A Scalable Approach to Learn Semantic Models of Structured Sources

Mohsen Taheriyan, Craig A. Knoblock, Pedro Szekely, José Luis Ambite
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292, USA
{mohsen, knoblock, szekely, ambite}@isi.edu

***Abstract*—Semantic models of data sources describe the meaning of the data in terms of the concepts and relationships defined by a domain ontology. Building such models is an important step toward integrating data from different sources, where we need to provide the user with a unified view of underlying sources. In this paper, we present a scalable approach to automatically learn semantic models of a structured data source by exploiting the knowledge of previously modeled sources. Our evaluation shows that the approach generates expressive semantic models with minimal user input, and it is scalable to large ontologies and data sources with many attributes.**

I. Introduction

A significant amount of information available on the Web is available in sources such as relational databases, spreadsheets, XML, JSON, and Web APIs. A common approach to integrate these sources involves building a domain model and constructing *source descriptions* that represent the intended meaning of the data by specifying mappings between the sources and the domain model [1]. In the Semantic Web, the domain model is an ontology that defines concepts and relationships within a domain, and source descriptions are formal specifications of *semantic models*. Semantic models can be viewed as graphs with ontology classes as the nodes and ontology properties as the links between the nodes.

Manually constructing semantic models is a time-consuming task that requires significant effort and expertise. Automatically generating these models involves two steps. The first step is specifying the *semantic types*, i.e., labeling each data field, or *source attribute* with a class or a data property of the domain ontology. However, simply annotating the attributes is not sufficient. A precise semantic model needs a second step that specifies the relationships between the source attributes in terms of the properties in the ontology. In Semantic Web research, there are many studies on mapping data sources to ontologies [2]–[7], but most focus on the first step of the modeling process or are very limited in automatically inferring the relationships.

In our previous work [8], we presented a novel approach to learn semantic models of data sources from *known semantic models*, semantic models of sources that have already been modeled. The work is inspired by the idea that different sources in the same domain often provide similar or overlapping data and have similar semantic models. Given sample data from the new source, we use an existing machine learning technique [9] to label each source attribute with a semantic type from the ontology. Next, we build a weighted directed graph with known semantic models as the main components and expand the graph by adding the paths in the ontology connecting the nodes across different components. This graph models the space of plausible semantic models. Then, we produce mappings from the semantic types to the nodes of the graph, and for each mapping we generate a candidate model by computing the minimal tree that connects the mapped nodes. Finally, we score the candidate models to prefer the ones formed with more coherent and frequent patterns.

This past work has some limitations. First, we do not take into account the uncertainty of the machine learning algorithm when labeling the source attributes. That is, although the machine learning algorithm learns a set of candidate semantic types for each source attribute, we assign the one with higher confidence value to the attributes and ignore the other suggested semantic types. This is a strict assumption, because in many cases, the learning algorithm cannot distinguish between the semantic types of similar data values. Second, for data sources with many attributes, the number of mappings between the semantic types and the nodes of the graph can be large. In these cases, processing all the mappings to generate the candidate models is infeasible.

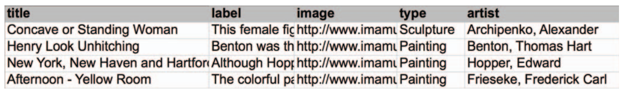
In this paper, we address the limitations of our past work. We generalize the previous approach to consider a set of candidate semantic types for each attribute rather than only one semantic type per attribute. To overcome the problem of large number of mappings, we introduce a search algorithm that explores the space of possible mappings as we map the semantic types to the nodes of the graph and expands only the more promising mappings. We evaluated the approach on a set of museum data sources modeled using relatively large ontologies (119 classes and 351 properties). The evaluation shows that the approach generates rich semantic models with minimal user input, and it also scales well, learning semantic models of data sources that contain many attributes.

II. Example

In this section, we provide an example to demonstrate the problem of learning semantic models. We will use this example in the rest of the paper to illustrate different steps of our approach. In this example, the goal is to model a

978-1-4799-4003-5/14 $31.00 © 2014 IEEE
DOI 10.1109/ICSC.2014.13

set of museum data sources using EDM (www.europeana.eu/schemas/edm), AAC (www.americanartcollaborative.org/ontology), SKOS (www.w3.org/2008/05/skos#), Dublin Core Metadata Terms (purl.org/dc/terms), FOAF (xmlns.com/foaf/0.1), ORE (www.openarchives.org/ore/terms), and ElementsGr2 (rdvocab.info/ElementsGr2) ontologies and then use the created semantic models to publish their data as RDF [10]. Suppose that we want to model a data source containing data of artworks in the Indianapolis Museum of Art (www.imamuseum.org). Figure 1 shows examples of the data values in this source. We formally write this source as $s(title, label, image, type, artist)$ where s is the name of the source and *title*, *label*, *image*, *type*, and *artist* are names of the source attributes (columns).



title	label	image	type	artist
Concave or Standing Woman	This female fig	http://www.imamu	Sculpture	Archipenko, Alexander
Henry Look Unhitching	Benton was th	http://www.imamu	Painting	Benton, Thomas Hart
New York, New Haven and Hartfor	Although Hopp	http://www.imamu	Painting	Hopper, Edward
Afternoon - Yellow Room	The colorful pa	http://www.imamu	Painting	Frieseke, Frederick Carl

Fig. 1. The source $s(title, label, image, type, artist)$ contains information about artworks in the Indianapolis Museum of Art.

First, we must determine the semantic type of each source attribute. For example, the attribute *title* declares the title of a cultural heritage object and the attribute *artist* specifies the name of a person. Then, we need to identify the relationships between the classes used to model the attributes. Unless the relationship between the person and the cultural heritage object is explicitly specified, we do not know whether the person is the creator, sitter, or copyrights holder of the object. The precise semantic model of s is depicted in Figure 2.

A semantic model of the source s, $sm(s)$, is a directed graph containing two types of nodes. *Class nodes* (ovals in Figure 2) correspond to classes in the ontology and are labeled with class URIs. *Data nodes* (rectangles in Figure 2) correspond to the source attributes and are labeled with the attribute names. The links in the graph correspond to ontology properties and are labeled with property URIs.

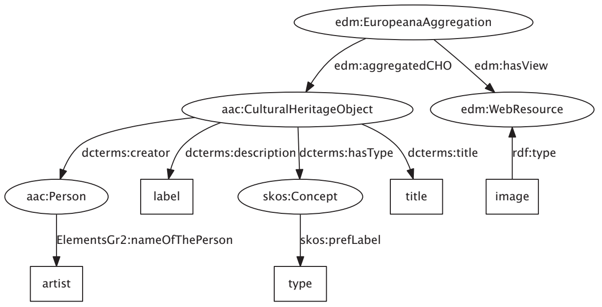


Fig. 2. Semantic model of the source s. Class nodes and links are labeled with URIs (prefixed by the ontology namespace).

Automatically building a semantic model for a new source is difficult. Machine learning methods can help us in assigning semantic types to the attributes, however, these methods are error prone when similar data values have different semantic types. Extracting the relationships between the attributes is a more complicated problem. There might be multiple paths connecting two classes in the ontology and we do not know which one models the intended meaning of the data.

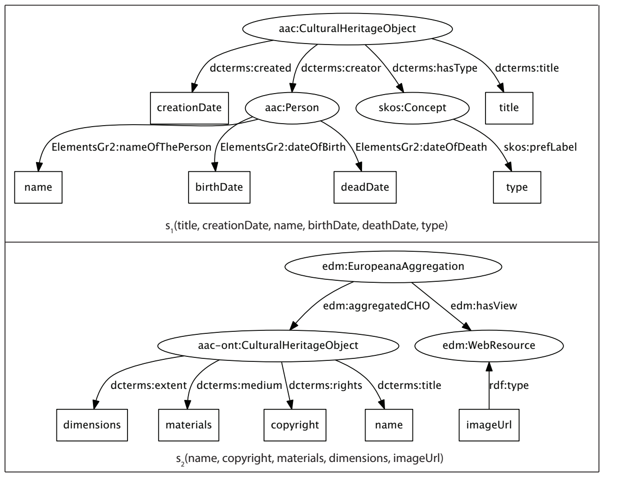


Fig. 3. Semantic models of s_1 (artworks at Dallas Museum) and s_2 (artworks at The Metropolitan Museum of Art).

In general, the ontology defines a large space of possible semantic models and without additional information, we do not know which one describes the source more precisely. In this work, we present a scalable approach that exploits the knowledge of previously modeled sources to limit the search space and to hypothesize correct semantic models. The main idea is that data sources in the same domain usually provide overlapping data. Therefore, we can leverage attribute relationships in known semantic models to hypothesize attribute relationships for new sources. Assume that before modeling the source s, we have already modeled two other museum sources $s_1(title, creationDate, name, birthDate, deathDate, type)$, which contain data of artworks in the Dallas Museum of Art (www.dma.org) and $s_2(name, copyright, materials, dimensions, imageUrl)$, which has data of artworks in The Metropolitan Museum of Art (www.metmuseum.org). Figure 3 illustrates the semantic models of these two data sources. In the next section, we explain how our approach uses these known models to learn a semantic model for the source s.

III. Learning Semantic Models

The problem of learning semantic models of data sources can be stated as follows. Let O be the domain ontology[1] and $\{sm(s_1), sm(s_2), \cdots, sm(s_n)\}$ is a set of known semantic models corresponding to the data sources $\{s_1, s_2, \cdots, s_n\}$. Given sample data from a new source $s(a_1, a_2, \cdots, a_n)$, our goal is to automatically compute a semantic model $sm(s)$ that captures the intended meaning of the source s.

Our approach to learn a semantic model for a new source has four steps: (1) Using sample data from the new source,

[1] O can be a set of ontologies. In our example, O consists of the EDM, AAC, SKOS, Dublin Core, FOAF, ORE, and ElementsGr2 ontologies.

learn the semantic types of the source (2) Construct a graph with the known source models, augmented with paths connecting the the learned semantic types in the domain ontology. (3) Compute the candidate mappings from the semantic types to the nodes of the graph. (4) Finally, build a candidate semantic model for each mapping, and rank the candidate models.

This work has two major contributions compared to our previous work on learning semantic models of data sources [8]. First, we consider uncertainty in learning the semantic types: we assume that for each source attribute we are given a set of candidate semantic types along with their confidence values rather than a single, correct semantic type. The second and main contribution of this paper is a search algorithm to find the good candidate mappings between the semantic types and the nodes of the graph. The algorithm scores the mappings as we map the semantic types to the nodes of the graph and eliminates the mappings that are less likely to generate good semantic models later. This algorithm enables our approach to handle both the uncertainty of semantic types and sources with a large number of attributes. Next we describe the overall approach, focusing on new parts and only providing high level descriptions of what is common to our previous work.

A. Learning Semantic Types of Source Attributes

The first step to model a source, which follows our previous work, is to recognize the semantic types of its data. The objective of this step is mapping the attributes to the concepts of the domain ontology. We formally define a semantic type to be either an ontology class $\langle class\_uri \rangle$ or a pair consisting of a data property and its domain class $\langle class\_uri, property\_uri \rangle$. We assign a class to attributes whose values are URIs and assign a domain/data property pair to attributes containing literal values. For example, the semantic types of the attributes *image* and *type* in s are respectively $\langle edm{:}WebResource \rangle$ and $\langle skos{:}Concept, skos{:}prefLabel \rangle$.

To learn semantic types, our algorithm uses a supervised machine learning technique [9] based on Conditional Random Fields (CRF) [11] with features extracted from the attribute names and sample data from the new source. A CRF model is useful for this problem because it can handle large numbers of features and learn from a small number of examples.

Once we apply this method, it generates a set of candidate semantic types for each source attribute, each with a confidence value. Our algorithm then selects the top k semantic types for each attribute as an input to the next step of the process. Thus, the output of the labeling step for $s(a_1, a_2, \cdots, a_n)$ is $T = \{(t_{11}^{p_{11}}, \cdots, t_{1k}^{p_{1k}}), \cdots, (t_{n1}^{p_{n1}}, \cdots, t_{nk}^{p_{nk}})\}$, where in $t_{ij}^{p_{ij}}$, t_{ij} is the jth semantic type learned for the attribute a_i and p_{ij} is the associated confidence value which is a decimal value between 0 and 1. Considering $k = 2$, we will have the following output after labeling the attributes of the source $s(title, label, image, type, artist)$:

$$\begin{aligned}
&title \rightarrow (\langle aac{:}CulturalHeritageObject, dcterms{:}title \rangle^{0.19},\\
&\quad\quad \langle aac{:}CulturalHeritageObject, rdfs{:}label \rangle^{0.08})\\
&label \rightarrow (\langle aac{:}CulturalHeritageObject, dcterms{:}description \rangle^{0.7},\\
&\quad\quad \langle aac{:}Person, ElementsGr2{:}note \rangle^{0.03})\\
&image \rightarrow (\langle edm{:}WebResource \rangle^{0.58}, \langle foaf{:}Document \rangle^{0.41})\\
&type \rightarrow (\langle skos{:}Concept, skos{:}prefLabel \rangle^{0.82},\\
&\quad\quad \langle skos{:}Concept, rdfs{:}label \rangle^{0.15})\\
&artist \rightarrow (\langle foaf{:}Person, foaf{:}name \rangle^{0.27},\\
&\quad\quad \langle aac{:}Person, ElementsGr2{:}nameOfThePerson \rangle^{0.19})
\end{aligned}$$

As we can see in the output, the machine learning method prefers $\langle foaf{:}Person, foaf{:}name \rangle$ for the semantic type of the attribute *artist*, while according to the correct model (Figure 2), $\langle aac{:}Person, ElementsGr2{:}nameOfThePerson \rangle$ is the correct semantic type. We will show later how our approach recovers the correct semantic type by considering coherence of structure in computing the semantic models.

B. Building A Graph from Known Semantic Models, Semantic Types, and Domain Ontology

So far, we have annotated the attributes of s with semantic types. To build a complete semantic model we still need to determine the relationships between the attributes. We leverage the knowledge of the known semantic models to discover the most popular and coherent patterns connecting the semantic types. The central component of our method is a directed weighted graph G built on top of the known semantic models and expanded using the semantic types T and the domain ontology O. The algorithm we use to construct G is described in detail in our past work [8] and here we only provide a brief explanation.

Similar to a semantic model, G contains both class nodes and data nodes and links. The links correspond to properties in O and there are weights on the links. Constructing the graph has three parts: adding the known semantic models ($sm(s_1)$ and $sm(s_2)$ in our example); adding the semantic types; and expanding the graph using the domain ontology O (in our scenario, O is a set of ontologies including EDM, AAC, SKOS, Dublin Core Metadata Terms, FOAF, ORE, and ElementsGr2).

Adding Known Semantic Models: We add each known semantic model as a new component to G if it is not a subgraph of already added components. In our example, at the end of this part, G will consist of two disconnected subgraphs corresponding to $sm(s_1)$ and $sm(s_2)$. These components are depicted using a gray background in Figure 4.

Adding Semantic Types: As mentioned before, we have two kinds of semantic types: $\langle class\_uri \rangle$ and $\langle class\_uri, property\_uri \rangle$. For each learned semantic type t, we search the graph to see whether G includes a *match* for t. We say (u, v, e) is a match for $t = \langle class\_uri \rangle$ if u is a data node, v is a class node with the label $class\_uri$, and e is a link from u to v with the label $rdf{:}type$. For example, in Figure 4, $(n_{25}, rdf{:}type, n_{18})$ is a match for the semantic type $\langle edm{:}WebResource \rangle$. We say (u, v, e) is a match for $t = \langle class\_uri, property\_uri \rangle$ if u is a class node with the label $class\_uri$, v is a data node, and e is a link from u to v with the label $property\_uri$. For example, in Figure 4, $(n_4, skos{:}prefLabel, n_{11})$ is a match for the semantic type

$\langle skos\colon Concept, skos\colon prefLabel\rangle$. We say $t = \langle class\_uri\rangle$ or $t = \langle class\_uri, property\_uri\rangle$ has a *partial match* in G when we cannot find a full match for t but there is a class node in G whose label matches $class\_uri$. For instance, the semantic type $\langle skos\colon Concept, rdfs\colon label\rangle$ only has a partial match in G (n_4), because G does not contain the link $rdfs\colon label$ after adding the known models (when G only includes the gray components).

For each semantic type t learned in the labeling step, we add the necessary nodes and links to G to create a match or complete existing partial matches. For example, for $title \rightarrow \langle aac\colon CulturalHeritageObject, dcterms\colon title\rangle$, we do not need to change G because the graph contains two matches: $(n_1, n_5, dcterms\colon title)$ and $(n_{17}, n_{24}, dcterms\colon title)$. For $type \rightarrow \langle skos\colon Concept, rdfs\colon label\rangle$, we have only one partial match (n_4), thus, we add one data node (n_{13}) and one link ($rdfs\colon label$) from n_4 to n_{13} in order to complete the existing partial match. For $image \rightarrow \langle foaf\colon Document\rangle$, there is neither a match nor a partial match. We add a class node (n_{15}), a data node (n_{19}), and a link ($rdf\colon type$) from n_{19} to n_{15} to create a match. The nodes and the links that are added in this step are shown with the blue color in Figure 4.

Adding Paths from the Ontology: We use the domain ontology to find all the paths that relate the current class nodes in G. We do this only for class nodes that do not belong to the same pattern. The goal is to connect class nodes of G using the direct paths or the paths inferred through the subclass hierarchy in O. For instance, in Figure 4, there is the link $edm\colon aggregatedCHO$ from n_{14} to n_1. We add this link because the object property $edm\colon aggregatedCHO$ is defined with $ore\colon Aggregation$ as domain and $edm\colon ProvidedCHO$ as range, and $edm\colon EuropeanaAggregation$ is a subclass of $ore\colon Aggregation$ and $aac\colon CulturalHeritageObject$ is a subclass of $edm\colon ProvidedCHO$.

Assigning weights to the links of the graph is important in our algorithm. We assign a very low weight ϵ to all the links inside a component (black links in Figure 4). We assign a high weight to all other links (blue and green links in Figure 4). The intuition behind this decision is to produce more coherent models later when we are generating candidate semantic models for s. For the links that are outside a component, we also take into account the link popularity. We use a simple counting mechanism to assign lower weights (but still very high compared to ϵ) to the links that appear more frequently in the set of known semantic models. For example, the link $dcterms\colon creator$ from n_{17} to n_3 has a lower weight compared to the link $dcterms\colon sitter$ from n_{17} to n_3 because $dcterms\colon creator$ has appeared in one of the semantic models ($sm(s_1)$), but $dcterms\colon sitter$ has not been seen before.

C. Mapping Semantic Types to the Graph

We use the graph built in the previous step to find the relationships between the source attributes. First, we map the source attributes to a subset of the nodes of the graph. Then, we compute the minimal tree that connects those nodes (Section III-D). To map the attributes of s to the nodes of G, we search G to find the matches for the semantic types assigned to the attributes. For example, the attribute $type$ in s maps to the nodes n_4 and n_{11} because $(n_4, n_{11}, skos\colon prefLabel)$ is a match for $\langle skos\colon Concept, skos\colon prefLabel\rangle$, which is a semantic type learned for the attribute $type$.

Since it is possible that a semantic type has more than one match in G, more than one mapping m might exist from the source attributes to the nodes. In our past work [8], we generated all the possible mappings and then selected the candidate mappings after sorting them based on a metric called *coherence*. However, generating all the mappings is not feasible in cases where we have a data source with many attributes and learned semantic types have many matches in the graph. The problem is worse when we have more than one semantic type for each attribute. Suppose that we are modeling the source s consisting of n attributes and we have learned k semantic type for each attribute. If there are r matches for each semantic type, we will have $(k * r)^n$ mappings from $attributes(s)$ to $nodes(G)$.

In this section, we present an algorithm that scores a mapping as we map the attributes to the nodes of the graph and removes the low score mappings after mapping each attribute. In addition to coherence, we also take into account the confidence values of the semantic types and the size of the mappings in the scoring function. The inputs to the algorithm are the learned semantic types $T = \{(t_{11}^{p_{11}}, \cdots, t_{1k}^{p_{1k}}), \cdots, (t_{n1}^{p_{n1}}, \cdots, t_{nk}^{p_{nk}})\}$ for the attributes $\{a_1, \cdots, a_n\}$ and the graph G, and the output is a set of candidate mappings m from $attributes(s)$ to a subset of $nodes(G)$. Algorithm 1 shows the steps of our approach. The key idea is that instead of generating all the mappings and then sorting them, we score the partial mappings after processing each attribute and prune the mappings with lower scores. In other words, we do not wait until all the attributes are mapped. Instead, as soon as we find the matches for the semantic types of an attribute, we rank the partial mappings and keep the better ones. In this way, the number of candidate mappings never exceeds a fixed size after mapping each attribute.

The heart of the algorithm is the scoring function we use to rank the partial mappings (line 24 in in Algorithm 1). We compute three functions for each mapping m: $confidence(m)$, $coherence(m)$, and $sizeReduction(m)$. Then, we calculate $score(m)$ as the arithmetic mean of these three values. We explain these functions using an example. Suppose that the maximum number of the mappings we expand in each step is 8 ($max\_mappings$ in line 3). After mapping the second attribute of the source s ($label$), we will have $mappings = \{$

$m_1: \{title, label\} \rightarrow \{n_1, n_5, n_7\}$,
$m_2: \{title, label\} \rightarrow \{n_1, n_5, n_3, n_{12}\}$,
$m_3: \{title, label\} \rightarrow \{n_1, n_5, n_{17}, n_{27}\}$
$m_4: \{title, label\} \rightarrow \{n_1, n_6, n_7\}$,
$m_5: \{title, label\} \rightarrow \{n_1, n_6, n_3, n_{12}\}$,
$m_6: \{title, label\} \rightarrow \{n_1, n_6, n_{17}, n_{27}\}$,
$m_7: \{title, label\} \rightarrow \{n_{17}, n_{24}, n_{27}\}$,
$m_8: \{title, label\} \rightarrow \{n_{17}, n_{24}, n_3, n_{12}\}$,
$m_9: \{title, label\} \rightarrow \{n_{17}, n_{24}, n_1, n_7\}$,

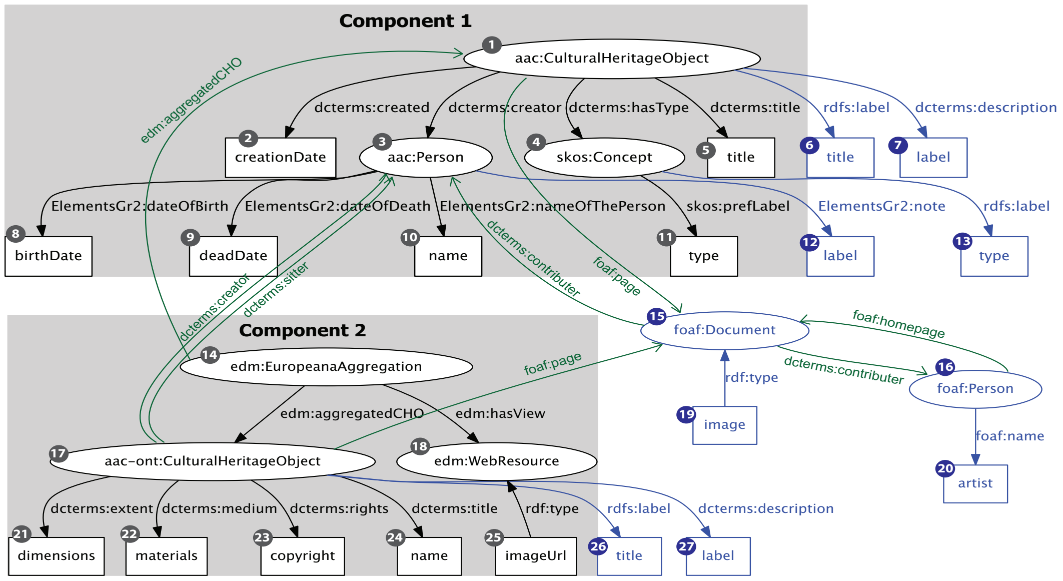


Fig. 4. The graph constructed from the known semantic models $sm(s_1), sm(s_2)$, semantic types, and a set of ontologies including EDM, AAC, SKOS, Dublin Core Metadata Terms, FOAF, ORE, and ElementsGr2. For legibility, only a few of all the possible paths between the class nodes are shown.

Algorithm 1 *GenerateCandidateMappings*

```
Input: $G(V, E)$,
       $attributes(s) = \{a_1, \cdots, a_n\}$
       $T = \{(t_{11}^{p_{11}}, \cdots, t_{1k}^{p_{1k}}), \cdots, (t_{n1}^{p_{n1}}, \cdots, t_{nk}^{p_{nk}})\}$
Output: a set of candidate mappings $m$ from $attributes(s)$ to $S \subset V$

$mappings \leftarrow \{\}$
$candidates \leftarrow \{\}$
$max\_mappings \leftarrow$ maximum number of mappings to expand
$num\_of\_candidates \leftarrow$ number of candidate mappings
for each $a_i \in attributes(s)$ do
    for each $t_{ij}^{p_{ij}} \in (t_{i1}^{p_{i1}}, \cdots, t_{ik}^{p_{ik}})$ do
        $matches \leftarrow$ all the $(u, v, e)$ in $G$ matching $t_{ij}$
        if $mappings = \{\}$ then
            for each $(u, v, e) \in matches$ do
                $m \leftarrow (\{a_i\} \rightarrow \{u, v\})$
                $mappings \leftarrow mappings \cup m$
            end for
        else
            for each $m : X \rightarrow Y \in mappings$ do
                for each $(u, v, e) \in matches$ do
                    $m' \leftarrow (X \cup \{a_i\} \rightarrow Y \cup \{u, v\})$
                    $mappings \leftarrow mappings \cup m'$
                end for
                remove $m$ from $mappings$
            end for
        end if
    end for
    if $|mappings| > max\_mappings$ then
        compute $score(m)$ for each $m \in mappings$
        sort items in $mappings$ based on their score
        keep top $max\_mappings$ mappings and remove others
    end if
end for
$candidates \leftarrow$ top $num\_of\_candidates$ items from $mappings$
return $candidates$
```

$$m_{10} : \{title, label\} \rightarrow \{n_{17}, n_{26}, n_{27}\},$$
$$m_{11} : \{title, label\} \rightarrow \{n_{17}, n_{26}, n_3, n_{12}\},$$
$$m_{12} : \{title, label\} \rightarrow \{n_{17}, n_{26}, n_1, n_7\} \ \}$$

There are 4 matches for the attribute *title*: (n_1, n_5) and (n_{17}, n_{24}) for the semantic type $\langle aac\colon CulturalHeritageObject, dcterms\colon title\rangle$ and (n_1, n_6) and (n_{17}, n_{26}) for the semantic type $\langle aac\colon CulturalHeritageObject, rdfs\colon label\rangle$; and 3 matches for the attribute *label*: (n_1, n_7) and (n_{17}, n_{27}) for the semantic type $\langle aac\colon CulturalHeritageObject, dcterms\colon description\rangle$ and (n_3, n_{12}) for the semantic type $\langle aac\colon Person, ElementsGr2\colon note\rangle$. This yields $4 * 3 = 12$ different mappings. Since $max\_mappings = 8$, we have to eliminate four of these mappings. Now, we describe how the algorithm ranks the mappings.

Confidence(m) is the arithmetic mean of the confidence values associated with a mapping. For example, m_1 is consisting of the matches for the semantic types $\langle aac\colon CulturalHeritageObject, dcterms\colon title\rangle^{0.19}$ and $\langle aac\colon CulturalHeritageObject, dcterms\colon description\rangle^{0.7}$. Thus, $confidence(m_1) = 0.445$.

Coherence(m) measures the largest number of nodes in a mapping that belong to the same component. For instance, $coherence(m_1) = 0.66$ because two nodes out of three nodes in m_1 ($n1$ and n_5) are from component 1, and $coherence(m_3) = 0.5$ because 2 is the largest number of nodes in m_3 that are from the same component. The goal of defining the coherence is to give more priority to the models containing larger segments from the known patterns.

Size Reduction: Since we prefer concise models, we seek mappings with fewer nodes. If a mapping has n attributes, the smallest possible size is $l = n + 1$ (when all the attributes map to the same class node, e.g., m_1) and the largest is $u = 2*n$ (when all the attributes map to different class nodes, e.g., m_2). Thus, the possible size reduction in a mapping is $u - l$. We define $sizeReduction(m) = \frac{u - size(m)}{u - l + 1}$ as how much the size of a mapping is reduced compared to the possible size reduction. For example, $sizeReduction(m_1) = 0.5$ and $sizeReduction(m_2) = 0$.

Score(m) is the arithmetic mean of $confidence(m)$, $coherence(m)$, and $sizeReduction(m)$. All these functions have a value in $[0, 1]$. Here are the scores of the 12 mappings we mentioned before: $score(m_1) = 0.535$, $score(m_2) = 0.286$, $score(m_3) = 0.315$, $score(m_4) = 0.406$, $score(m_5) = 0.185$, $score(m_6) = 0.213$, $score(m_7) = 0.535$, $score(m_8) = 0.203$, $score(m_9) = 0.315$, $score(m_{10}) = 0.380$, $score(m_{11}) = 0.101$, $score(m_{12}) = 0.213$. Therefore, m_5, m_6, m_8, and m_{11} will be removed from the $mappings$ (line 26), and the algorithm continues to the next iteration, which is mapping the next attribute of the source s ($image$) to the graph. At the end, we will have maximum $max\_mappings$ mappings, each of them will include all the attributes. We sort these mappings based on their score and consider the top $num\_of\_candidates$ mappings as the candidates (line 29). In the next part of the approach, we compute a semantic model for each of these candidate mappings.

D. Generating and Ranking Semantic Models

Once we generated candidate mappings from the source attributes to the nodes of the graph, we compute a semantic model for each mapping and then rank the resulting models. To compute a semantic model for a mapping m, we find a minimum-cost tree in G that connects the nodes of m. This problem is known as the Steiner Tree problem [12], [13]. Given an edge-weighted graph and a subset of the vertices, called Steiner nodes, the goal is to find the minimum-weight tree that spans all the Steiner nodes. The general Steiner tree problem is NP-complete, thus, we use an approximation algorithm [13] to compute the tree for each mapping m. The inputs to the algorithm are the graph G and the nodes of m (as Steiner nodes) and the output is a tree that we consider it as a candidate semantic model for s. For example, for the mapping $m : \{title, label, image, type, artist\} \rightarrow \{n_1, n_5, n_7, n_{18}, n_{25}, n_4, n_{11}, n_3, n_{10}\}$, the resulting Steiner tree is the correct semantic model of s shown in Figure 2. The final step of our approach is ranking the semantic models generated for the candidate mappings where we rank them based on their cost (sum of the weights of the links). The output is a ranked list of plausible semantic models for s.

It is important to note that considering coherence of patterns in scoring the mappings and also ranking the final semantic models enables our approach to compute the correct semantic model in many cases where the first-ranked semantic types are not the correct ones. For example, the mapping $m : \{title, label, image, type, artist\} \rightarrow \{n_1, n_5, n_7, n_{18}, n_{25}, n_4, n_{11}, n_3, n_{10}\}$ that maps the attribute *artist* to n_3 and n_{10} using the semantic type $\langle aac{:}Person, ElementsGr2{:}nameOfThePerson\rangle$ will be scored higher than the mapping $m' : \{title, label, image, type, artist\} \rightarrow \{n_1, n_5, n_7, n_{18}, n_{25}, n_4, n_{11}, n_{16}, n_{20}\}$ that maps this attribute to n_{16} and n_{20} using the semantic type $\langle foaf{:}Person, foaf{:}name\rangle$. The mapping m has lower confidence value than m', but it will be scored higher because its coherence value is higher. The model computed from the mapping m will also be ranked higher than the model computed from m', because it includes more links from known patterns, thus resulting in a lower cost tree.

IV. Evaluation

We evaluated our approach on a dataset of 29 museum data sources containing data from different art museums in the US. The total number of attributes for this dataset was 332 (on average 11 attributes per source). We applied our approach on this dataset to find the candidate semantic models for each source and then compared the best suggested models (the first ranked models) with models created manually by domain experts. The sources were modeled using the EDM, AAC, SKOS, Dublin Core Metadata Terms, FOAF, ORE, and ElementsGr2 ontologies.

We compare semantic models using *precision* and *recall*:

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

where for a semantic model sm, $rel(sm)$ is the set of triples $\langle u, e, v\rangle$ in which e is a directed link from u to v in sm. For example, for the semantic model in Figure 2, $rel(sm) = \{\langle edm{:}EuropeanaAggregation, edm{:}aggregatedCHO, aac{:}CulturalHeritageObject\rangle, \langle edm{:}EuropeanaAggregation, edm{:}hasView, edm{:}WebResource\rangle, \langle aac{:}CulturalHeritageObject, dcterms{:}creator, aac{:}Person\rangle, \cdots\}$.

Assume that the correct semantic model of the source s is sm and the semantic model learned by our approach is sm'. We can prove that if all the nodes in sm have unique labels and all the nodes in sm' also have unique labels, $rel(sm) = rel(sm')$ ensures that sm and sm' are equivalent. However, if the semantic models have more than one instance of an ontology class, we will have nodes with the same label. In this case, $rel(sm) = rel(sm')$ does not guarantee $sm = sm'$. Many sources in our dataset have models that include two instances of an ontology class. Therefore, before measuring the precision and recall, we number the nodes having the same label to assign them a unique label. For example, if we have two nodes labeled with the class URI $aac{:}Person$ in a model, we change the labels to $aac{:}Person1$ and $aac{:}Person2$.

We ran three experiments: (1) We labeled each source attribute with the correct semantic type. The goal was to see how well our approach learns the attribute relationships having the correct semantic types. (2) We applied the CRF technique to learn the semantic types and then only considered the semantic type with the highest confidence value ($k = 1$). (3) We used CRF for labeling but instead of the top semantic type, we considered the top four learned semantic types as the candidate semantic types ($k = 4$).

In all experiments, we applied our method to learn a semantic model for the source s_i, $sm(s_i)$, assuming that the semantic models of the other sources are known. To investigate how the number of the known models influences the results, we used variable number of known models as input. Suppose that M_j is the set of known semantic models including j

models. Running the experiment with M_0 means that we do not use any knowledge other than the domain ontology and running it with M_{28} means that the semantic models of all other sources are known. For example, for s_1, we ran the code 29 times using $M_0 = \{\}, M_1 = \{sm(s_2)\}, M_2 = \{sm(s_2), sm(s_3)\}, \cdots, M_{28} = \{sm(s_2), \cdots, sm(s_{29})\}$. We used $max\_mappings = 100$ in our mapping algorithm and then only considered the top 10 mappings as the candidate mappings ($num\_of\_candidates = 10$). Figure 5 shows the average precision and recall of all the learned semantic models $(sm'(s_1), ..., sm'(s_{29}))$ for all the three experiments.

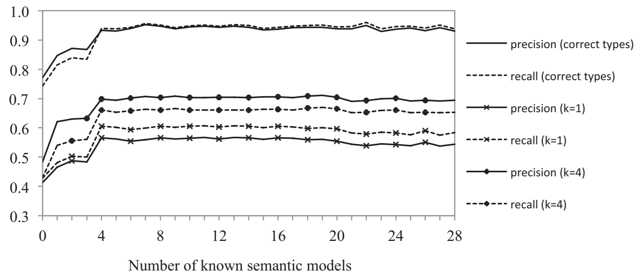


Fig. 5. Average precision and recall for the learned semantic models when: the correct semantic type is known; only the top CRF suggested type is considered (k=1); and the top four suggested types are considered (k=4).

The results show that the precision and recall increase significantly even with a few known semantic models. When all the attributes have the correct type, our approach can learn relationships with high precision and recall. Obviously, learning the correct semantic type for all the attributes is not possible. As we can see in the graph, when we use the CRF technique to learn the semantic types, the results are not as good as when we know the correct types. In the case where we only consider the top semantic type ($k = 1$), the average precision and recall for most of the sources fall in $[0.5, 0.6]$. The reason is that for 38% of the attributes, the top learned semantic type was not the correct semantic type. Nonetheless, when we consider more than one semantic type ($k = 4$), we can learn more accurate models. Given that still the correct semantic type of 13% of the attributes was not among the top four suggested semantic types, the results demonstrate that our algorithm performs fairly well when there is uncertainty is learning the semantic types.

To evaluate the running time of the approach, we measured the time of running the algorithm (excluding the labeling step and starting from building the graph) on a single machine with a Mac OS X operating system and a 2.3 GHz Intel Core i7 CPU. Figure 6 shows the average time (in seconds) of learning the semantic models. We believe that the overall time of the process can be further reduced by using parallel programming and some optimizations in the implementation. For example, the graph can be built incrementally. When a new known model is added, we do not need to create the graph from beginning. We just need to add a new component to the existing graph and update the links.

We compared our new approach with the previous work [8]

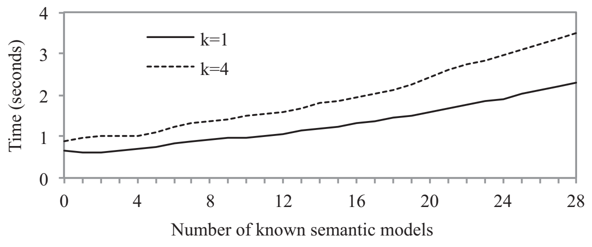


Fig. 6. Average semantic model learning time, for $k = 1$ and $k = 4$.

to show that the new approach scales better. In this experiment, we used $k = 1$ because the previous work was not able to take into account more than one semantic type for each source attribute. For each source, we applied both approaches to learn a semantic model assuming that the semantic models of all other sources are known (running both approaches with M_{28}).

The previous approach could only learn a semantic model for 2 out of 29 sources in the timeout of 1 hour. These two sources only had 4 attributes. The reason is that the number of mappings for sources that have more than a few attributes is very large and it takes a long time to generate all of them. For example, in modeling s_{16} with only 5 attributes, the number of mappings was 16,633,298.

V. Related Work

One of the basic approaches to data integration is building a global schema and then defining every source as a view over the global schema [1]. The core of this approach are source descriptions that model the semantics of data sources as logical mappings between the sources and the global schema. There has been much work to automate the task of generating these mappings. Some work [14], [15] only finds correspondences between elements of the source and global schemas. This is analogous to the semantic labeling step in our work, where we use a machine learning technique [9] to learn candidate semantic types for a source attribute. Every semantic type maps an attribute to an element in the domain ontology (a class or property in the domain ontology). Further work [16]–[19] generates more complex mappings that express relationships. However, they do not exploit the knowledge of previously modeled sources as we do.

In the Semantic Web, the global model is an ontology that defines the concepts and relationships within a domain. There are many studies on mapping data sources to ontologies. Most of this work [2]–[6] focus on semantic annotation, but is limited in learning relationships. Carman and Knoblock [20] use known source descriptions to generate a mapping for an unknown target source. However, their approach was limited to learning descriptions that were conjunctive *combinations* of known source descriptions.

Our previous work on source modeling in Karma [7], [10], [21] maps a source to an ontology *interactively*. The system uses learned semantic types and a Steiner tree algorithm to propose models to the user, who can correct them as needed. Although Karma remembers new semantic type labels assigned by the user, it does not learn from the structure of previously modeled sources. This prompted our recent work [8] to exploit the known semantic models to learn a semantic model for a new unknown source. However, this approach assumed that each source attribute was labeled with the correct semantic type and failed to scale to sources with a large number of attributes. The work we have presented addresses these limitations. Our algorithm considers multiple possible semantic types for each attribute and searches the space of semantic models efficiently.

VI. Discussion

We presented a scalable approach to learn semantic models of structured data sources as a mapping from the source to a domain ontology. The core idea is to exploit the knowledge of previously learned semantic models to hypothesize a plausible semantic model for a new source. We extend on previous work [8] by allowing for uncertainty in learning semantic types and by scaling to sources with many attributes. Our evaluation shows that the new approach can learn semantic models for data sources that the previous approach was not able to handle.

The first step in learning semantic models is learning the semantic types in which we label each source attribute with a class or property from the ontology. Here, we generalize the previous work to deal with uncertainty in labeling the source attributes. The output of the labeling step is a set of candidate semantic types and their confidence values rather than one fixed semantic type. This new extension is very important because machine learning techniques often cannot distinguish the types of the source attributes that have similar data values, e.g., *birthDate* and *deathDate*.

Once we learned the semantic types, we create a graph from known semantic models and augment it by adding the nodes and the links corresponding to the semantic types and adding the paths inferred from the ontology. The next step is mapping the source attributes to the nodes of the graph where we introduce a new algorithm that enables us to do the mapping even when the source has many attributes. In the new algorithm, after processing each source attribute, we prune the existing mappings by scoring them and removing the ones having lower scores. The proposed scoring function not only contributes to the scalability of our method, but also increases the accuracy of the learned models.

The final part of the approach is computing the minimal tree that connects the nodes of the candidate mappings. This step might be computationally inefficient if we have a very large graph (e.g, when we have a large number of known models) or/and the number of candidate mappings is very large. Reducing the size of the graph is part of our future work where we want to investigate the idea of constructing a more compact graph by consolidating the overlapping segments of the known semantic models. Regarding the number of candidate mappings, our empirical evaluation showed that the algorithm works very well even with a few number of candidates (10 in our experiment).

Another direction of future work is to leverage the large amount of data available in the Linked Open Data (LOD) cloud to improve the quality of the automatically generated models. LOD contains lots of resources connected to each other using the relationships defined by different domain ontologies. Performing record linkage between some of the source data and the entities in the LOD, allows us to exploit existing links between those entities to improve the accuracy of our automatically-generated source models.

Acknowledgment

This research is based upon work supported in part by the Intelligence Advanced Research Projects Activity (IARPA) via Air Force Research Laboratory (AFRL) contract number FA8650-10-C-7058 and in part by the National Science Foundation under award number IIS-1117913. 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 IARPA, AFRL, NSF, or the U.S. Government.

References

[1] A. Doan, A. Halevy, and Z. Ives, *Principles of Data Integration*. Morgan Kauffman, 2012.

[2] V. Mulwad, T. Finin, and A. Joshi, “Semantic Message Passing for Generating Linked Data from Tables,” in *ISWC*, 2013.

[3] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu, “Recovering Semantics of Tables on the Web,” *Proc. VLDB Endow.*, vol. 4, no. 9, pp. 528–538, 2011.

[4] G. Limaye, S. Sarawagi, and S. Chakrabarti, “Annotating and Searching Web Tables Using Entities, Types and Relationships,” *PVLDB*, vol. 3, no. 1, pp. 1338–1347, 2010.

[5] A. P. Sheth, K. Gomadam, and A. Ranabahu, “Semantics Enhanced Services: METEOR-S, SAWSDL and SA-REST,” *IEEE Data Eng. Bulletin*, vol. 31, no. 3, pp. 8–12, 2008.

[6] V. Saquicela, L. M. V. Blázquez, and Ó. Corcho, “Lightweight Semantic Annotation of Geospatial RESTful Services,” in *ESWC*, 2011.

[7] C. Knoblock, P. Szekely, J. L. Ambite, A. Goel, S. Gupta, K. Lerman, M. Muslea, M. Taheriyan, and P. Mallick, “Semi-Automatically Mapping Structured Sources into the Semantic Web,” in *ESWC*, 2012.

[8] M. Taheriyan, C. A. Knoblock, P. Szekely, and J. L. Ambite, “A Graph-based Approach to Learn Semantic Descriptions of Data Sources,” in *ISWC*, 2013.

[9] A. Goel, C. A. Knoblock, and K. Lerman, “Exploiting Structure within Data for Accurate Labeling Using Conditional Random Fields,” in *Procs. 14th International Conference on Artificial Intelligence (ICAI)*, 2012.

[10] P. Szekely, C. A. Knoblock, F. Yang, X. Zhu, E. Fink, R. Allen, and G. Goodlander, “Connecting the Smithsonian American Art Museum to the Linked Data Cloud,” in *ESWC*, 2013.

[11] J. Lafferty, A. McCallum, and F. Pereira, “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data,” in *Procs. 18th International Conference on Machine Learning*, 2001.

[12] P. Winter, “Steiner Problem in Networks - A survey,” *Networks*, vol. 17, pp. 129–167, 1987.

[13] L. Kou, G. Markowsky, and L. Berman, “A Fast Algorithm for Steiner Trees,” *Acta Informatica*, vol. 15, pp. 141–145, 1981.

[14] E. Rahm and P. Bernstein, “A Survey of Approaches to Automatic Schema Matching,” *VLDB Journal*, vol. 10, no. 4, 2001.

[15] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos, “iMAP: Discovering Complex Semantic Matches between Database Schemas,” in *SIGMOD*, 2004.

[16] Z. Bellahsene, A. Bonifati, and E. Rahm, *Schema Matching and Mapping*, 1st ed. Springer, 2011.

[17] R. Fagin, L. M. Haas, M. A. Hernndez, R. J. Miller, L. Popa, and Y. Velegrakis, “Clio: Schema Mapping Creation and Data Exchange,” in *Conceptual Modeling: Foundations and Applications*, 2009.

[18] B. Alexe, B. ten Cate, P. G. Kolaitis, and W. C. Tan, “Designing and Refining Schema Mappings via Data Examples,” in *SIGMOD*, 2011.

[19] Y. An, A. Borgida, R. J. Miller, and J. Mylopoulos, “A Semantic Approach to Discovering Schema Mapping Expressions,” in *Procs. of the 23rd International Conference on Data Engineering (ICDE)*, 2007.

[20] M. J. Carman and C. A. Knoblock, “Learning Semantic Definitions of Online Information Sources,” *Journal of Artificial Intelligence Research*, vol. 30, no. 1, pp. 1–50, Sep. 2007.

[21] M. Taheriyan, C. A. Knoblock, P. Szekely, and J. L. Ambite, “Rapidly Integrating Services into the Linked Data Cloud,” in *ISWC*, 2012.