ABSTRACT

We focus on the problem of aligning ontology relations, namely finding relation names that correspond to the same or related concepts. Such alignment is a prerequisite to the integration of the multiple available Knowledge Bases many of which include similar concepts, differently termed. We propose a novel approach for this problem, by leveraging association rules – originally mined in order to enrich the ontological content. Here, we treat the rules as Datalog programs and look for bounded-depth sub-programs that are contained in (or equivalent to) each other. Heads of such programs intuitively correspond to related concepts, and we propose them as candidates for alignment. The candidate alignments require further verification by experts; to this end we accompany each aligned pair with explanations based on the provenance of each relation according to its sub-program. We have implemented our novel solution in a system called Datalog Alignment. We propose to demonstrate Datalog Alignment, presenting the aligned pairs that it finds, and the computed explanations, in context of real-life Knowledge Bases.

PVldb Reference Format:
DOI: https://doi.org/10.14778/3352063.3352087

1. INTRODUCTION

In large-scale ontologies, it is common to see semantically related – or even equivalent – concepts that are captured by different relations. For example, DBpedia [2] includes the birthplace, birthPlace and placeOfBirth relations that intuitively have the same semantic meaning, as well as other related relations such as countryOfBirth. When we look at multiple ontologies – say YAGO [11] and DBpedia – we observe many further cases of semantically related relations, such as wasBornIn in YAGO or birthPlace in DBpedia.

Ontology alignment aims at merging such semantically equivalent relations, thereby enriching the ontological knowledge. Some approaches to ontology alignment focus on similarity of relation content, but the latter may not reveal the full picture: for instance, the aforementioned birthplace and birthPlace DBPedia relations share less than 2% of common tuples. Due to the Open World Assumption, where absence of ontological data does not indicate its falseness, such low intersection rate is not necessarily an evidence of dissimilarity.

Instead, we focus in this work on an alternative novel approach, as follows. We leverage correlation rules between relations, which may be found using systems for mining logical rules from Knowledge Bases (KBs), such as AMIE [7]. Such system produces rules of the form wasBornIn(x,y) ∧ isLocatedIn(y,z) ⇒ isCitizenOf(x,z).

As observed in [5], the rules produced by AMIE, when combined, may be viewed as possibly recursive datalog programs. The programs may be very complex, and contain large number of rules. To illustrate, the number of different rules that may be used to directly derive a single fact in AMIE exceeds 20, many of these rules are recursive (in the datalog sense). While the rules may be intricate, they can reveal complex relationships between properties and concepts of the KBs.

Our main observation here is that with this view, alignment of ontology relations corresponds to equivalence of datalog programs – the rules of these programs are the fragments reachable from each of the relation names that one wishes to align. In the following example we illustrate such a simple program using a fragment of rules mined by AMIE.

Example 1.1. Consider the program with goal relation wasBornIn, consisting the following rules:

r1 wasBornIn(x,y) :- pb(x,y)

r2 wasBornIn(x,y) :- birthPlace(x,y), placeOfBirth(x,y)

r3 pb(x,y) :- placeOfBirth(x,y)

where pb stands for placeOfBirth, for the sake of brevity and avoiding confusion with placeOfBirth. Note that r3 was added to this program because pb appeared in the body of r1. Relations birthPlace and placeOfBirth do not appear in any head.

Further, this gives rise to another kind of correlation: if the corresponding datalog programs are not equivalent but rather strictly contained in each other, we can infer semantic containment of the relations.

Towards this end we present Datalog Alignment, a system for ontology alignment via datalog containment. The general framework of Datalog Alignment is depicted in Figure 1. The
system is comprised of two main components. The first, is a preprocessing step, which includes entity alignment, and rules mining (using AMIE) on a coalesced KB. On the second component, datalog programs are created from the mined rules, and the system searches for containments of programs such that their goal relations are from different KBs. Note that this approach is similarly applicable for the single KB case, where we refine the KB and find containments between relations. In this case we skip the preprocessing step as entities are already shared between relations.

Naturally, datalog containment is undecidable, and so if one of the sub-programs is recursive we approximate it by a bounded-depth variant. In addition, the datalog programs themselves are based on statistical correlations and are thus imprecise. We thus view our solution as one that can aid analysts in performing alignments, rather than a fully automated solution. In this respect, it is important that the proposed alignments are explained, to allow the analyst to understand them and verify their correctness. Our approach is unique in that it allows such explanations, based on the provenance of aligned relations with respect to the rules contained in the datalog programs.

We will demonstrate the usefulness of our system in the context of ontology alignment using the real-life data from YAGO and DBpedia. Using multiple examples, and interactively engaging the audience, we will explore the alignments proposed by Datalogment and their explanations.

Related Work. The problem of ontology alignment has been extensively studied and multiple approaches were developed to solve sub-problems in the field such as aligning entities, relations and classes of given ontologies. For instance, PARIS [10] receives as input two ontologies and produces one-to-one alignment of their entities, relations and classes. Additional specialized systems such as SORAL [8], propose a supervised machine learning model for specifically aligning relations across two ontologies based on sampling relations using SPARQL endpoint and evaluating relations similarity. Both do not provide explanation for alignments. To our knowledge, we are the first to propose ontology alignment via datalog containment.

The approach closest to ours is that of [6]. In [6] the authors define a set of alignment patterns called ROSA rules which are used for the alignment process. The idea of [6] subsumes state-of-the-art ontology matching by finding more complex correlations, e.g., that one “hop” in one KB can correspond to several hops in the other. ROSA rules are generated similarly to our preprocessing stage, of aligning entities, coalescing the KB and executing AMIE to mine rules. Each ROSA rule pattern consist of a single rule produced by AMIE. The approach we present here further processes the rules treating rules with similar heads as a datalog program. This general approach leads to the discovery of other complex correlation between the KBs that cannot be found using single rule patterns. For example, Datalogment finds containment between programs for which the intersection of their corresponding head relations in the KB is empty.

2. TECHNICAL DETAILS

We (informally) introduce the main technical notions involved in the development of Datalogment, using examples.
Rule of rules to a mining rule system (such as AMIE) that produces a set of idb relations in KB such that \[ R_i, R_j \in KB_i \text{ for } i \in \{1, 2\}. \] Each pair of relations \[ R^1 \in C_1, R^2 \in C_2 \] are compared in search for containment. To this end, the system generates the datalog program corresponding for each relation. Rules sharing the same relation as head relation are grouped into a single program, and the program is recursively extended by adding rules whose head atom appears in the body of some rule of the program. This is repeated until no more rules are added (or the program contains all rules of \( R \)). The same procedure may be applied for the single KB case, where we refine the KB and find containments between relations. The only difference in the latter case is that both \( C_1 \) and \( C_2 \) are from the same KB, essentially, pairwise comparison between \( R_1 \neq R_2 \in KB \).

We note that the datalog programs obtained by using AMIE in the above way are restricted, in the following sense. First, since relations in a KB are binary, so are the relations in the program. Further, we say that two body atoms of a rule are connected if they share a variable. A rule is connected if all its body atoms are transitively connected. A rule is closed if each of its variables (either in the head or body) appear at least twice. We observe that rules mined by AMIE are connected and closed. As example, rule \( A(x,y) : = A(x,z), C(z,y) \) is connected and closed, while \( A(x,y) : = A(x,y), C(z,y) \) is neither connected nor closed.

It is well known that general datalog containment is undecidable [9]. It is further undecidable when the programs are restricted to chain programs [3], such as those comprised of the first rule (connected and closed) in the above example.

### 2.3 Solution Overview

In light of the undecidability of the problem, we have implemented a simple heuristic algorithm that, naturally, is incomplete. We first recall that undecidability is due to the fact that datalog programs may be recursive. Indeed, containment is decidable for non-recursive datalog. Further analyzing (manually) the structure of the concrete datalog programs obtained from rules mined by AMIE, we note that although there is significant value in looking for indirect derivation rules, the “distance” of relations with respect to their appearance in rules matter. That is, if a fact \( f_0 \) is used in a rule that derives a fact \( f_1 \) and in turn \( f_1 \) is used in a rule that derives \( f_2 \), etc. up to fact \( f_n \), then naturally the strength of connection – and in turn the importance of \( f_0 \) appearing in the program of \( f_0 \), diminishes quite fast as \( n \) grows. Recall also that rules are associated with confidence values, usually less than 1; treating this confidence as (independent) probabilities, one may claim that the confidence in the connection also decreases exponentially with \( n \).

We define an expansion of a datalog rule in the usual sense, by the substitution of an idb predicate that appears in a rule’s body with the body of another rule having the aforementioned idb predicate as its head. The confidence for the resulting rule is the product of all rules that were used for expansion. The set of all possible expansion of a rule \( r \) may be viewed as a tree, where each node represents a rule body (i.e., a possible expansion). The root node is the body of \( r \), and children of a node \( r' \) are rule’s bodies obtained by a substitution of an idb predicate in \( r' \)’s body. Then, the expansions trees of rules \( r_1, \ldots, r_n \) with the same head relation \( R \) can be grouped into a single tree, rooted at \( R \), denoted as the expansion tree of \( R \).

We simply put a bound \( maxDepth \) on the depth of expansion trees we consider, thereby “approximating” the datalog program through a non-recursive one. We further prune nodes that have confidence below threshold, and our experience shows that this threshold needs to be quite high, e.g. 0.7. This also means that \( maxDepth \) parameter can be set to be quite low, e.g. 2 or 3. These parameters are of course easily configurable.

Then, the algorithm is quite straightforward. It is given as input two datalog programs \( P_1, P_2 \) with goal relations \( R_1, R_2 \) whose containment one wishes to check. It is also given as parameters \( maxDepth \) that limits the depth of expansion trees and a minimal confidence level as threshold under which nodes in the tree are discarded. It then exhaustively expand the two programs up to the required depth, computing the confidence of each node in the expansion tree as the multiplication of confidence values of its “ancestor” rules used for derivation. If this node has confidence below threshold it is discarded, otherwise it is kept, and we obtain Conjunctive Queries (CQs) whose containment can be verified via Chandra and Merlin algorithm [4].

**Explanations.** Each candidate containment is accompanied with an explanation for the analyst. An explanation is based on the (bounded) expansions trees of each relation in a containment pair. Assuming \( R_1 \) is contained in \( R_2 \), an explanation is an implicit mapping between each node of the expansion tree of \( R_1 \) to a containing node (CQ-wise) in the expansions tree of \( R_2 \). The system displays the expansions tree of each program for an analyst to interactively examine the derivation process and the containment relationships between corresponding nodes.

### 3. SYSTEM OVERVIEW

We have implemented our solution of ontology alignment based on datalog containment in a system prototype called Datalog. The prototype has been implemented in Python 3, and runs on a multi-core Linux server. The user interface was implemented in JavaScript and Node.js.

The general framework of our system is depicted in Figure 1. The system is comprised of two main components. As input we receive two KBs. The first step is to align the entities of both KBs using owl:sameAs links or by using an entity aligning system. Next, we coalesce the KBs into a single new KB while prefixing every relation name with a unique identifier corresponding to its original KB. AMIE is executed on the coalesced KB and rules are mined. Datalog programs are created from the mined rules. The system pairwise checks for containments of programs \( P_1, P_2 \) such
that the goal relation of $P_1$ is from $KB_1$ and the goal relation of $P_2$ is from $KB_2$ and outputs all containments for the user.

For the second step, the user provides the system two parameters: $\text{threshold}$ for pruning low confidence rules and $\text{maxDepth}$ for limiting the number of rule expansions. Once the inputs are fed, the system compute candidate containments using the method explained in the previous section, and present them to the user as shown in Figure 2a. The user can then examine the results by browsing the entire list, or filter the results to quickly find the relation of interest. Each containment is associated with an explanation, that intuitively present the sequence of rules expansion “leading” to the containment, and the user can double click on any containment pair to explore its explanation.

The explanations screen (Figure 2b) allows the user to explore how different rules are utilized in the expansion process of the algorithm. Each program is presented using a hierarchical graph, depicting the expansion tree structure. The root node of each program is its goal relation, each child node represents a possible expansion using a single rule of the program, and the label of each rule used for expansion is shown on the edge. The confidence level in each node, based on the rules used to derive it is displayed in brackets. Double clicking on a single node in program 1 highlights the node itself along with one of its containing nodes in program 2. Whereas, double clicking on a node in program 2, highlights it and all nodes in program 1 which it contains.

4. DEMO SCENARIO

We will demonstrate the usefulness of our system in the context of ontology alignment using real-world data from YAGO and DBpedia. We will use rules mined by AMIE on a coalesced version of the KBs. The participants will be asked to play the role of data analyst, examining the usefulness of Datalignment in the process of ontology alignment.

We will start by presenting the audience the pre-mined rules and let them set the $\text{threshold}$ and $\text{maxDepth}$ parameters. Next, we will run the system and examine the proposed alignments and containments. We will walk the audience through the process of analyzing the results, as follows. We will ask the participant to pick a proposed alignment pair in order to further examine the corresponding programs. Upon selection of a pair we will let the user expand the programs graphs, looking at the relevant rules. If the participant finds a rule that appears erroneous to her, she will be able to delete it and re-execute the system, observing the effect of her decision on the alignment results.

The system provides a sample of tuples for the relations enabling the analyst to justify the correctness of containments. If needed, we will allow the participants to perform further iterations until verifying the alignment results that are of interest to her.

Finally, we will let the audience “look under the hood”. In particular, we will show relevant tuples from the original KBs of the relations pairs of which we explore, as well as relevant intermediate results of our algorithm execution.

Acknowledgements. This research has been funded by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 804302) and the Israeli Ministry of Science, Technology and Space.

5. REFERENCES


