A Graph-Based Approach to Learn Semantic Descriptions of Data Sources

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

University of Southern California
Information Sciences Institute and Department of Computer Science
{mohsen,knoblock,pszekely,ambite}@isi.edu

Abstract. Semantic models of data sources and services provide support to automate many tasks such as source discovery, data integration, and service composition, but writing these semantic descriptions by hand is a tedious and time-consuming task. Most of the related work focuses on automatic annotation with classes or properties of source attributes or input and output parameters. However, constructing a source model that includes the relationships between the attributes in addition to their semantic types remains a largely unsolved problem. In this paper, we present a graph-based approach to hypothesize a rich semantic description of a new target source from a set of known sources that have been modeled over the same domain ontology. We exploit the domain ontology and the known source models to build a graph that represents the space of plausible source descriptions. Then, we compute the top k candidates and suggest to the user a ranked list of the semantic models for the new source. The approach takes into account user corrections to learn more accurate semantic descriptions of future data sources. Our evaluation shows that our method produces models that are twice as accurate than the models produced using a state of the art system that does not learn from prior models.

Keywords: semantic description, source modeling, source description, semantic model, Semantic Web.

1 Introduction

Today, information sources such as relational databases and Web services provide a vast amount of structured data. A common approach to integrate sources involves building a global model and constructing source descriptions that specify mappings between the sources and the global model [8]. In the traditional data integration approaches, source descriptions are specified as global-as-view (GAV) or local-as-view (LAV) descriptions. In the Semantic Web, what is meant by a source description is a semantic model describing the source in terms of the concepts and relationships defined by an ontology. This semantic model can be viewed as a graph with ontology classes as the nodes and ontology properties as the links between the nodes.

The first step in building a source description of a source is to determine the semantic types. That is, each source attribute is labeled with a class or a data property of the domain ontology. However, simply annotating the attributes is not sufficient. For example, consider a data table with two columns: *name*, which is mapped to the class $Person$, and *place*, which is mapped to the class $City$. Unless the relationship between the two columns is explicitly specified, we do not know whether the city is the birth place of the person or it is the place where she currently lives. A precise source description needs a second step that determines the relationships between attributes in terms of properties in the ontology.

Writing source descriptions by hand requires significant effort and expertise. Although desirable, generating these descriptions automatically is a challenging problem [1, 7, 10, 17, 20]. Most of the proposed approaches on the Semantic Web focus on the first step of the modeling process. In our previous work [13], we presented an algorithm to construct semantic models of data sources by computing a Steiner tree in a graph derived from the ontology and the semantic types. If the suggested tree is not the correct model of the data, the user interactively imposes constraints on the algorithm through a graphical user interface to build the correct model. However, the system does not learn the refinements done by the user and always suggests a random minimal tree as the initial model of the new sources.

In this work, we present algorithms to improve the quality of the automatically generated models by using the already modeled sources to learn the patterns that more likely represent the intended meaning of a new source. The insight of our approach is that different sources in the same domain often provide similar or overlapping data. Thus, it should be possible to exploit knowledge of previously modeled sources to learn descriptions for new sources. First, we construct a graph whose main components are subgraphs corresponding to the known source models. We use the domain ontology to infer the possible paths connecting the nodes across different subgraphs. This graph models the space of plausible source descriptions. Next, we label each source attribute with a semantic type and try to find a set of candidate mappings between these semantic types and the nodes in the graph. For each resulting set of the nodes, we compute the minimal tree that connects them and consider this tree as a plausible source model. Then, we score the models to prefer the ones formed with more coherent and frequent patterns. Finally, we generate a ranked list of possible semantic models. We can put users in the loop by allowing them to select the correct model or refine one of the suggested models. The correct model will be added to the graph as a new component yielding more accurate models in the future.

Our work provides a basis to learn the source descriptions of information sources. The main contribution of our work are the techniques to leverage attribute relationships in known source descriptions to hypothesize attribute relationships for new sources, and capturing them in source descriptions. Such source descriptions are beneficial to automate tasks such as source discovery, information integration, and service composition. They also make it possible to convert sources into RDF and publish them in the Linked Data cloud [23].

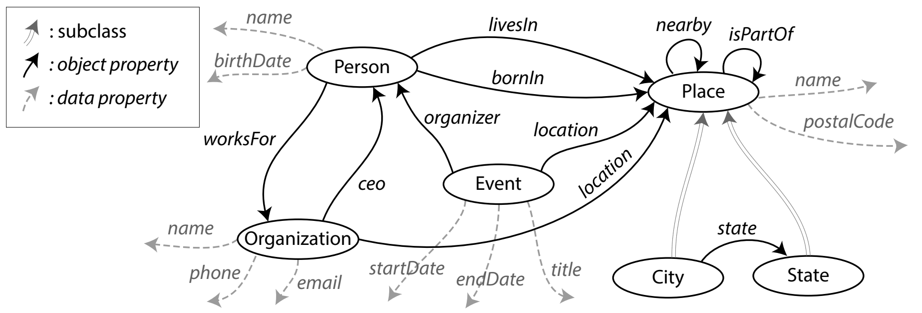

Fig. 1. The ontology that we use to model the sources in the example

2 Motivating Example

In this section, we explain the problem of learning source descriptions by giving a concrete example that will be used throughout the paper to illustrate our approach. In this example we have five data sources whose definitions are as follows:

$s_1 = personalInfo(name, birthdate, city, state, workplace)$
$s_2 = getCities(state, city)$
$s_3 = businessInfo(company, ceo, city, state)$
$s_4 = getEmployees(employer, employee)$
$s_5 = postalCodeLookup(zipcode, city, state)$

s_1 is a dataset providing information about people; s_2 is a Web service that takes as input a U.S. state and returns all the cities of that state; s_3 is a dataset providing information about businesses such as their name; s_4 is a list of U.S. companies and their employees; and s_5 is a Web service that returns the list of all the cities in a ZIP code. We use the ontology shown in Figure 1 to build a source description for each source. These descriptions are illustrated in Figure 2. Now, suppose that the first three sources (s_1, s_2, and s_3) have already been modeled and the other two (s_4 and s_5) are new sources not modeled yet. The goal is to automatically infer the source descriptions for s_4 and s_5 given the ontology and the models for s_1, s_2, and s_3.

Automatically building a source description of an unknown source is difficult. First, the system must map the source attributes to classes in the ontology. Considering source s_4 in Figure 2, attribute *employer* should be mapped to *name* of *Organization* and attribute *employee* should be mapped to *name* of *Person*. Next, the system needs to infer the relationships between the classes used to model the attributes. Our sample ontology has two links between *Person* and *Organization*, namely *worksFor* and *ceo*. The system needs to select *worksFor*, which captures the intended meaning of s_4. The problem is more complicated in cases when the relevant classes are not directly connected in the ontology and there exist multiple paths connecting them to each other.

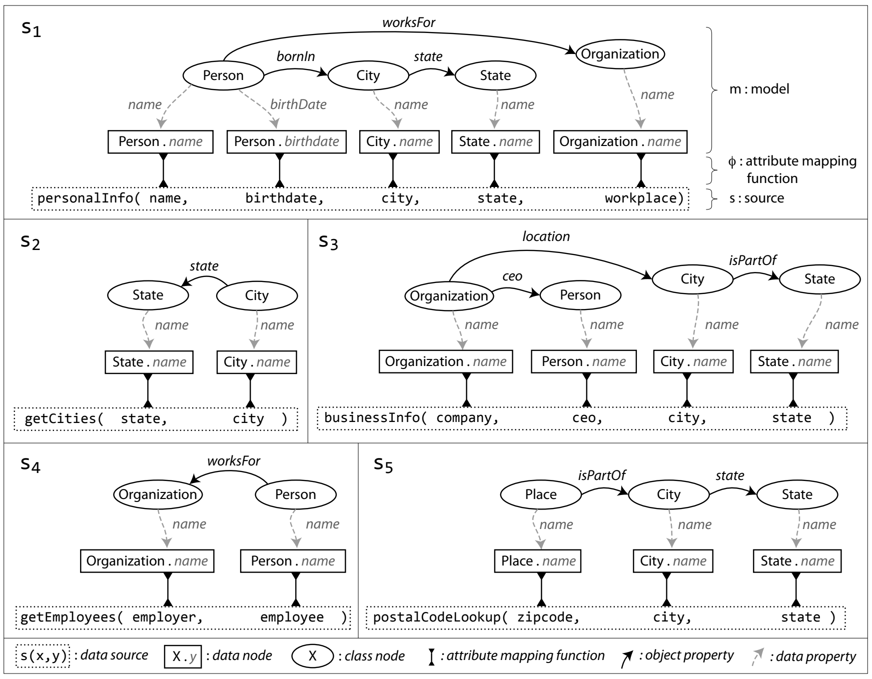

Fig. 2. Source descriptions of sample data sources according to the introduced ontology

In this work, we exploit already modeled sources to build semantic models that are more accurate in terms of the relationships between the source attributes. One of the metrics helping us to build our models is the link popularity, nevertheless, simply using link and node popularity would lead to myopic decisions that select nodes and links that appear frequently in other models without taking into account how these nodes are connected to other nodes in the given models. The main idea of our approach is taking into account the coherency of the patterns and this is much harder to do. Suppose that we have one model containing the link *organizer* between *Event* and *Person* and the link *location* between *Event* and *Place*. We also have several models including *Person* and *Place* (but not *Event*) connected by the relation *bornIn*. If the new source contains *Person*, *Place*, and *Event*, just using the link popularity yields to an incorrect model.

3 Problem Formulation

Having given the example above, we state the problem of learning source descriptions of sources formally.

A *source* is a n-ary relation $s(a_1, \cdots, a_n)$, with a set of attributes $\langle a_1, \cdots, a_n \rangle$, denoted as $Attributes(s)$.

A *semantic model m* is a directed graph containing two types of nodes, *class nodes* and *data nodes*. Class nodes (ovals in Figure 2) correspond to classes in the ontology and are labeled with class URIs (if v is a class node, $uri(v)$ is URI of the associated class). Data nodes (rectangles in Figure 2), correspond to the range of data properties and are labeled with a pair of URIs: one is the URI of a data property and the the other is the URI of one of the domains of that property e.g., $\langle Person, name\rangle$ or $\langle City, name\rangle$. The links in the graph correspond to ontology properties and are labeled with property URIs (if e is a link, $uri(e)$ represents the URI of the property). In general, a semantic model may contain multiple nodes and links labeled with the same URI.

An *attribute mapping function* ϕ:$Attributes(s) \rightarrow Nodes(m)$ is a mapping from the attributes of source s to a subset of the nodes in m. It can be a partial mapping, i.e, only some of the attributes are connected to the nodes in m.

A *source description* is a triple (s, m, ϕ) where s is a source, m is a semantic model, ϕ is an attribute mapping function that connects the source to the model, and m can be written as a conjunctive query over the predicates of the domain ontology O (in this work, we do not deal with source descriptions involving more complex constructs such as *aggregation*, *union*, or *negation*).

Figure 2 shows the source descriptions for our five example sources. In the figure, ϕ is represented using the inverted arrows symbols (↥) connecting attributes in the source to nodes in the model.

The problem of inferring source descriptions can be stated as follows. Let O be a domain ontology and $S = \{(s_1, m_1, \phi_1) \cdots, (s_k, m_k, \phi_k)\}$ a set of source descriptions. Given a new source $\hat{s}$, we want to compose a semantic model $\hat{m}$ and an attribute mapping function $\hat{\phi}$ such that $(\hat{s}, \hat{m}, \hat{\phi})$ is an *appropriate* source description. Clearly, many triples $(\hat{s}, \hat{m}_i, \hat{\phi}_i)$ are well-formed source descriptions, i.e., $\hat{m}_i$ and $\hat{\phi}_i$ are well defined, but only one or a few capture the intended meaning of the data contained in $\hat{s}$. Our goal is to automatically compute $(\hat{s}, \hat{m}, \hat{\phi})$ such that it minimizes the graph edit distance to a source description that the user would deem correct. We evaluate our approach by computing the graph edit distance from $(\hat{s}, \hat{m}, \hat{\phi})$ to a user-defined source description.

4 Learning Semantic Descriptions

The approach we take to learn the source description of a new source aims to characterize a source in terms of the concepts and properties in the domain ontology. In general, the ontology defines a large space (sometimes infinite) of possible source descriptions and without additional information, we do not know which one describes the source more precisely. The main idea here is that data sources in the same domain usually provide overlapping data. Therefore, we can leverage the knowledge of previously described sources to limit the search space and get some hints to hypothesize more plausible candidates.

Our approach has four steps. First, we construct a graph whose main components are the semantic models of the known source descriptions. We use the domain ontology to enrich the graph by adding the nodes and the links that connect these components. Second, we label the source attributes with semantic types. This step is not the focus of this paper and we use an existing technique [12] to annotate the attributes. Third, we find all possible mappings between the assigned semantic types and nodes of the graph and select the k most promising mappings. We compute a semantic model $\hat{m}$ and an attribute mapping function $\hat{\phi}$ for each mapping to construct the top k source descriptions $(\hat{s}, \hat{m}, \hat{\phi})$. Finally, we define metrics to rank the generated candidates.

4.1 Building a Graph from Known Semantic Models

The central component of our method is a directed weighted graph G built on top of the known semantic models m_i and expanded using the domain ontology O. Similar to the graph of semantic models, G contains both class nodes and data nodes and links corresponding to properties in O. However, the links in G are weighted and they also store a list of the model identifiers, called $support\_models$. Algorithm 1 explains how we create G from the known models.

Before we describe the algorithm, we need to define the functions $closure(c)$ and $relations(c_i, c_j)$. For every class c in O, we define $closure(c)$ as the set of classes that either are superclasses of c or can reach c or one of its superclasses by a directed path whose links are object properties. For example, $closure(Person) = \{Organization, Event\}$ because there are the links *ceo* and *organizer* from *Organization* and *Event* to *Person*. As another example, $closure(City) = \{Place, State, Person, Organization, Event\}$. *Place* is in the set because it is a superclass of *City* and the other classes have a path to either *Place* or *City*. We define $relations(c_i, c_j)$ between two class nodes as the properties connecting c_i to c_j. It includes the $subClassOf$ relation and also the properties inherited from the class hierarchy. For instance, $relations(Person, City) = \{bornIn, livesIn\}$ and $relations(City, Place) = \{subClassOf, nearby, isPartOf\}$.

The algorithm has three main parts. In the first part (lines 4-17), we add a component to G for each semantic model m_i that is not a subgraph of the existing components, i.e., m_i introduces a new pattern. If m_i is subsumed by some of the components, we just update the $support\_models$ of the corresponding links in those components. That means if a pattern appears in k models, all the links of that pattern will have k elements in their $support\_models$. Figure 3 illustrates the graph built over $M = \{m_1, m_2, m_3\}$. Although we have three known models, two components are added to G since m_2 is a subgraph of m_1 and we only need to update the $support\_models$ of the common links between m_1 and m_2.

Next (lines 18-20), we find all the classes that are connected to the current nodes through a path in the ontology. To do this, for every class node v, we calculate $closure(uri(v))$. Then, for each $class\_uri$ in the resulting set, if G does not already include a node with the same label, we add a new node marked with $class\_uri$. In our example in Figure 3, applying this step adds nodes 9 and 11 to the graph because $Event \in closure(City) \cup closure(State) \cup closure(Person)$ and $Place \in closure(City) \cup closure(State)$. In the final part (lines 21-25), we use the ontology properties to connect different components of the graph. We compute $relations(v_i, v_j)$ to find all the possible links between the two class

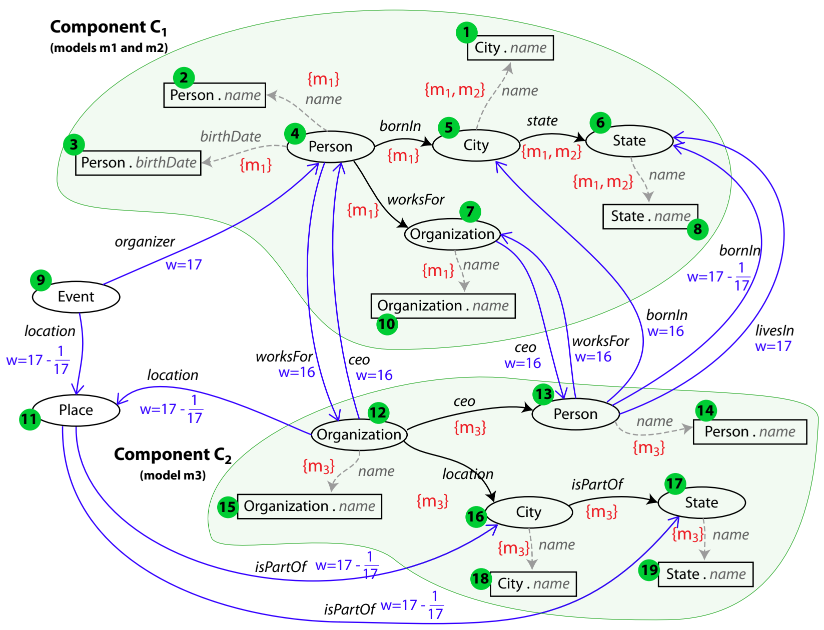

Fig. 3. The graph constructed from $M = \{m_1, m_2, m_3\}$ (semantic models of s_1, s_2, and s_3) and the domain ontology O

nodes v_i and v_j that are not both in the same component. For legibility, we only show some of the links in Figure 3.

Assigning weights to the links of the graph is fundamental in our algorithm. We assign a very low weight $w_{min} = \epsilon$ to all the links inside a component associated with semantic models (black links in Figure 3). For all other links (blue links in Figure 3), we assign a high weight w_{max}. The intuition behind this decision is to produce more coherent models later when we compute the minimal tree. In our example, $w_{max} = 17$ because the total number of links in the set of known models (M) is 17. Regarding the links that do not belong to any component of G (their *support_models* is empty), we use a simple counting mechanism to prefer the more popular ones (lines 42-44). For example, the weight of the link *worksFor* from node 13 to node 7 is 16 because we have seen the same link with the same domain and range in m_1 (the link from node 4 to node 7). As another example, the link *bornIn* from node 13 to node 6 has a weight of $(17 - \frac{1}{17})$. The reason is that although there exist one link with the same label in m_1, the target of the links do not match each other.

4.2 Semantic Labeling of Source Attributes

When presented with a new source $\hat{s}$ whose source description is unknown, the first step is recognizing semantic types of its data. We formally define a *semantic type* to be either an ontology class or a pair consisting of a data property and its

Algorithm 1. *BuildGraph(M, O)*

```
    Input: A set of known semantic models $M = \{m_1, \cdots, m_n\}$ and the domain ontology $O$
    Output: A weighted directed graph $G$ that will be used later in learning semantic descriptions of new sources

 1: $w_{min} \leftarrow \epsilon$
 2: $w_{max} \leftarrow \sum_{i=1}^{n} |Edges(m_i)|$        ▷ $w_{max}$ = total number of the links in $M$
 3: $components \leftarrow \{\}$        ▷ subgraphs of $G$ corresponding to the semantic models

    ▷ add the known semantic models to the graph
 4: for each $m_i \in M$ do
 5:     if $m_i$ is a subgraph of $c \in components$ then
 6:         let $c' \subseteq c$ be a subgraph of $c$ that matches $m_i$
 7:         add "$m_i$" to the $support\_models$ of the links in $c'$
 8:     else        ▷ $m_i$ introduces a new pattern
 9:         create a new component $c_i$ by cloning $m_i$
10:         for each link $e \in c_i$ do
11:             $support\_models(e) \leftarrow$ "$m_i$"
12:             $weight(e) \leftarrow w_{min}$
13:         end for
14:         $components \leftarrow components \cup c_i$
15:     end if
16: end for
17: $G = \bigcup_{c_i \in components} c_i$        ▷ add disjoint components to the graph $G$

    ▷ traverse the ontology $O$ to find the classes that do not map to any node in the graph
    but are connected to them through a path in the ontology
18: for each $class\ node\ v \in G$ do
19:     AddClosure($v, G$)
20: end for

    ▷ use the properties defined in $O$ to join the disconnected components
21: for $v_i, v_j \in G$ do
22:     if $v_i, v_j$ are both $class\ nodes$ but do not belong to the same pattern then
23:         AddLinks($v_i, v_j, G$)
24:     end if
25: end for

26: return $G$

27: procedure AddClosure($v, G$)
28:     $ClosureSet \leftarrow closure(uri(v))$
29:     for each $class\_uri \in ClosureSet$ do
30:         if there is no node in $G$ labeled with $class\_uri$ then
31:             add a new node $u$ to $G$
32:             $uri(u) \leftarrow class\_uri$
33:         end if
34:     end for
35: end procedure

36: procedure AddLinks($v_i, v_j, G$)
37:     $RelationSet \leftarrow relations(uri(v_i), uri(v_j))$
38:     for each $property\_uri \in RelationSet$ do
39:         add a new link $e$ from $v_i$ to $v_j$
40:         $uri(e) \leftarrow property\_uri$
41:         $support\_models(e) \leftarrow empty$
42:         $c_1 \leftarrow \sum_{e' \in E_1} |support\_models(e')|$ where $E_1$={links of $G$ whose label, source,
            and target match $e$}
43:         $c_2 \leftarrow \sum_{e' \in E_2} |support\_models(e')|$ where $E_2$={links of $G$ labeled with $uri(e)$}
44:         $weight(e) \leftarrow min(w_{max} - c_1, w_{max} - c_2/w_{max})$
45:     end for
46: end procedure
```

domain. We use a class as a semantic type for attributes whose values are URIs for instances of a class and for attributes containing automatically-generated database keys that can also be modeled as instances of a class. We use a data property/domain pair as a semantic type for attributes containing literal values. For example, the semantic type for the first attribute of s_4, *employer*, is $\langle Organization, name \rangle$.

We employ a supervised machine learning technique introduced in our previous work [12] to learn semantic types. To achieve high accuracy, we use a Conditional Random Field (CRF) [15] method that uses features extracted both from the attribute names and their values. The CRF is trained with user-specified assignments of attributes to semantic types, specified when the source descriptions for sources s_1 to s_n were constructed.

Applying this method on a new source $\hat{s}$ yields a set of candidate semantic types, each with a confidence value. Our algorithm then selects the top m semantic types for each attribute as an input to the next step of the process. To make the description of the source description construction algorithm simpler, we describe the case of $m = 1$ and then explain how our algorithm can be generalized to handle $m > 1$ (the general case is interesting because it enables the algorithm to cope with situations when the top ranked semantic type is incorrect). Thus, if the new source $\hat{s}$ has n attributes denoted by $Attributes(\hat{s}) = \{a_1, \cdots, a_n\}$, the output of the semantic labeling step is $Labels(\hat{s}) = \{l_1, \cdots, l_n\}$ where l_i is the semantic type of a_i. For example, for s_4 with $Attributes(s_4) = \{employer, employee\}$ we will have $Labels(s_4) = \{\langle Organization, name \rangle, \langle Person, name \rangle\}$.

4.3 Generating Candidate Models

So far, we have annotated the source attributes with semantic types. To build a complete source description we still need to determine the relationships between the attributes. For example, after labeling s_4's attributes, even though $\langle Organization, name \rangle$ and $\langle Person, name \rangle$ are assigned as the semantic types, the relationship between the attributes is not fully determined. It is not clear whether s_4 describes organizations and their employees (using *worksFor* property from *Person* to *Organization*) or organizations and their CEOs (using *ceo* property from *Organization* to *Person*). As we can see, even for simple sources like s_4 having few attributes, the problem of learning an accurate semantic description is a difficult problem. What we propose here is to leverage the knowledge of the known semantic models to discover the most popular and coherent patterns connecting the semantic labels of a new source $\hat{s}$.

The inputs to this step of our algorithm, $Labels(\hat{s})$, are the semantic types assigned to the new source $\hat{s}$ and the graph G, which includes the known semantic models, M, and is expanded using the domain ontology O. The output is a set of candidate source descriptions for the new source $\hat{s}$ where each candidate $(\hat{s}, \hat{m}, \hat{\phi})$ contains the model $\hat{m}$ along with the mapping $\hat{\phi}$ from the source attributes to the nodes of $\hat{m}$. Algorithm 2 shows the steps of our approach.

Algorithm 2. *GenerateCandidates(Labels($\hat{s}$), G)*

```
    Input: A set of semantic types $Labels(\hat{s}) = \{l_1, \cdots, l_n\}$ and the graph $G$
    Output: A set of candidate source descriptions $(\hat{s}, \hat{m}, \hat{\phi})$
 1: $Candidates \leftarrow \{\}$
    ▷ mapping semantic types to the nodes of the graph
 2: for each $l_i \in Labels(\hat{s})$ do
 3:     $matched(l_i) \leftarrow$ all the nodes in $G$ whose label match $l_i$
 4:     if $matched(l_i)$ is empty then          ▷ add new node(s) if no node matches $l_i$
 5:         if $l_i$ represents a class then
 6:             add a new class node $u$ to $G$ and $uri(u) \leftarrow l_i$
 7:             $matched(l_i) \leftarrow u$
 8:         else                                 ▷ $l_i$ is in form of $\langle domain, data\ property\rangle$
 9:             add a new data node $v$ to $G$ and $uri(v) \leftarrow l_i$
10:             $matched(l_i) \leftarrow v$
11:             add a new class node $u$ to $G$ and $uri(u) \leftarrow domain(l_i)$
12:             add a new link $e$ from $u$ to $v$ and $uri(e) \leftarrow data\ property(l_i)$
13:         end if
14:         AddClosure$(u, G)$                    ▷ compute the closure of the new node $u$
15:         for class nodes $v_i, v_j$ where either $v_i$ or $v_j \in$ new nodes do
16:             AddLinks$(v_i, v_j, G)$          ▷ connect the new added nodes to the other nodes
17:         end for
18:     end if
19: end for
20: $MatchedSet \leftarrow \{\{v_1, \cdots, v_n\} | v_i \in matched(l_i)\}$
    ▷ for each possible mapping from the semantic types to the nodes, we compute the minimal
    tree that connects those nodes
21: for each $\{v_1, \cdots, v_n\} \in MatchedSet$ do
22:     $SteinerNodes \leftarrow \{v_1, \cdots, v_n\}$
23:     $\hat{m} \leftarrow$ SteinerTree$(G, SteinerNodes)$          ▷ find the tree with minimal cost
24:     $\hat{\phi} \leftarrow \{\langle a_1, v_1\rangle, \cdots, \langle a_n, v_n\rangle\}$
25:     $Candidates \leftarrow Candidates \cup (\hat{s}, \hat{m}, \hat{\phi})$
26: end for
    return $Candidates$
```

In the first part of the algorithm (lines 2-20), we find all the mappings from the semantic types to the nodes of the graph. Since it is possible that a semantic type maps to more than one node in G, more than one mapping might exist from the semantic types to the nodes. For example, if we look into the graph shown in Figure 3, the semantic type $\langle Organization, name\rangle$ maps to nodes 10 and 15 and the semantic type $\langle Person, name\rangle$ maps to nodes 2 and 14. Thus, for $\hat{s} = s_4$, we have four mappings from the semantic types to the graph nodes, $r_1 = \{\langle Organization, name\rangle \rightarrow node\ 10, \langle Person, name\rangle \rightarrow node\ 2\}$, $r_2 = \{\langle Organization, name\rangle \rightarrow node\ 10, \langle Person, name\rangle \rightarrow node\ 14\}$, $r_3 = \{\langle Organization, name\rangle \rightarrow node\ 15, \langle Person, name\rangle \rightarrow node\ 2\}$, and $r_4 = \{\langle Organization, name\rangle \rightarrow node\ 15, \langle Person, name\rangle \rightarrow node\ 14\}$. If a semantic type does not map to any node in the graph, we add a new node to the graph. We use the procedure AddClosure to add the related nodes and then call AddLinks to connect the newly-added nodes to the rest of the nodes in G.

In the next step (lines 21-26), for each set of nodes in each mapping, we find the minimum-cost tree connecting these nodes. 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. Because the Steiner tree problem is NP-complete, we use an approximation algorithm [14] with an approximation ratio bounded by $2(1 - 1/l)$, where l is the number of leaves in the optimal Steiner tree. One problem with this algorithm is that if there is large number of mappings from the semantic types to the nodes of the graph, the algorithm becomes inefficient, because we will get a large number of sets as the possible Steiner nodes and we need to run the Steiner tree module over all of them. To solve this problem, we perform an optimization step right after computing the possible mappings. We use a blocking method to eliminate the mappings that are unlikely to generate higher ranked models. This step is not shown in Algorithm 2 to make the algorithm more readable, but we explain it here.

As we will see in the next section, one of the factors to rank the candidate models is their cost. While it is true that the exact cost cannot be calculated until we compute the Steiner tree, the way the links are weighted in G enables us to estimate which sets of Steiner nodes generate lower-cost models. As previously described, all the links inside a known pattern have a very low weight (ϵ). Consequently, sets of Steiner nodes containing larger number of nodes belonging to the same pattern are more likely to yield Steiner trees with lower cost. We apply this idea by sorting all sets of Steiner nodes ($MatchedSet$ in line 20) based on how coherent each set is. For example, considering the four possible mappings we showed earlier for s_4, r_1 and r_4 will be ranked higher than r_2 and r_3, as their matched nodes are part of the same pattern. Once all the node sets are sorted, we pick the top k ones and compute the Steiner tree algorithm only over these sets to generate k candidate models for the new source $\hat{s}$. Figure 4 illustrates the top two candidate models for s_4 and s_5. The inverted arrows ($\updownarrow$) depict the mappings from the source attributes to the nodes of the models (ϕ).

The blocking method to reduce the number of mappings also supports generalizing our algorithm to handle the case where each attribute is labeled with a set of possible semantic types rather than only one semantic type (case $m > 1$ discussed in the previous section). To handle this case, we compute the set of all permutations of the semantic types and for each permutation, we find the

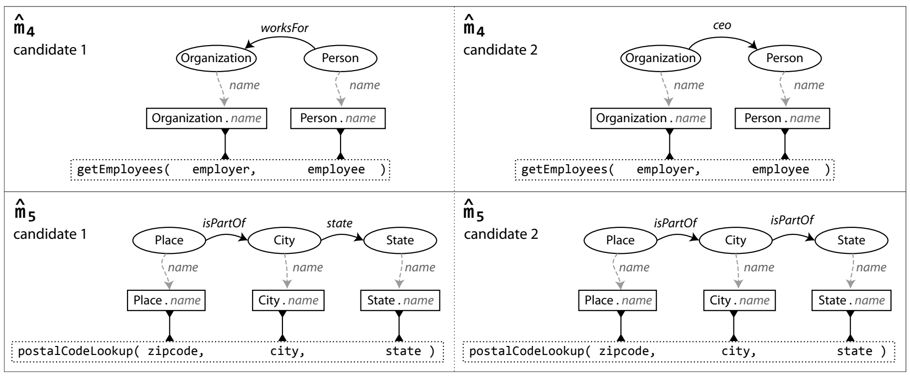

Fig. 4. Top two candidate models generated for s_4 and s_5

possible mappings from the semantic types to the nodes of the graph. Once the mappings are calculated, we apply the blocking technique to get the k most promising node sets. Then, we run the Steiner tree algorithm for each node set to generate k candidate models.

4.4 Ranking Source Descriptions

The final step in learning the semantic description of a source is ranking the candidates produced in the previous step. We define two metrics to rank source descriptions, *coherence* and *cost*. The cost of a candidate $(\hat{s}, \hat{m}, \hat{\phi})$ is the sum of the link weights, $\sum_{e \in \hat{m}} weight(e)$. The goal of defining the coherence is to give more priority to the models containing larger segments from the known patterns. Let $E_p = \{e | e \in \hat{m} \wedge |support\_models(e)| > 0\}$ be the links in model $\hat{m}$ coming from an observed pattern. We partition E_p to create groups of links that belong to the same pattern. More concretely, we define a list $I = (\langle x_1, y_1 \rangle, \cdots, \langle x_n, y_n \rangle)$, where x_i is the size of a group of links sharing a model identifier in their *support_models* (links seen in the same pattern), y_i is the number of model identifiers shared between all of the links in that group, and $\sum_{i=1}^{n} x_i = |E_p|$. In Figure 4, both candidates of $\hat{m}_4$ have $I = \{\langle 3, 1 \rangle\}$, for the first candidate of $\hat{m}_5$ (at the left) $I = \{\langle 3, 2 \rangle\}$, and for the second candidate of $\hat{m}_5$ (at the right), $I = \{\langle 3, 1 \rangle\}$. Note that in $\hat{m}_5$, E_p does not include the links connecting $Place$ to the other nodes because $Place$ does not exist in any of the observed patterns.

We sort the candidate source descriptions first based on their coherence and then based on their cost. To compare two models regarding the coherence, we compare their coherence lists. We sort the elements of each list descending and then compare the elements one by one. The inequality equation between two elements $z_1 = \langle x_i, y_i \rangle$ and $z_2 = \langle x_j, y_j \rangle$ can be defined as $[z_1 > z_2;\ if\ (x_i > x_j) \vee (x_i = x_j \wedge y_i > y_j)]$. For example, for s_5 the first candidate will be ranked higher than the second one and for s_4, both candidates will be ranked in the same place since they have the same coherence list and the same cost.

5 Evaluation

We evaluated our approach using 17 data sources containing overlapping data (the first column in Table 1 shows the signatures of these sources). We created source descriptions for them manually using the DBpedia, FOAF, GeoNames, and WGS84 ontologies, and used these source descriptions as the gold standard to evaluate our approach. We then used our algorithm to learn a source description for each source given the manually created source descriptions of the other sources as training data (The original source descriptions and the results are available at: `http://isi.edu/integration/data/iswc2013`). Since the semantic labeling is not the focus of this paper, we assume that Algorithm 2 is given the correct semantic type for each attribute (we evaluated the semantic labeling in our previous work [13, 24]).

We used $k = 50$ in the third step of our approach to generate 50 candidate source descriptions and measured the graph edit distance (GED) between the top ranked description and the manually created one. The results are shown in the second column in Table 1. The value of GED is the (minimum) sum of the costs of the edit operations needed to convert one graph to another. The edit operations include node insertion, node deletion, edge insertion, edge deletion and edge relabeling. Edge relabeling means that we substitute a link between two nodes with another link with the same direction but another URI. We assigned a cost of one to each edit operation.

We compared the results of our approach with the results of Karma [13], a data integration tool that allows users to semi-automatically create source descriptions for sources and services. To make the Karma results comparable to our approach, we also gave Karma the correct semantic types for each attribute. We measured GED between the source description that Karma generates automatically (i.e., without user adjustments) and the gold standard. The results are shown in the third column of Table 1.

The results show that our algorithm generates source descriptions that are more than twice as accurate than Karma-generated ones. Karma learns to assign semantic types, but in this evaluation we gave it the correct semantic types, so Karma was not leveraging any knowledge learned from other source descriptions. Our approach outperformed Karma on all the sources except one. The one incorrect choice is not unexpected since there are cases for which there is no prior evidence or the evidence is misleading. Both systems use a Steiner-tree algorithm to compute their models, so the results show that the learning

Table 1. Evaluation results for learning the semantic descriptions of sample data sources. The second column is the graph edit distance between our hypotheses and the correct source descriptions and the third column is the edit distance between the Karma-generated source descriptions and the correct ones.

Source Signature	**GED** of our models	**GED** of Karma models
nearestCity(lat,lng,city,state,country)	1	6
findRestaurant(zipcode,restaurantName,phone,address)	0	1
zipcodesInCity(city,state,postalCode)	1	3
parseAddress(address,city,state,zipcode,country)	1	6
companyCEO(company,name)	0	1
personalInfo(firstname,lastname,birthdate,brithCity,birthCountry)	1	4
citiesOfState(state,city)	0	1
restaurantChef(restaurant,firstname,lastname)	1	2
capital(country,city)	1	2
businessInfo(company,phone,homepage,city,country,name)	8	10
findSchool(city,state,name,code,homepage,ranking,dean)	6	8
ocean(lat,lng,name)	1	2
employees(organization,firstname,lastname,birthdate)	2	1
education(person,hometown,homecountry,school,city,country)	4	9
postalCodeLookup(zipcode,city,state,country)	1	6
country(lat,lng,code,name)	0	2
administrativeDistrict(city,province,country)	1	4
Total	**29**	**68**

algorithms presented here enable the system to produce more accurate source descriptions.

We also evaluated our approach on five museum datasets modeled using the Europeana EDM, SKOS and FOAF ontologies. The models were created by domain experts for the purpose of creating an integrated dataset. The preliminary results show a 30% improvement, which we believe can be improved further. This improvement is not as large as the improvement on the other test dataset, but it shows that the method works with large, real-world datasets and ontologies.

6 Related Work

The problem of describing semantics of data sources is at the core of data integration [8] and exchange [3]. The main approach to reconcile the semantic heterogeneity among sources consists in defining logical mappings between the source schemas and a common target schema. The R2RML W3C recommendation [6] captures this practice for Semantic Web applications. Although these mappings are declarative, defining them requires significant technical expertise, so there has been much interest in techniques that facilitate their generation.

The mapping generation problem is usually decomposed in a *schema matching* phase followed by *schema mapping* phase [4]. Schema matching [20] finds correspondences between elements of the source and target schemas. For example, iMAP [7] discovers complex correspondences by using a set of special-purpose searchers, ranging from data overlap, to machine learning and equation discovery techniques. We use our previous work on semantic labeling [12], which considers attributes that map to the same semantic type as potential matches. Schema mapping defines an appropriate transformation that populates the target schema with data from the sources. Mappings may be arbitrarily procedures, but of greater interest are declarative mappings expressible as queries in SQL, XQuery, or Datalog. These mapping formulas are generated by taking into account the schema matches and schema constraints. There has been much research in schema mapping, from the seminal work on Clio [10], which provided a practical system and furthered the theoretical foundations of data exchange [11] to more recent systems that support additional schema constraints [17]. Alexe et al. [1] generate schema mappings from examples of source data tuples and the corresponding tuples over the target schema. Karma [13] and An et al. [2] generate mappings into ontologies, suggested by exploring low-cost Steiner trees that connect matching semantic types within a graph derived from the target ontology. Karma allows the user to correct the mappings interactively.

Our work in this paper is complementary to these schema mapping techniques. Instead of focusing on satisfying schema constraints, we analyze known source descriptions to propose mappings that capture more closely the semantics of the target source, in ways that schema constraints could not disambiguate. For example, suggesting that a *worksFor* relationship is more likely than *ceo* in a given domain. Moreover, following Karma, our algorithm can incrementally refine the mappings based on user feedback and improve future predictions. Carman and

Knoblock [5] also use known source descriptions to generate a LAV mapping for an unknown target source. However, a limitation of that work is that their approach could only learn descriptions that were conjunctive combinations of known source descriptions. By exploring paths in the domain ontology, in addition to patterns in the known sources, we can hypothesize target mappings that are more general than previous source descriptions or their combinations.

Semantic annotation of services [21, 22] and more recently of web tables [16, 18, 25] has also received attention. Most of this work learns types for services parameters or table columns, but is limited in learning relationships. Limaye et al [16] generate binary relationships leveraging the Yago ontology.

Ontology alignment [9] usually considers alignments between individual classes, so it is more applicable to the matching phase. However, Parundekar et al. [19] use an extensional approach to discover alignments between conjunctions and disjunctions of classes from linked data ontologies.

7 Discussion

We presented a novel approach to automatically learn the semantic description of a new source given a set of known semantic descriptions as the training set and the domain ontology as the background knowledge. The learned semantic descriptions explicitly represent the relationships between the source attributes in addition to their semantic types. These precise descriptions of data sources makes it possible to automatically integrate the data across sources and provides rich support for source discovery.

In our approach we build a graph whose main components are the known semantic descriptions expanded using the domain ontology. Next, we use a machine learning technique to label the attributes of the new source with classes and properties of the ontology. We find the possible one-to-one mappings from the semantic types to the nodes of the graph and calculate the top k promising mappings. Then, we build a tree over each mapping to generate k candidate models. Finally, we score the candidates to output a ranked list of the most plausible semantic models. The evaluation results showed that our algorithm generates models that are more accurate than Karma, a state of the art tool to semi-automatically model data sources.

The graph construction in the presented algorithm is an incremental process, i.e., we augment the graph with a new component when a new known model is presented to the system. The algorithm that we use to compute the Steiner tree is an approximation algorithm whose complexity is $O(|S||V|^2)$ where V is the set of nodes in the graph and S is a subset of the nodes (size of S is equal to the number of the new source attributes). Thus, computing the candidate models might be challenging when the size of the graph is very large in terms of the number of the nodes, even though we are using a polynomial time algorithm. In future work, we plan to investigate the idea of creating a more compact graph by consolidating the overlapping segments of the known semantic models. This will reduce the number of nodes added to the graph when a new pattern is given to the system. We also plan to integrate our approach into Karma in order to suggest more accurate semantic models to users. This will make it possible to automatically produce source descriptions with minimal user input.

Acknowledgements. This research is based upon work supported in part by the National Science Foundation under award number IIS-1117913 and in part by the NIH through the following NIGMS grant: U24 GM104203 Bio-Informatics Research Network Coordinating Center (BIRN-CC). 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 or any person connected with them.

References

1. Alexe, B., ten Cate, B., Kolaitis, P.G., Tan, W.C.: Designing and Refining Schema Mappings via Data Examples. In: SIGMOD, Athens, Greece, pp. 133–144 (2011)
2. An, Y., Borgida, A., Miller, R.J., Mylopoulos, J.: A Semantic Approach to Discovering Schema Mapping Expressions. In: Proceedings of the 23rd International Conference on Data Engineering (ICDE), Istanbul, Turkey, pp. 206–215 (2007)
3. Arenas, M., Barcelo, P., Libkin, L., Murlak, F.: Relational and XML Data Exchange. Morgan & Claypool, San Rafael (2010)
4. Bellahsene, Z., Bonifati, A., Rahm, E.: Schema Matching and Mapping, 1st edn. Springer (2011)
5. Carman, M.J., Knoblock, C.A.: Learning Semantic Definitions of Online Information Sources. Journal of Artificial Intelligence Research 30(1), 1–50 (2007)
6. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. W3C Recommendation (September 27, 2012), http://www.w3.org/TR/r2rml/
7. Dhamankar, R., Lee, Y., Doan, A., Halevy, A., Domingos, P.: iMAP: Discovering Complex Semantic Matches between Database Schemas. In: International Conference on Management of Data (SIGMOD), New York, NY, pp. 383–394 (2004)
8. Doan, A., Halevy, A., Ives, Z.: Principles of Data Integration. Morgan Kauffman (2012)
9. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer, Heidelberg (2007)
10. Fagin, R., Haas, L.M., Hernández, M., Miller, R.J., Popa, L., Velegrakis, Y.: Clio: Schema Mapping Creation and Data Exchange. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Mylopoulos Festschrift. LNCS, vol. 5600, pp. 198–236. Springer, Heidelberg (2009)
11. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data Exchange: Semantics and Query Answering. Theoretical Computer Science 336(1), 89–124 (2005)
12. Goel, A., Knoblock, C.A., Lerman, K.: Exploiting Structure within Data for Accurate Labeling Using Conditional Random Fields. In: Proc. ICAI (2012)
13. Knoblock, C.A., et al.: 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)
14. Kou, L., Markowsky, G., Berman, L.: A Fast Algorithm for Steiner Trees. Acta Informatica 15, 141–145 (1981)
15. Lafferty, J., McCallum, A., Pereira, F.: Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In: Proceedings of the 18th International Conference on Machine Learning (2001)

16. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and Searching Web Tables Using Entities, Types and Relationships. PVLDB 3(1), 1338–1347 (2010)
17. Marnette, B., Mecca, G., Papotti, P., Raunich, S., Santoro, D.: ++Spicy: an OpenSource Tool for Second-Generation Schema Mapping and Data Exchange. In: Procs. VLDB, Seattle, WA, pp. 1438–1441 (2011)
18. Mulwad, V., Finin, T., Joshi, A.: A Domain Independent Framework for Extracting Linked Semantic Data from Tables. In: Ceri, S., Brambilla, M. (eds.) Search Computing III. LNCS, vol. 7538, pp. 16–33. Springer, Heidelberg (2012)
19. Parundekar, R., Knoblock, C.A., Ambite, J.L.: Discovering Concept Coverings in Ontologies of Linked Data Sources. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012, Part I. LNCS, vol. 7649, pp. 427–443. Springer, Heidelberg (2012)
20. Rahm, E., Bernstein, P.: A Survey of Approaches to Automatic Schema Matching. VLDB Journal 10(4) (2001)
21. 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)
22. Sheth, A.P., Gomadam, K., Ranabahu, A.: Semantics Enhanced Services: METEOR-S, SAWSDL and SA-REST. IEEE Data Eng. Bulletin 31(3), 8–12 (2008)
23. Szekely, P., Knoblock, C.A., Yang, F., Zhu, X., Fink, E.E., Allen, R., Goodlander, G.: Connecting the Smithsonian American Art Museum to the Linked Data Cloud. In: Cimiano, P., Corcho, O., Presutti, V., Hollink, L., Rudolph, S. (eds.) ESWC 2013. LNCS, vol. 7882, pp. 593–607. Springer, Heidelberg (2013)
24. Taheriyan, M., Knoblock, C.A., Szekely, P., Ambite, J.L.: Rapidly Integrating Services into the Linked Data Cloud. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012, Part I. LNCS, vol. 7649, pp. 559–574. Springer, Heidelberg (2012)
25. 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)