different relations when compared to the input query.

As a part of our future work, we intend to perform a user evaluation of the ranking results to: (1) come up with a universally applicable gold standard for relation paths equivalence, and (2) determine the influence of particular weights in the tri-criteria function on the performance of the method across different use cases. We are also interested in using the Δ-equivalences between relation paths to improve embedding and prediction methods that require a deeper knowledge of the KG structure. An example could be generating similar embeddings for entities that have equivalent relation paths.

References

1. Abiteboul, S., Hull, R., Vianu, V., eds.: Foundations of Databases: The Logical Level. 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)
2. Baeza, P.B.: Querying graph databases. In: PODS, ACM (2013) 175–188
3. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: Containment of conjunctive regular path queries with inverse. In: KR, Morgan Kaufmann (2000) 176–185
4. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: Reasoning on regular path queries. SIGMOD Record **32**(4) (2003) 83–92

5. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: STOC, ACM (1977) 77–90
6. Consens, M.P., Mendelzon, A.O.: GraphLog: a visual formalism for real life recursion. In: PODS, ACM Press (1990) 404–416
7. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In: KDD, ACM (2014) 601–610
8. Freitas, A., da Silva, J.C.P., Curry, E., Buitelaar, P.: A distributional semantics approach for selective reasoning on commonsense graph knowledge bases. In: Natural Language Processing and Information Systems - 19th International Conference on Applications of Natural Language to Information Systems, NLDB 2014, Montpellier, France, June 18-20, 2014. Proceedings. (2014) 21–32
9. Galárraga, L., Teflioudi, C., Hose, K., Suchanek, F.M.: Fast rule mining in ontological knowledge bases with AMIE+. VLDB J. **24**(6) (2015) 707–730
10. Kostylev, E.V., Reutter, J.L., Romero, M., Vrgoc, D.: SPARQL with property paths. In: International Semantic Web Conference (1). Volume 9366 of Lecture Notes in Computer Science., Springer (2015) 3–18
11. Lao, N., Cohen, W.W.: Relational retrieval using a combination of path-constrained random walks. Mach. Learn. **81**(1) (October 2010) 53–67
12. Lao, N., Subramanya, A., Pereira, F.C.N., Cohen, W.W.: Reading the web with learned syntactic-semantic inference rules. In: EMNLP-CoNLL, ACL (2012) 1017–1026
13. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia - a large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web **6**(2) (2015) 167–195
14. Lin, X., Liang, Y., Guan, R.: Compositional learning of relation paths embedding for knowledge base completion. CoRR **abs/1611.07232** (2016)
15. Lin, Y., Liu, Z., Sun, M.: Modeling relation paths for representation learning of knowledge bases. CoRR **abs/1506.00379** (2015)
16. Mahdisoltani, F., Biega, J., Suchanek, F.M.: YAGO3: A knowledge base from multilingual wikipedias. In: CIDR, www.cidrdb.org (2015)
17. Mendelzon, A.O., Wood, P.T.: Finding regular simple paths in graph databases. SIAM J. Comput. **24**(6) (1995) 1235–1258
18. Miller, G.A.: WordNet: A lexical database for english. Commun. ACM **38**(11) (1995) 39–41
19. Mitchell, T.M., Cohen, W.W., Jr., E.R.H., Talukdar, P.P., Betteridge, J., Carlson, A., Mishra, B.D., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E.A., Ritter, A., Samadi, M., Settles, B., Wang, R.C., Wijaya, D.T., Gupta, A., Chen, X., Saparov, A., Greaves, M., Welling, J.: Never-ending learning. In: AAAI, AAAI Press (2015) 2302–2310
20. Morzy, M., Lawrynowicz, A., Zozulinski, M.: Using substitutive itemset mining framework for finding synonymous properties in Linked Data. In: RuleML. Volume 9202 of Lecture Notes in Computer Science., Springer (2015) 422–430
21. Pichler, R., Skritek, S.: Containment and equivalence of well-designed SPARQL. In: PODS, ACM (2014) 39–50
22. Vardi, M.Y.: A note on the reduction of two-way automata to one-way automata. Inf. Process. Lett. **30**(5) (1989) 261–264
23. Zhang, Z., Gentile, A.L., Augenstein, I., Blomqvist, E., Ciravegna, F.: Mining equivalent relations from Linked Data. In: ACL (2), The Association for Computer Linguistics (2013) 289–293