Leveraging Linked Data to Discover Semantic Relations Within Data Sources

Mohsen Taheriyan(✉), Craig A. Knoblock, Pedro Szekely, and José Luis Ambite

University of Southern California Information Sciences Institute, Marina del Rey, USA
{mohsen,knoblock,pszekely,ambite}@isi.edu

Abstract. Mapping data to a shared domain ontology is a key step in publishing semantic content on the Web. Most of the work on automatically mapping structured and semi-structured sources to ontologies focuses on semantic labeling, i.e., annotating data fields with ontology classes and/or properties. However, a precise mapping that fully recovers the intended meaning of the data needs to describe the semantic relations between the data fields too. We present a novel approach to automatically discover the semantic relations within a given data source. We mine the small graph patterns occurring in Linked Open Data and combine them to build a graph that will be used to infer semantic relations. We evaluated our approach on datasets from different domains. Mining patterns of maximum length five, our method achieves an average precision of 75 % and recall of 77 % for a dataset with very complex mappings to the domain ontology, increasing up to 86 % and 82 %, respectively, for simpler ontologies and mappings.

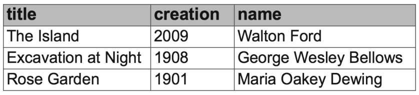
Keywords: Semantic model · Semantic relation · Semantic label · Linked data · Semantic web

1 Introduction

A critical task in generating rich semantic content from information sources such as relational databases and spreadsheets is to map the data to a domain ontology. Manually mapping data sources to ontologies is a tedious task. Several approaches have been proposed to automate this process [3,4,6–12,17,18], nonetheless, most approaches focus on *semantic labeling*, annotating data fields, or *source attributes*, with classes and/or properties of a domain ontology. However, a precise mapping needs to describe the semantic relations between the source attributes in addition to their semantic types.

In our earlier Karma work [5], we build a graph from learned semantic types and a domain ontology and use this graph to map a source to the ontology. Since only using the ontology does not necessarily generate accurate models, we had the user in the loop to interactively refine the suggested mappings. Later, we

DOI: 10.1007/978-3-319-46523-4_33



title	creation	name
The Island	2009	Walton Ford
Excavation at Night	1908	George Wesley Bellows
Rose Garden	1901	Maria Oakey Dewing

Fig. 1. Sample data from the Crystal Bridges Museum of American Art

introduced an automatic approach that exploits the mappings already created for similar sources in addition to the domain ontology to learn a model for a new unknown source (*target source*) [15]. One limitation of this approach is that the quality of the generated mappings is highly dependent on the availability of the previous mappings. However, in many domains, there are none or very limited number of known mappings available, but we still want to learn the mapping of a new source without requiring the user to map some similar sources first.

We present a novel approach that exploits Linked Open Data (LOD) to automatically infer the semantic relations within a given data source. LOD contains a vast amount of semantic data in many domains that can be used to learn how instances of different classes are linked to each other. The new approach leverages the instance-level data in LOD rather than schema-level mappings as in previous work. Our work in this paper focuses on learning the relationships between the source attributes once they are annotated with semantic labels. The main contribution of our work is leveraging the graph patterns occurring in the linked data to disambiguate the relationships between the source attributes. First, we mine graph patterns with different lengths occurring in the linked data. We combine these patterns into one graph and expand the resulting graph using the paths inferred from the domain ontology. Then, we explore the graph starting from the semantic labels of the source to find the candidate mappings covering all the labels.[1]

We evaluated our approach on different datasets and in different domains. Using patterns of maximum length five, our method achieved an average precision of 75 % and recall of 77 % in inferring the semantic relations within a dataset with complex mappings to the domain ontology, including 13.5 semantic types and 12.5 object properties on average per mapping. The precision and recall are over 80 % for datasets with simpler mappings. Our evaluation shows that longer patterns yield more accurate semantic models.

2 Motivating Example

In this section, we provide an example to explain the problem of inferring semantic relations within structured sources. We want to map a data source containing data about artwork in the Crystal Bridges Museum of American Art (Fig. 1) to the CIDOC-CRM ontology (www.cidoc-crm.org). We formally write the signature of this source as *s(title, creation, name)* where *s* is the name of the source and *title*, *creation*, and *name* are the names of the source attributes (columns).

[1] This paper is a significantly extended version of a workshop paper [16].

The first step in mapping the source *s* to the ontology is to label its attributes with semantic types. We formally define a semantic type to be a pair consisting of a domain class and one of its data properties ⟨*class_uri,property_uri*⟩ [6,9]. In this example, the correct semantic types for the columns *title*, *creation*, and *name* are ⟨*E35_Title,label*⟩, ⟨*E52_Time-Span,P82_at_some_time_within*⟩, and ⟨*E82_Actor_Appellation,label*⟩. A mapping that only includes the types of the attributes is not complete because it does not necessarily reveal how the attributes relate to each other. To build a precise mapping, we need a second step that determines how the semantic labels should be connected to capture the intended meaning of the data. Various techniques can be employed to automate the labeling task [4,6–10,17,18], however, in this work, we assume that the labeling step is already done and we focus on inferring the semantic relations.

We use a conceptual graph called *semantic model* to represent the complete set of mappings between the sources and ontologies. In a semantic model, the nodes correspond to ontology classes and the links correspond to ontology properties. Figure 2 shows the correct semantic model of the source *s*. As we can see in the figure, none of the semantic types corresponding to the source columns are directly connected to each other, which makes the problem of finding the correct semantic model more complex. There are many paths in the CIDOC-CRM ontology connecting the assigned labels. For instance, we can use the classes *E39_Actor* and *E67_Birth* to relate the semantic types *E82_Actor_Appellation* and *E52_Time-Span*:

(*E39_Actor*, *P1_is_identified_by*, *E82_Actor_Appellation*)
(*E39_Actor*, *P98i_was_born*, *E67_Birth*)
(*E67_Birth*, *P4_has_time-span*, *E52_Time-Span*)

However, this way of modeling does not correctly represent the semantics of this particular data source (Fig. 2 shows the correct model). In general, the ontology defines a large space of possible semantic models and without additional knowledge, we do not know which one is a correct interpretation of the data.

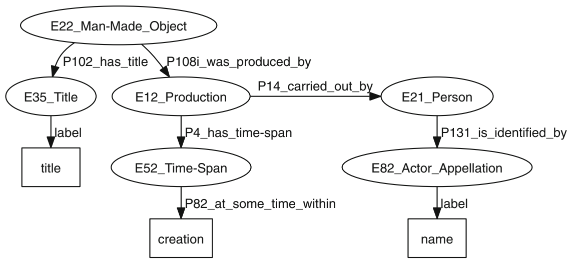


Fig. 2. The semantic model of the source *s*. Class nodes (ovals) correspond to ontology classes and data nodes (rectangles) correspond to the source attributes

The LOD cloud includes a vast and growing collection of semantic content published by various data providers in many domains. Suppose that some other museums have already mapped their data to the CIDOC-CRM ontology and published it as linked data. Can we exploit the available linked data as background knowledge to infer relationships between attributes of s? The basic idea of our approach is to leverage this linked data to bias the search space to prefer those relationships that are used for related sources. Once we have identified the semantic types of the source attributes, we can search the linked data to find the frequent patterns connecting the corresponding classes. For example, by querying the available linked data corpus, we find that *P131_is_identified_by* is more popular than *P1_is_identified_by* to connect instances of *E82_Actor_Appellation* and instances of *E21_Person*, and this makes sense when we investigate the definitions of these two properties in the ontology. The property *P1_is_identified_by* describes the naming or identification of any real world item by a name or any other identifier, and *P131_is_identified_by* is a specialization of *P1_is_identified_by* that identifies a name used specifically to identify an instance of *E39_Actor* (superclass of *E21_Person*). We can query the linked data to find longer paths between entities. For instance, by inspecting the paths with length two between the instances of *E22_Man-Made_Object* and *E21_Person*, we observe that the best way to connect these two classes is through the path: *E22_Man-Made_Object* $\xrightarrow{P108\mathit{i\_was\_produced\_by}}$ *E12_Production* $\xrightarrow{P14\mathit{\_is\_carried\_out\_by}}$ *E21_Person.*

3 Inferring Semantic Relations

In this section, we explain our approach to automatically deduce the attribute relationships within a data source. The inputs to our approach are the domain ontology, a repository of (RDF) linked data in the same domain, and a data source whose attributes are already labeled with the correct semantic types.[2] The output is a semantic model expressing how the assigned labels are connected.

3.1 Extracting Patterns from Linked Data

Given the available linked data, we mine the schema-level patterns connecting the instances of the ontology classes. Each pattern is a graph in which the nodes correspond to ontology classes and the links correspond to ontology properties. For example, the pattern $c_1 \xrightarrow{p} c_2$ indicates that at least one instance of the class c_1 is connected to an instance of the class c_2 with the property p.

Depending on the structure of the domain ontology, there might be a large number of possible patterns for any given number of ontology classes. Suppose that we want to find the patterns that only include the three classes c_1, c_2, and c_3. Figure 3 exemplifies some of the possible patterns to connect these classes. We define the length of a pattern as the number of links (ontology properties) in a pattern. Thus, the general forms of the patterns with length two will be $c_1 \rightrightarrows c_2$, $c_1 \rightleftarrows c_2$, $c_1 \rightarrow c_2 \rightarrow c_3$ (chain), $c_1 \rightarrow c_2 \leftarrow c_3$ (V-shape), or $c_2 \leftarrow c_1 \rightarrow c_3$ (A-shape).

[2] For this paper, we assume that the correct semantic types are given, but our approach can support the more general case where a set of candidate semantic types are assigned to each attribute.

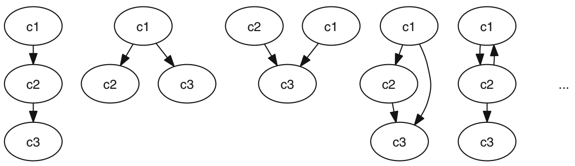


Fig. 3. Sample structures of the patterns connecting the classes c_1, c_2, and c_3

We can write SPARQL queries to extract patterns with different lengths from a triplestore. For example, the following SPARQL query extracts the patterns with length one and their frequencies from the linked data:

```
SELECT DISTINCT ?c1 ?p ?c2 (COUNT(*) as ?count)
WHERE { ?x ?p ?y. ?x rdf:type ?c1. ?y rdf:type ?c2. }
GROUP BY ?c1 ?p ?c2
```

Pattern mining is a preprocessing step in our approach and can be done offline. Yet, using SPARQL queries to extract long patterns from a large number of triples is not efficient. For example, having a Virtuoso repository containing more than three million triples on a Linux machine with a 2.4 GHz Intel Core CPU and 16 GB of RAM, the response time to the query to extract V-shape patterns of length two was roughly 1 h. We were only able to collect a few forms of the patterns with length three, four, and five in a 5-hour timeout. The problem is that the SPARQL queries to extract long patterns include many joins and there is no binding between the variables in the queries and the classes and properties in the ontology. Therefore, we adopt a different approach to extract patterns of length two or more.

Algorithm 1 shows our method to find patterns from a triplestore. This is a recursive algorithm that incrementally generates longer patterns by joining shorter patterns with patterns of length one. The intuition is that we enumerate the candidate patterns and create SPARQL queries whose variables are bound to ontology classes and properties rather than writing a join-heavy query with unbound variables. This technique allows us to exploit the indexes created by triplestore over subjects, predicates, objects, and different combinations of them.

First, we use the SPARQL query shown above to retrieve all the patterns of length one from the linked data. Then, we construct candidate patterns of length two by joining patterns of length one with themselves (join on the shared ontology class). For example, the pattern $c1 \xrightarrow{p_1} c_2$ can be joined with $c_1 \xrightarrow{p_2} c_3$ on c_1 to construct a pattern of length two: $c_2 \xleftarrow{p_1} c_1 \xrightarrow{p_2} c_3$. For each candidate pattern, we query the linked data to see whether an instance of such pattern exists in the data. Once we find the patterns of length two that occur in the linked data, we join them with patterns of length one and check the occurrence of the newly formed patterns (length three) in the triplestore. We can repeat the same process to retrieve longer patterns. To prevent generating duplicate patterns, we perform some optimizations in the code that are not shown in Algorithm 1. Using this algorithm, we could extract all patterns of length 1, 2,

Algorithm 1. Extract LD Patterns

```
    Input: LD (the linked data repository), k (maximum length of patterns)
    Output: A set of LD patterns
 1: $P_1 \leftarrow$ extract patterns of length one from LD
 2: $P \leftarrow P_1$, $i \leftarrow 1$
 3: while $i < k$ do
 4:     for each pattern $p_i \in P_i$ do
 5:         for each ontology class $c$ in $p_i$ do
 6:             $P_{1,c} \leftarrow$ all the patterns in $P_1$ that include $c$
 7:             for each pattern $p_{1,c} \in P_{1,c}$ do
 8:                 $p_{join} \leftarrow$ construct a pattern by joining $p_i$ and $p_{1,c}$ on node $c$
 9:                 if $p_{join}$ exists in LD then
10:                     $P_{i+1} \leftarrow P_{i+1} \cup p_{join}$
11:                 end if
12:             end for
13:         end for
14:     end for
15:     $P \leftarrow P \cup P_{i+1}$, $i \leftarrow i + 1$
16: end while
    return $P$
```

Algorithm 2. Construct Graph G

```
    Input: LD Patterns, Semantic Types, Domain Ontology
    Output: Graph $G$
    ▷ Add LD patterns
 1: sort the patterns descending based on their length
 2: exclude the patterns contained in longer patterns
 3: merge the nodes and links of the remaining patterns into $G$
    ▷ Add Semantic Types
 4: for each semantic type $\langle$class_uri,property_uri$\rangle$ do
 5:     add the class to the graph if it does not exist in $G$
 6: end for
    ▷ Add Ontology Paths
 7: for each pair of classes $c_i$ and $c_j$ in $G$ do
 8:     find the directed and inherited properties between $c_i$ and $c_j$ in the ontology
 9:     add the properties that do not exist in $G$
10: end for
    return $G$
```

3, and 4 from the same triplestore in only 10 min, and all patterns of length 5 in less than one hour.

3.2 Merging Linked Data Patterns into a Graph

Once we extract the patterns from the linked data (LD patterns), we combine them into a graph G that will be used to infer the semantic relations. Building the graph has three parts: (1) adding the LD patterns, (2) adding the semantic labels assigned to the source attributes, and (3) expanding the graph with the paths inferred from the ontology.

The graph G is a weighted directed graph in which nodes correspond to ontology classes and links correspond to ontology properties. The algorithm to construct the graph is straightforward (Algorithm 2). First, we add the LD patterns to G. We start from the longer patterns and merge the nodes and links of patterns into G if they are not subgraphs of the patterns already added to the graph (lines 1–3). Next, we add the semantic types of the target source for which we want to learn the semantic model (lines 4–6). As we mentioned before, we assume that the source attributes have already been labeled with semantic types. Each semantic type is a pair consisting of a domain class and one of its data properties ⟨*class_uri,property_uri*⟩. If G does not contains any node with the label *class_uri*, we add a new node to G and label it with *class_uri*. The final step in building the graph is to find the paths in the ontology that relate the current classes in G. The goal is to connect class nodes of G using the direct paths or the paths inferred through the subclass hierarchy in the ontology.

The links in G are weighted. We adopt a subtle approach to assign weights to the links. There are two types of links; the links that are added from the LD patterns, and the links that do not exist in any pattern and are inferred from the ontology. The weight of the links in the former group has an inverse relation with the frequency of the links. If the number of instances for the pattern $c1 \xrightarrow{p} c_2$ in the linked data is x and the total number of instances of patterns with length one is n, the weight of the link p from c_1 to c_2 in G is computed as $1 - x/n$. Since we are generating minimum-cost models in the next section, this weighting strategy gives more priority to the links occurring more frequently in the linked data. For the links in the second category, the ones added from the ontology, we assign a much higher weight comparing to the links in the LD patterns. The intuition is that the links used by other people to model the data in a domain are more likely to represent the semantics of a given source in the same domain. One reasonable value for the weight of the links added from the ontology is the total number of object properties in the ontology. This value ensures that even a long pattern costs less than a single link that does not exist in any pattern.

The other important feature of the links in the graph is their *tags*. We assign an identifier to each pattern added to the graph and annotate the links with the identifiers of the supporting patterns. Suppose that we are adding two patterns $m_1 : c_1 \xrightarrow{p_1} c_2 \xrightarrow{p_2} c_3$ and $m_2 : c_1 \xrightarrow{p_1} c_2 \xrightarrow{p_3} c_4$ to G. The link p_1 from c_1 to c_2 will be tagged with $\{m_1, m_2\}$, the link p_2 from c_2 to c_3 will have only $\{m_1\}$ as its tag set, and the link p_3 from c_2 to c_4 will be tagged with $\{m_2\}$. We use the link tags later to prioritize the models containing larger segments from the LD patterns.

We provide an example to help the reader to understand our algorithm for creating the graph. Suppose that we are trying to infer a semantic model for the source s in Fig. 1 and we only extract the patterns with length one and two from the available linked data. Table 1 lists the extracted LD patterns. Figure 4 shows the graph constructed using Algorithm 2. The black links are the links added from the LD patterns. The weight of these links is calculated as $1 - (link\ frequency)/(sum\ of\ frequencies)$. For example, the weight of the link *P98i_was_born* from *E21_Person* to *E67_Birth* is $1 - 5/63 = 0.92$ (total number of instances of the patterns with length one is 63). The black links are also tagged with the identifier of the patterns they originate from. For instance, the link *P98i_was_born* is tagged with m_1 because the pattern m_1 contains this link. Note that only the patterns m_1, m_2, m_3, and m_4 are added to the graph and the rest of the patterns are ignored by the algorithm. This is because the patterns m_5, m_6, m_7, m_8, m_9, and m_{10} are subgraphs of the longer patterns that are already added to the graph (line 2 in Algorithm 2). The blue node *E53_Title* does not exist in any pattern, however, it is added to the graph since the semantic type of the column *title* in the source s, ⟨*E53_Title,label*⟩, contains *E53_Title* (line 5 in

Table 1. The sample patterns extracted from the linked data. Each pattern is a set of (c_1, p, c_2) triples where c_1 and c_2 are ontology classes and p is an ontology property. The second and third columns are the length and frequency of patterns (only the frequency of the patterns with length one matters in our algorithm)

Id	Pattern	Len	Freq
m_1	(*E21_Person,P98i_was_born,E67_Birth*), (*E67_Birth,P4_has_time-span,E52_Time-Span*)	2	-
m_2	(*E22_Man-Made_Object,P108i_was_produced_by,E12_Production*), (*E12_Production,P4_has_time-span,E52_Time-Span*)	2	-
m_3	(*E22_Man-Made_Object,P14_carried_out_by,E39_Actor*), (*E39_Actor,P131_is_identified_by,E82_Actor_Appellation*)	2	-
m_4	(*E21_Person,P131_is_identified_by,E82_Actor_Appellation*)	1	12
m_5	(*E22_Man-Made_Object,P108i_was_produced_by,E12_Production*)	1	8
m_6	(*E12_Production,P4_has_time-span,E52_Time-Span*)	1	3
m_7	(*E22_Man-Made_Object,P14_carried_out_by,E39_Actor*)	1	10
m_8	(*E39_Actor,P131_is_identified_by,E82_Actor_Appellation*)	1	20
m_9	(*E21_Person,P98i_was_born,E67_Birth*)	1	5
m_{10}	(*E67_Birth,P4_has_time-span,E52_Time-Span*)	1	5

Algorithm 2). The red links are the links that do not exist in the LD patterns but are inferred from the ontology. For example, the red link *P14_carried_out_by* is added because *E21_Person* is a subclass of the class *E39_Actor*, which is in turn the range of the object property *P14_carried_out_by*. We assign a high weight to the red links, e.g., the total number of object properties in the ontology (in this example, assume that the ontology consists of 100 object properties).

3.3 Generating and Ranking Semantic Models

The final part of our approach is to compute the semantic model of the source s from the graph. First, we map the semantic types of s to the nodes of the graph. In our example, the semantic types are ⟨*E53_Title,label*⟩, ⟨*E52_Time-Span,P82_at_some_time_within*⟩, and ⟨*E82_Actor_Appellation,label*⟩, and they will be mapped to the nodes *E53_Title*, *E52_Time-Span*, and *E82_Actor_Appellation* of the graph in Fig. 4. Then, we compute the top k trees connecting the mapped nodes based on two metrics: *cost* and *coherence*. Cost of a tree is the sum of the link weights. Because the weights of the links have an inverse relation with their popularity, computing the minimum-cost tree results in selecting more frequent links. However, selecting more popular links does not always yield the correct semantic model. The coherence of a model is another important factor that we need to consider. Coherence in this context means the ratio of the links in a computed tree that belong to the same LD pattern. The coherence metric gives priority to the models that contain longer patterns. For example, a model that includes one pattern with length 3 will be ranked higher than a model including two patterns with length 2, and the latter in turn will be preferred over a model with only one pattern with length 2. Our algorithm prefers the coherence over the cost in choosing the best model. If two models are equivalent as far as coherence, the model with the lowest cost will be ranked higher.

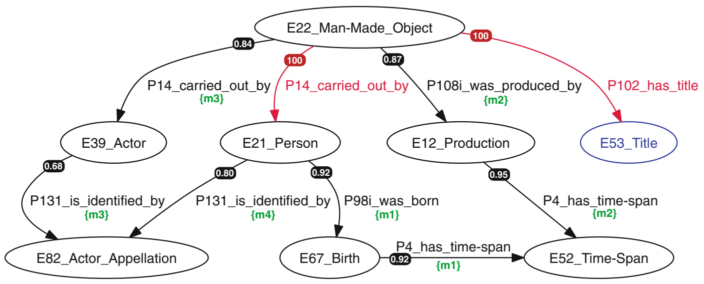


Fig. 4. The graph constructed from the patterns in Table 1, the semantic types of the source s, and an example subset of the CIDOC-CRM ontology

The algorithm that finds the top k trees is a customized version of the BANKS algorithms [1]. The BANKS algorithm computes the top k minimum-cost trees that span a subset of the nodes in a graph (the nodes that the semantic types are mapped to). It creates one iterator for each of the nodes corresponding to the semantic types, and then the iterators follow the incoming links to reach a common ancestor. The BANKS algorithm uses the iterator's distance to its starting point to decide which link should be followed next. To incorporate the coherence into the algorithm, we use a heuristic that prefers the links that are parts of the same pattern even if they have higher weights. In our example, the algorithm creates three iterators, one starting from the node *E53_Title*, one from *E52_Time-Span*, and one from *E82_Actor_Appellation*. Each iterator has a queue consisting of the candidate links to traverse. At each step, the algorithm chooses a link and adds the incoming links of the source node of the selected link to the queue of the corresponding iterator. At the beginning, the candidates are:

e_1 (*itr 1*): (*E22_Man-Made_Object,P102_has_title,E53_title*), $distance = 100$
e_2 (*itr 2*): (*E67_Birth,P4_has_time-span,E52_Time-Span*), $distance = 0.92$
e_3 (*itr 2*): (*E12_Production,P4_has_time-span,E52_Time-Span*), $distance = 0.95$
e_4 (*itr 3*): (*E21_Person,P131_is_identified_by,E21_Actor_Appellation*), $distance = 0.80$
e_5 (*itr 3*): (*E39_Actor,P131_is_identified_by,E21_Actor_Appellation*), $distance = 0.68$

The algorithm pulls e_5 from the queue of the third iterator because e_5 is the lowest-cost link in the candidates. Then, it inserts the incoming links of the source node of e_5 (*E39_Actor*) to the queue:

e_6 (*itr 3*): (*E22_Man-Made_Object,P14_carried_out_by,E39_Actor*), $distance = 1.52$

Now the candidate links to traverse are e_1, e_2, e_3, e_4, and e_6. Although the distance of e_4 to the starting point is less than other links, our algorithm prefers e_6. This is because e_6 is part of the pattern m_3 which includes the previously traversed link e_5. Considering the coherence in traversing the graph forces the algorithm to converge to models that include larger segments from the patterns.

Once we compute top k trees, we rank them first based on their coherence and then their cost. The model in Fig. 5 shows the top semantic model computed by our algorithm for the source s. Our algorithm ranks this model higher than a model that has *E21_Person* instead of *E39_Actor*, because the model in Fig. 5 is more coherent and has lower cost (the weight of the link *P14_carried_out_by* from *E22_Man-Made_Object* to *E21_Person* is 100, while the link with the same label from *E22_Man-Made_Object* to *E39_Actor* has a weight of 0.84).

It is important to note that the trees containing longer patterns are not necessarily more coherent. Suppose that a source has two semantic labels A and B, and there are only two LD patterns: $p_1 = \{(A, e_1, B)\}$ and $p_2 = \{(A, e_2, B)(B, e_3, C)\}$. After constructing the graph, we compute top k trees based on both coherence and cost. The candidate trees are $T_1 = \{(A, e_1, B)\}$, $T_2 = \{(A, e_2, B)\}$, $T_3 = \{(A, e2, B)(B, e3, C)\}$, and $T_4 = \{(A, e1, B)(B, e3, C)\}$. The trees T_1, T_2, and T_3 have a coherence value equal to 1 (all the links belong to one pattern), while coherence of T_4 is 0.5 (one link is from p_1 and one link is from p_2). Between T_1, T_2, and T_3, our algorithm ranks T_3 third because it is longer (higher cost) and ranks T_1 and T_2 as the top two trees (based on the frequency of e_1 and e_2 in the linked data). Thus, the algorithm does not always prefer the longer patterns.

4 Evaluation

To evaluate our approach, we performed four experiments on different datasets, using different ontologies, and in different domains. The details of the experiments are as follows:

– **Task 1**: The dataset consists of 29 museum data sources in CSV, XML, or JSON format containing data from different art museums in the US. The domain ontology is CIDOC-CRM, and the background linked data is the RDF triples generated from all sources in the dataset except the target source (leave-one-out settings).

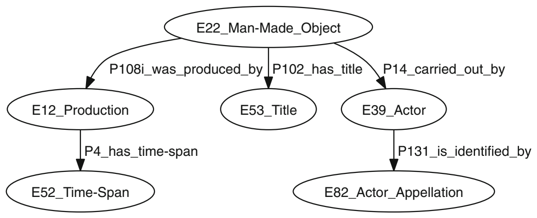


Fig. 5. The top semantic model computed for the source s

- **Task 2**: The dataset and the domain ontology are the same as the ones in Task 1, but we use the linked data published by Smithsonian American Art Museum[3] as the background knowledge. This linked data repository contains more than 3 million triples mapping the artwork collection in Smithsonian museum to the CIDOC-CRM ontology.
- **Task 3**: The dataset is the same as the one in Task 1, but we use a different domain ontology, EDM[4], which is much simpler and less structured than CIDOC-CRM. The goal is to evaluate how our approach performs with respect to different representations of knowledge in a domain. We use a leave-one-out setting is this task.
- **Task 4**: The dataset includes 15 data sources in a domain containing ads for weapons. The ontology is an extension of the schema.org ontology, and the experiment adopts a leave-one-out setting.

Table 2 shows more details of the evaluation tasks. In each task, we applied our approach on the dataset to find the top 50 candidate semantic models for each source. We then ranked the candidate models and compared the first ranked models with the gold standard models created by an expert in the test domain. The datasets including the sources, the domain ontologies, and the gold standard models are available on GitHub.[5] The source code of our approach is integrated into Karma which is available as open source.[6]

We assume that the correct semantic labels for the source attributes are known. The goal is to see how well our approach learns the relationships having the correct semantic types. For example, from the total of 825 links in the ground-truth models in Task 1, 458 of them correspond to data properties connecting the semantic types to the source attributes. Therefore, the 367 internal links are the semantic relations that we want to infer using our approach.

Table 2. The evaluation tasks

	Dataset		Ontology		Ground Truth	
	#sources	#attributes	#classes	#properties	#nodes	#links
Task 1	29	458	147	409	852	825
Task 2	29	458	147	409	852	825
Task 3	29	329	119	351	470	441
Task 4	15	175	736	1081	261	246

We measured the accuracy of the computed semantic models by comparing them with the gold standard models in terms of *precision* and *recall*. Assuming that the correct semantic model of the source s is sm and the semantic model learned by our approach is sm', we define the precision and recall as:

[3] http://americanart.si.edu/collections/search/lod/about.
[4] http://pro.europeana.eu/page/edm-documentation.
[5] https://github.com/taheriyan/iswc-2016.
[6] https://github.com/usc-isi-i2/Web-Karma (directory: `karma-research`).

$$precision = \frac{rel(sm) \cap rel(sm')}{rel(sm')}, \quad recall = \frac{rel(sm) \cap rel(sm')}{rel(sm)}$$

where $rel(sm)$ is the set of triples (u, v, e) in which e is a link from the class node u to the class node v in the semantic model sm. Note that we do not consider the links from the classes to the source attributes since they are part of the semantic types that are given as input to our algorithm. Consider the semantic model of the source s in Fig. 2, *rel(sm)={(E22_Man-Made_Object, P108i_was_produced_by, E12_Production)*, $\cdots$}. Assuming that the semantic model in Fig. 5 is what our algorithm infers for s, we will have $precision = 3/5 = 0.6$ and $recall = 3/5 = 0.6$. Note that we are using a strict evaluation metric. The learned semantic model is not semantically wrong because *E21_Person* is a subclass of *E39_Actor* according to the CIDOC-CRM ontology. However, our formula treats both (*E22_Man-Made_Object, P14_carried_out_by, E39_Actor*) and (*E39_Actor, P131_is_identified_by, E82_Actor_Appellation*) as incorrect links.

Table 3. The evaluation results

	Task 1		Task 2		Task 3		Task 4	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
Max length of patterns = 0	0.07	0.05	0.07	0.05	0.01	0.01	0.03	0.02
Max length of patterns = 1	0.60	0.60	0.28	0.29	0.85	0.78	0.84	0.79
Max length of patterns = 2	0.64	0.67	0.53	0.58	0.81	0.81	0.83	0.79
Max length of patterns = 3	0.67	0.68	0.55	0.60	0.84	0.83	0.86	0.81
Max length of patterns = 4	0.74	0.76	0.55	0.60	0.83	0.82	0.86	0.82
Max length of patterns = 5	0.75	0.77	0.61	0.67	0.83	0.82	0.86	0.82

Table 3 shows the average precision and recall for all sources in the evaluation tasks for different maximum lengths of LD patterns (the input parameter k in Algorithm 1). The maximum length 0 means that we did not incorporate the LD patterns and only used the domain ontology to build a graph on top of the semantic labels. An interesting observation is that when we do not use the LD patterns, the precision and recall are close to zero. This low accuracy comes from the fact that in most of the gold standard models, the attributes are not directly connected and there are multiple paths between each pair of classes in the ontology (and thus in our graph), and without additional information we cannot resolve the ambiguity. Leveraging LD patterns as background knowledge yields a remarkable improvement in both precision and recall compared to the case in which we only consider the domain ontology. Since we are using the pattern frequencies in assigning the weights to the links of the graph, using patterns of length one means that we are only taking into account the popularity of the links in computing the semantic models. Leveraging longer patterns improves both precision and recall. This means that considering coherence in addition to the link popularity empowers our approach to derive more accurate models.

As we can see in the table, the precision and recall in Task 2 are lower than Task 1 even though they have the same dataset and ontology. The reason is that Task 2 employs the triples from the Smithsonian museum rather than the triples generated from other sources in the same dataset, and the overlap between these triples and the ground truth models is less than Task 1. The results in Task 3 are better than both Task 1 and Task 2 because the ground-truth models created using EDM are simpler and smaller than the models created using CIDOC-CRM (441 links vs. 825 links). Although CIDOC-CRM is well structured, the level of ambiguity in inferring the relations is more than simpler ontologies with a flat structure because there are many links (and paths) between each pair of classes (many properties are inherited through the class hierarchy). We achieve high precision and recall in Task 4. The ontology in this task is much larger than the ontologies in the other tasks (cf. Table 2), however, the hierarchy level is less than the CIDOC-CRM ontology, and the models are also smaller. We believe that extracting longer patterns (length > 5) will improve the results for Task 1 and Task 2 as many of the semantic models in these tasks include more than 5 internal links. However, as Table 3 suggests, longer patterns do not contribute to the precision and recall for Task 3 and Task 4.

The results show that our method suggests semantic models with high precision and recall for large, real-world datasets even in complex domains. There are several reasons explaining why we did not achieve higher accuracy in Task 1 and Task 2. First and foremost, there were sources in the dataset whose semantic model contained structures that did not exist in any pattern. In other words, some sources did not have much overlap with other sources in the dataset. A simple example of this case is where the semantic model of a source includes an ontology class that does not have any instance in the linked data, and consequently, no LD pattern contains relations of that class. A second important reason is that in some cases, more than one pattern can be used to connect semantic labels. For example, many sources contain two columns labeled with the class *E52_Time-Span*. We have a pattern with length four in which one *E52_Time-Span* is connected to *E12_Production* and the other one is connected to *E67_Birth* (or *E69_Death*). We also have another pattern with length four in which one *E52_Time-Span* is connected to *E67_Birth* and the second one is linked to *E69_Death*. In such situations, our algorithm may select the wrong pattern. Inspecting these cases convinced us that taking into account longer patterns will resolve the issue. For a few sources, the ground-truth semantic model was not a rooted tree, while our algorithm computes only tree candidate models. Therefore, we missed some of the links in our learned semantic models. Finally, our strict evaluation metric penalizes the precision and recall even though some of the learned models are not semantically wrong. For instance, the correct model of one source includes the link *P138_has_representation* from *E22_Man-Made_Object* to *E38_Image*. Our algorithm infers the inverse link *P138i_represents* to connect these two classes because it is more frequently used in other models, and thus, it is more popular in the linked data. Our evaluation considers this link as an incorrect inference. Another example is the one we discussed earlier where our method suggests *E39_Actor* instead of *E21_Person*. We are exploring the possibility of defining a looser evaluation metric that provides more flexible interpretation of "correct" models. Our initial idea is to consider the subclass and subproperty definitions in the ontology. For example, we can give some credit if the algorithm infers a parent property of the one in the ground-truth model (instead of considering it completely wrong).

To compare the work in this paper with our previous approach that exploits the known semantic models [15], we applied the previous approach on the same dataset in Task 1, which resulted in semantic models with 81 % precision and 82 % recall. The reason why the accuracy is lower in the current work is that we only used patterns of maximum length five in our experiment. On the other hand, our previous work exploits complete semantic models of previously modeled sources, which are more coherent than the small graph patterns we used in this work. Our work in this paper complements our previous work in more common and critical case where few, if any, known semantic models are available.

We ran our experiments on a single machine with a Mac OS X operating system and a 2.3 GHz Intel Core i7 CPU. The total time to extract all the patterns of length one, two, three, and four from our Virtuoso repository was less than 10 min, and it was approximately one hour for the patterns of length five. Then, we fed the extracted patterns to Algorithm 2. Let T be the time from combining LD patterns into a graph until generating and ranking candidate semantic models. The average value of T is different when we run the code with different maximum length for patterns. However, the average value of T never exceeds 5 s (for most of the sources, the actual value of T was less than a second).

5 Related Work

There have been many studies to automatically describe the semantics of data sources as a mapping from the source to an ontology. Since the focus of our work is on inferring the semantic relations, we compare our work with the ones that pay attention to inferring semantic relationship and not only semantic labeling.

Limaye et al. [7] use the YAGO ontology to annotate web tables and generate binary relationships using machine learning approaches. Venetis et al. [17] present a scalable approach to describe the semantics of tables on the Web. To recover the semantics of tables, they leverage a database of class labels and relationships automatically extracted from the Web. They attach a class label to a column if a sufficient number of the values in the column are identified with that label in the database of class labels, and analogously for binary relationships. Although these approaches are very useful in publishing semantic data from tables, they are limited in learning the semantics of sources as a united model. Both of these approaches only infer individual binary relationships between pairs of columns. They are not able to find the link between two columns if no relation is directly instantiated between the values of those columns. Our approach can connect one column to another one through a path in the ontology.

Carman and Knoblock [2] use known source descriptions to learn a semantic description that describes the relationship between the inputs and outputs of a source. However, their approach is limited in that it can only learn sources whose models are subsumed by the models of known sources. That is, the description of a new source is a conjunctive *combination* of known source descriptions.

Our work is closely related to other work leveraging the Linked Open Data (LOD) cloud to capture the semantics of sources. Mulwad et al. [8] use *Wikitology* [14], an ontology which combines some existing manually built knowledge systems such as DBpedia and Freebase, to link cells in a table to Wikipedia entities. They query the background LOD to generate initial lists of candidate classes for column headers and cell values and candidate properties for relations between columns. Then, they use a probabilistic graphical model to find the correlation between the columns headers, cell values, and relation assignments. The quality of the semantic data generated by this category of work is highly dependent on how well the data can be linked to the entities in LOD. While for most popular named entities there are good matches in LOD, many tables contain domain-specific information or numeric values (e.g., temperature and age) that cannot be linked to LOD. Moreover, these approaches are only able to identify individual binary relationships between the columns of a table. However, an integrated and united semantic model is more than fragments of binary relationships between the columns. In a complete semantic model, the columns may be connected through a path including the nodes that do not correspond to any column in the table.

The most closely related work to ours is the recent work by Schaible et al. [13]. They extract *schema-level patterns* (SLPs) from LOD and generate a ranked list of vocabulary terms for reuse in modeling tasks. The main difference between their work and our method is the complexity of the patterns. SLPs are (sts, ps, ots) triples where sts and ots are sets of RDF types and ps is a set of RDF properties. For example, the SLP ({*Person,Player*}, {knows},{Person,Coach}) indicates that some instances of *Person* $\cap$ *Player* are connected to some instances of *Person* $\cap$ *Coach* via the property *knows*. In our approach, we mine graph patterns that are more complex than SLPs allowing us to automatically compose a complete semantic model for a target source rather than presenting recommendations in an interactive mapping task.

6 Discussion

We presented a novel approach to infer semantic relations within structured sources. Understanding how the source attributes are related is an essential part of building a precise semantic model for a source. Such models are the key ingredients to automatically integrate heterogeneous data sources. They also automate the process of publishing semantic data on the Web. The core idea of our work is to exploit the small graph patterns occurring in the Linked Open Data to hypothesize attribute relationships within a data source.

Manually constructing semantic models, in addition to being time-consuming and error-prone, requires a thorough understanding of the domain ontologies. Tools such as Karma can help users to model data sources through a graphical user interface. Yet, building the models in Karma without any automation requires significant user effort. Incorporating our method in source modeling tools enables them to infer an initial semantic model for the input source that can be transformed to the correct model with only a few user actions.

The evaluation shows that our approach infers the semantic relations with a high precision and recall for a dataset with very complex semantic models (on average 13.5 classes and 12.6 object properties per semantic model). We have shown that we gain higher precision and recall when we apply our method on data sources with simpler models. The results support the theory that more accurate models can be constructed when longer LD patterns are used. We observed that the structure of the patterns also affects the quality of the learned models. For example, using only the chain-shape patterns resulted in more precise models for some of the sources. One direction of our future work is to investigate the correlation between the shape of the LD patterns, the structure of the domain ontology, and the ground-truth semantic models. This can help us to incorporate certain types of patterns when mapping a data source to a domain ontology.

Our work plays a role in helping communities to produce consistent Linked Data so that sources containing the same type of data use the same classes and properties when published in RDF. Often, there are multiple correct ways to model the same type of data. A community is better served when all the data with the same semantics is modeled using the same classes and properties. Our work encourages consistency because our algorithms bias selection of classes and properties towards those used more frequently in existing data.

Acknowledgments. This research was supported in part by the National Science Foundation under Grant No. 1117913 and in part by Defense Advanced Research Projects Agency (DARPA) via AFRL contract numbers FA8750-14-C-0240 and FA8750-16-C-0045. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF, DARPA, AFRL, or the U.S. Government.

References

1. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword searching and browsing in databases using BANKS. In: Proceedings of the 18th International Conference on Data Engineering, pp. 431–440 (2002)
2. Carman, M.J., Knoblock, C.A.: Learning semantic definitions of online information sources. J. Artif. Intell. Res. **30**(1), 1–50 (2007)
3. Ding, L., DiFranzo, D., Graves, A., Michaelis, J., Li, X., McGuinness, D.L., Hendler, J.A.: TWC Data-gov Corpus: incrementally generating linked government data from data.gov. In: Proceedings of the 19th WWW, pp. 1383–1386 (2010)

4. Han, L., Finin, T.W., Parr, C.S., Sachs, J., Joshi, A.: RDF123: from spreadsheets to RDF. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 451–466. Springer, Heidelberg (2008)
5. Knoblock, C.A., Szekely, P., Ambite, J.L., Goel, A., Gupta, S., Lerman, K., Muslea, M., Taheriyan, M., Mallick, P.: Semi-automatically mapping structured sources into the semantic web. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 375–390. Springer, Heidelberg (2012)
6. Ramnandan, S.K., Mittal, A., Knoblock, C.A., Szekely, P.: Assigning semantic labels to data sources. In: Gandon, F., Sabou, M., Sack, H., d'Amato, C., Cudré-Mauroux, P., Zimmermann, A. (eds.) ESWC 2015. LNCS, vol. 9088, pp. 403–417. Springer, Heidelberg (2015)
7. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and searching web tables using entities, types and relationships. PVLDB **3**(1), 1338–1347 (2010)
8. Mulwad, V., Finin, T., Joshi, A.: Semantic message passing for generating linked data from tables. In: Alani, H., Kagal, L., Fokoue, A., Groth, P., Biemann, C., Parreira, J.X., Aroyo, L., Noy, N., Welty, C., Janowicz, K. (eds.) ISWC 2013, Part I. LNCS, vol. 8218, pp. 363–378. Springer, Heidelberg (2013)
9. Pham, M., Alse, S., Knoblock, C.A., Szekely, P.: Semantic labeling: a domain-independent approach. In: Proceedings of the 15th International Semantic Web Conference (ISWC) (2016)
10. Polfliet, S., Ichise, R.: Automated mapping generation for converting databases into linked data. In: ISWC Posters & Demos (2010)
11. Sahoo, S.S., Halb, W., Hellmann, S., Idehen Jr., K., T.T., Auer, S., Sequeda, J., Ezzat, A.: A survey of current approaches for mapping of relational databases to RDF. W3C RDB2RDF XG report (2009)
12. Saquicela, V., Vilches-Blazquez, L.M., Corcho, O.: Lightweight semantic annotation of geospatial RESTful services. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part II. LNCS, vol. 6644, pp. 330–344. Springer, Heidelberg (2011)
13. Schaible, J., Gottron, T., Scherp, A.: TermPicker: enabling the reuse of vocabulary terms by exploiting data from the linked open data cloud. In: Sack, H., Blomqvist, E., d'Aquin, M., Ghidini, C., Ponzetto, S.P., Lange, C. (eds.) ESWC 2016. LNCS, vol. 9678, pp. 101–117. Springer, Heidelberg (2016). doi:10.1007/978-3-319-34129-3_7
14. Syed, Z., Finin, T.: Creating and exploiting a hybrid knowledge base for linked data. In: Filipe, J., Fred, A., Sharp, B. (eds.) ICAART 2010. CCIS, vol. 129, pp. 3–21. Springer, Heidelberg (2011)
15. Taheriyan, M., Knoblock, C., Szekely, P., Ambite, J.L.: Learning the semantics of structured data sources. Web Seman.: Sci., Serv. Agents World Wide Web **37**, 152–169 (2016)
16. Taheriyan, M., Knoblock, C., Szekely, P., Ambite, J.L., Chen, Y.: Leveraging linked data to infer semantic relations within structured sources. In: Proceedings of the 6th International Workshop on Consuming Linked Data (COLD) (2015)
17. Venetis, P., Halevy, A., Madhavan, J., Paşca, M., Shen, W., Wu, F., Miao, G., Wu, C.: Recovering semantics of tables on the web. Proc. VLDB Endow. **4**(9), 528–538 (2011)
18. Wang, J., Wang, H., Wang, Z., Zhu, K.Q.: Understanding tables on the web. In: Atzeni, P., Cheung, D., Ram, S. (eds.) ER 2012 Main Conference 2012. LNCS, vol. 7532, pp. 141–155. Springer, Heidelberg (2012)