LEARNING THE SEMANTICS OF STRUCTURED DATA SOURCES

by

Mohsen Taheriyan

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

(COMPUTER SCIENCE)

December 2015

To my wife, Shima, and my parents

for their encouragement, love, and patience.

Acknowledgments

I would very much like to thank my thesis advisor Craig Knoblock for his support, patience, and valuable feedback on every aspect of my research. Craig is a wonderful mentor. He spent hours with me discussing research ideas and reviewing my drafts. Through him, I learned how to present my research ideas and how to write research papers. What I like about him the most is his ability to keep the research direction aligned with the big picture. He always gave me the freedom to explore my own paths toward solving problems, while at the same time helping me to stay on the right track. I am very grateful to him to make my doctoral study an enjoyable experience.

I would also like to thank the other members of both my proposal and dissertation committees for providing me valuable comments on my thesis: Professor Pedro Szekely, Professor Jose Luis Ambite, Professor Cyrus Shahabi, Professor Victor Prasanna, and Professor Dennis McLeod. Their feedback and support is greatly appreciated. I would like to especially thank Pedro Szekely and Jose Luis Ambite who contributed greatly to my research. They were present in my weekly research meetings with Craig and suggested insightful ideas whenever I faced difficulties in tackling hard problems. I have been lucky to have their advice over the course of my research.

I feel very fortunate to have been a member of the Intelligent Systems Division (ISD) at the Information Sciences Institute (ISI), where I have had the great pleasure to work with a supportive and inspiring group of researchers. I want to thank all of my colleagues in the Information Integration Group at ISI. Thanks to my officemates Bo Wu and Jason Slepicka who were always available when I needed help. Thank you Yinyi Chen for your help in creating the gold standard models for my evaluation. Thank you Dipsy Kapoor for your help in programming the Karma data integration system. Thank you Alma Nava for your administrative help during my years at ISI.

I would like to thank my wonderful friends in LA who made my graduate life fun. Thanks to Payman for being a challenging opponent in playing Mortal Kombat, and thanks to Reza, Amirsoheil, Nima, and Jalal for being easy to beat in playing FIFA. Thank you Soheil for being a great gym companion. Thank you Marjan and Sepideh for being such great friends. Thank you Farshad for the exciting billiard games we had in our coffee breaks at ISI. I would like to thank you all for the amazing time we had together.

I am thankful to my family for always supporting me in my academic pursuits. Thank you Mom and Dad for your love and patience during my studies. Thank you my brother, Mehran, and my sister, Nahid, for your support.

Finally, I would like to thank my lovely wife Shima. The patience, love, and compassion you have shown me goes beyond words. I cherish all the moments we spend together and I feel incredibly lucky to share my life with you.

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

Table of Contents

List of Tables

List of Figures

Abstract

Information sources such as relational databases, spreadsheets, XML, JSON, and Web APIs contain a tremendous amount of structured data, however, they rarely provide a semantic model to describe their contents. Semantic models of data sources capture the intended meaning of data sources by mapping them to the concepts and relationships defined by a domain ontology. Such models are the key ingredients to automate many tasks such as source discovery, data integration, and publishing semantic content on the Web. Manually modeling the semantics of data sources requires significant effort and expertise, and although desirable, building these models automatically is a challenging problem. Most of the effort to automatically build semantic models is focused on labeling the data fields (source attributes) with ontology classes and/or properties, e.g., annotating the first column of a table with the class Person and the second one with the class Movie. However, a precise semantic model needs to explicitly represent the relationships between the attributes in addition to their semantic types, e.g., stating that the person is the director of the movie. Automatically constructing such precise models is a difficult task.

We present a novel approach that exploits the knowledge from a domain ontology, the semantic models of previously modeled sources, and the vast amount of data available in the Linked Open Data (LOD) cloud to automatically learn a rich semantic model for a new source. This model represents the semantics of the new source in terms of the concepts and relationships defined by the domain ontology. Given some sample data from the new source, we leverage the knowledge in the domain ontology and either the known semantic models or the LOD cloud to construct a weighted graph that represents the space of plausible semantic models for the new source. Then, we compute the top k candidate semantic models 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 models on future data sources. Our evaluation shows that our method generates expressive semantic models for data sources and services with minimal user input. These precise models make it possible to automatically integrate the data across sources and provide rich support for source discovery and service composition. They also make it possible to automatically publish semantic data into knowledge graphs.

Chapter 1

Introduction

Today, information sources such as relational databases and Web services provide a vast amount of structured data. Given all the structured data available, we would like to combine the data to answer specific user queries. A common approach to integrate 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 [Doan et al., 2012].

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, the domain model is an *ontology* that formally represents concepts within a domain and properties and interrelationships of those concepts. In this context, what is meant by a source description is a schema mapping from the source to an ontology. We can represent this mapping as a *semantic network* with ontology classes as the nodes and ontology properties as the links between the nodes. This network, also called a *semantic model*, describes the source in terms of the concepts and relationships defined by the domain ontology. Considering the ontology shown in Figure 1.1 as the domain ontology, Figure 1.2 depicts the semantic model of a sample data source as a mapping between the source and this ontology.

One step in building a semantic model of a data source is *semantic labeling*, determining the *semantic types*[1] of its data fields, or *source attributes*. That is, each source attribute is labeled with a class and/or a data property of the domain

[1] In this dissertation, the terms *semantic type* and *semantic label* are used interchangeably.

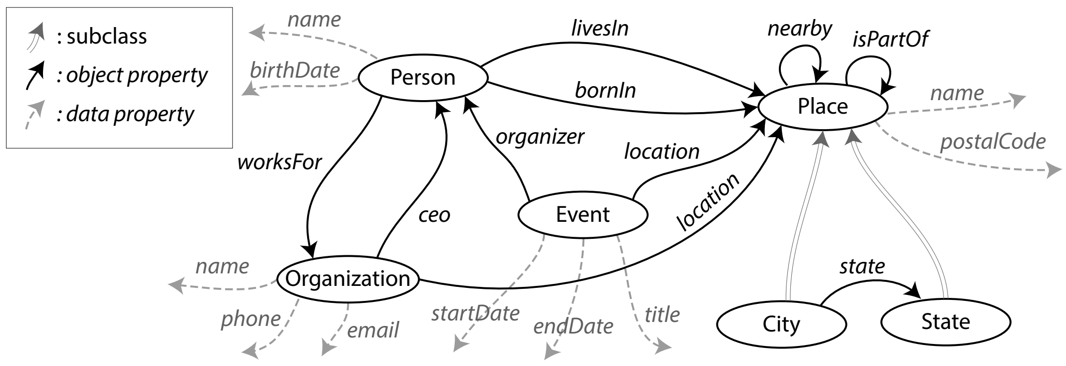
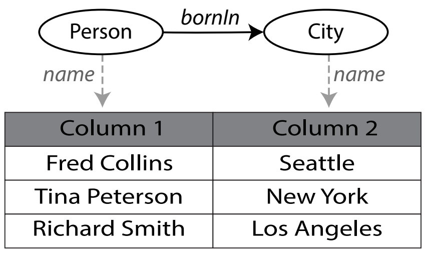


Figure 1.1: An example ontology.



Column 1	Column 2
Fred Collins	Seattle
Tina Peterson	New York
Richard Smith	Los Angeles

Figure 1.2: The semantic model of a sample data source.

ontology. In our example in Figure 1.2, the semantic types of the first and second columns are *name* of *Person* and *name* of *City* respectively. However, simply annotating the attributes is not sufficient. Unless the relationship between the columns is explicitly specified, we do not know whether the city is the place where the person was born or it is the place where the person lives in. To build a semantic model that fully recovers the semantics of the data, we need a second step that determines the relationships between the source attributes in terms of the properties in the ontology.

Manually constructing semantic models requires significant effort and expertise. Although desirable, generating these models automatically is a challenging problem. In Semantic Web research, there are many studies on mapping data sources to ontologies [Han et al., 2008; Sheth et al., 2008; Langegger and Wöß, 2009; Sahoo et al., 2009; Polfliet and Ichise, 2010; Limaye et al., 2010; Vavliakis et al., 2010; Ding et al., 2010; Saquicela et al., 2011; Venetis et al., 2011; Wang et al., 2012; Mulwad et al., 2013], but most focus on semantic labeling or are very limited in automatically inferring the relationships. In this work, we introduce a novel approach to construct semantic models that not only include the semantic types of the source attributes, but also describe the relationships between them.

1.1 Motivation

Semantic models of data sources are beneficial to automate tasks such as source discovery and information integration. We illustrate this by giving a concrete example. Suppose that we want to develop an intelligent system to answer queries about people and organizations in Los Angeles. Examples of the questions the system should be able to answer are: (a) Who is the CEO of a given company? (b) In what cities does a particular person work?[2] In order to answer the queries, the system needs to look for sources that are relevant. In this case, relevant sources might be:

- $s_1 = personalInfo(name, birthdate, city, state, workplace)$
- $s_2 = businessInfo(company, manager, postalcode)$
- $s_3 = postalCodeLookup(zipcode, city, state)$

s_1 is a table in a remote database providing information about people who live in Los Angeles including their name, birthdate, birthplace, and the organization they are currently working for; s_2 is a spreadsheet on the Web containing name,

[2]A system capable of generating a plan to answer such queries is called an Information Mediator[Wiederhold, 1992].

CEO, and location of the Los Angeles companies; and s_3 is a Web service that returns the city and state of a given ZIP code.

For the system to know which sources are relevant, it needs to know what information each source provides. While syntactic information about data sources such as attribute names or attribute types (string, int, date, ...) may give the system some hints to discover relevant sources, they are often not sufficient, e.g., name of the first field in s_1 is *name* and we do not know whether this is an organization's name or a person's name. Moreover, different sources may have different terms as names of the attributes that actually represent the same meaning, e.g., *workplace* in s_1 and *company* in s_2 both represent names of organizations. Therefore, the system cannot rely only on names of attributes to automatically combine the data from different sources.

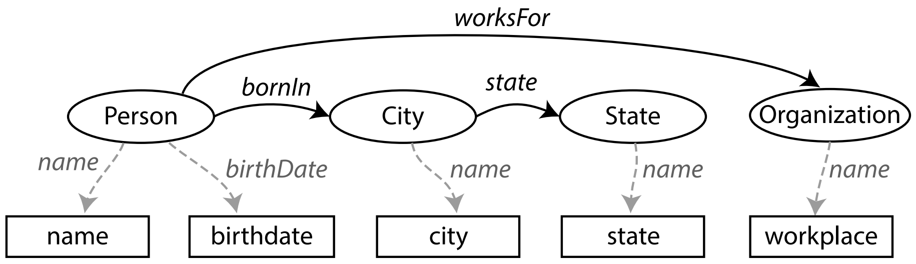
Knowing the semantic types of the source attributes, to some extent, empowers the system to discover relevant sources. Assume that the system knows the mapping from the source attributes to the types defined in the ontology given in Figure 1.1. Now, to answer the query (a),"Who is the CEO of a given company?", the system finds the sources including *Organization.name* and *Person.name* in their semantic types, namely s_1 and s_2.

Semantic types partially reveal the meaning of the data, however, because the relationships between the attributes are not specified, there is uncertainty about where to extract and how to combine the required data to answer a query. In our example, s_1 contains work place of people and s_2 contains CEO of companies. Thus, the link *worksFor* is the appropriate link to model the relationship between *Person* and *Orgnization* in s_1 and the link *ceo* is the correct one to express the relationship between the same semantic types is s_2. Given such information, now the system will be able to provide an answer to the query (a) by looking up the query input in the values of the first column in s_2 (*company*) and returning the corresponding value in the second column (*manager*).

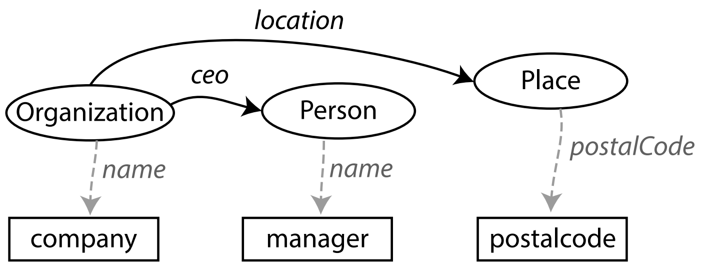
Hence, to be able to automatically discover relevant sources and integrate them to answer queries, we need a precise model of each source describing the semantic types and the relationships according to the domain ontology. This expressive model is what we call a semantic model. Semantic models of s_1, s_2, and s_3 are shown in Figure 1.3. Having these semantic models, the system computes the answer to the query (b), "In what cities does a particular person work?", by first finding the person's name in s_1 to get the company where he or she is working, then looking up the company name in s_2 to extract its postal code, and finally invoking the Web service s_3 using the extracted postal code as input and returning the output city as the answer to the query.

Semantic models of structured sources can be exploited to automatically publish data into knowledge graphs. Knowledge graphs have recently emerged as a rich and flexible representation of domain knowledge. Nodes in this graph represent the entities and edges show the relationships between the entities. Associating semantics with data provides support for better search, interoperability and integration. Large companies such as Google and Microsoft employ knowledge graphs as a complement for their traditional search methods to enhance the search results with semantic information.

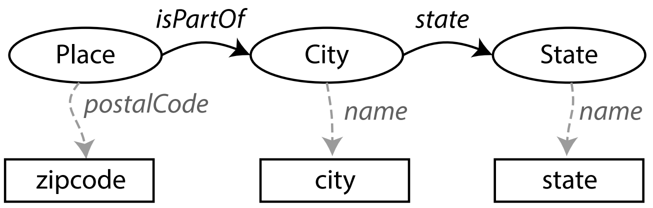
Semantic models of data sources enable us to easily transform the data into semantic formats such as RDF and publish it on the Web. Once data of a source is converted to RDF, the published data can be linked to the semantic data already published from other datasets, making a huge cloud of linked data available on the Web, which is called Linked Open Data (LOD) [Bizer et al., 2009]. LOD is an ongoing effort in the Semantic Web community to build a massive public knowledge



(a) s_1 =*personalInfo(name, birthdate, city, state, workplace)*



(b) s_2 =*businessInfo(company, manager, postalcode)*



(c) s_3 =*postalCodeLookup(zipcode, city, state)*

Figure 1.3: Semantic models of s_1, s_2, and s_3.

graph. The goal is to extend the Web by publishing various open datasets as RDF on the Web and then linking data items to other useful information from different data sources. With linked data, starting from a certain point in the graph, a person or machine can explore the graph to find other related data. Our work plays an important role in the first step of publishing linked data, automatically publishing datasets as RDF using a common domain ontology.

1.2 Thesis Statement

The knowledge of previously modeled sources as well as the semantic data available in the Linked Open Data cloud can be leveraged to learn accurate semantic models of structured data sources, enabling automated source discovery and data integration.

1.3 Proposed Approach

Our work on learning semantic models of structured sources is divided into three parts. First, we present an algorithm to semi-automatically construct a semantic model for a new source given samples of the source data and the domain ontology as inputs [Knoblock et al., 2012]. The algorithm learns candidate semantic types by employing the technique proposed by Krishnamurthy et al. [Krishnamurthy et al., 2015] and then extracts the relationships by computing a minimal tree in a graph derived from the domain ontology and the semantic types. In the first part, learning the semantic types, if the correct semantic type is not among the suggested types, users can browse the ontology through a user-friendly interface to select the appropriate type and the system learns from these manual assignments. In the second part, extracting the relationships, if the suggested tree is not the correct model of the data, users can refine the relationships through a graphical user interface to build the correct model. However, users need to go through this refinement process for each source even if they have already modeled some similar sources. The reason is that the algorithm does not learn the changes done by the user in relationships.

In the second part of our work, we exploit the knowledge of previously modeled sources to make the process of constructing semantic models more automated

[Taheriyan et al., 2013, 2014, 2015a]. The insight of our approach is that different sources in the same domain often provide similar or overlapping data. Thus, we use the already modeled sources to learn the patterns that more likely represent the intended meaning of a new source. Once we learned the candidate semantic types for the source attributes, we construct a graph using the known semantic models, the semantic types, and the domain ontology. This graph represents the space of plausible semantic models. Next, we produce mappings from the source attributes 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. Then, we score the models to prefer the ones formed with more coherent and frequent patterns and generate a ranked list of semantic models for the given source. We can put users in the loop by allowing them to select the correct model or refine one of the suggested models. The graph will be updated with the final semantic model yielding more accurate models for future data sources.

Finally, we present an approach that leverages the significant amount of semantic content available in Linked Open Data (LOD) to infer semantic models [Taheriyan et al., 2015b]. This part, complements the second part of our work in cases where few known semantic models are available. The LOD cloud contains a vast amount of semantic data that can be used to learn how instances of different classes are linked to each other. First, we extract small graph patterns occurring in the linked data. Next, we combine these patterns into one graph and expand the resulting graph using the paths inferred from the domain ontology. Then, we map the source attributes to the nodes of the graph and compute top k candidate models.

1.4 Contributions of the Research

The main contribution of our work are the techniques to leverage attribute relationships in a domain ontology, known semantic models, and the LOD cloud to hypothesize relationships among attributes for new sources and capture them in source models. Such source models are key to automating tasks such as source discovery, information integration, and service composition. They also make it possible to publish data sources into knowledge graphs, e.g., converting data to RDF and publishing it into the LOD cloud. Overall, in this thesis, we make the following contributions:

- Introduce a graph-based approach to semi-automatically model the relationships within structured data
- Learn semantic models of data sources from the semantic models of the previously modeled sources (known semantic models)
- Leverage Linked Open Data to infer semantic relations within data sources

1.5 Outline of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, we introduce our graph-based approach to semi-automatically build a semantic model for structured data sources. Chapter 3 presents our approach to automatically learn a semantic model for a new unknown source from the known semantic models. Chapter 4 describes our technique leveraging Linked Open Data to learn semantic models. Chapter 5 includes the related work and Chapter 6 concludes the dissertation.

Chapter 2

Semi-Automatically Building Semantic Models of Data Sources

In this chapter, we introduce a semi-automatic approach to model the semantics of a structured data source [Knoblock et al., 2012]. The inputs to the approach are the domain ontology and some sample data from the source and the output is a graph model that includes the semantic types of the source attributes and explicitly represents their relationships.

We provide an example to clarify the problem of building semantic models. This example will be used throughout this chapter to explain our approach to solve the problem. Suppose that we have a table in a relational database that contains five columns: *name*, *birthdate*, *city*, *state*, and *workplace*. The signature of this data source can be written as: $s(name, birthdate, city, state, workplace)$. Given the ontology in Figure 1.1, our goal is to build a semantic model that characterizes the source s in terms of the concepts and relationships defined by the ontology.

Automatically building a semantic model of an unknown source is difficult. First, the system must map the source attributes to the classes and data properties in the ontology. For example, the attribute *name* should be mapped to the data property *name* of the class *Person* and the attribute *workplace* should be mapped to the data property *name* of the class *Organization*. 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

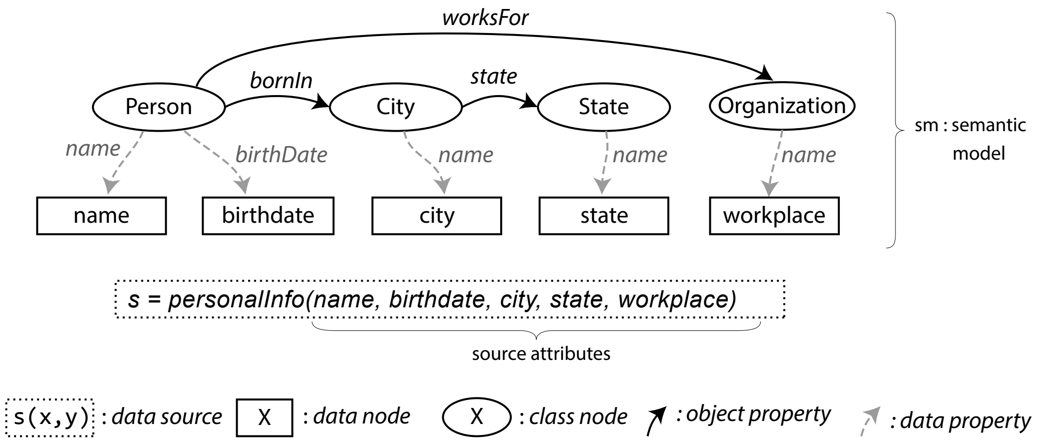


Figure 2.1: Semantic model of the source s.

ceo. The system needs to select *worksFor*, which captures the intended meaning of s. The problem is more complicated in cases where the relevant classes are not directly connected in the ontology and there exist multiple paths connecting them to each other. Figure 2.1 illustrates the desired semantic model of s.

2.1 Problem Formulation

Having given the example above, we state the problem of modeling a source formally.

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

A *semantic model* of the source s, $sm(s)$, is a directed graph containing two types of nodes, *class nodes* and *data nodes*. Class nodes (ovals in Figure 2.1) correspond to the classes in the ontology and are labeled with class URIs ($v \in class\ nodes \Rightarrow l_v = class\_uri$). Data nodes (rectangles in Figure 2.1) correspond to the source attributes and are labeled with the names of the attributes ($v \in data\ nodes \Rightarrow l_v = attribute\_name$). The mapping from the source attributes to the data nodes can be a partial mapping, i.e., only some of the attributes of s have corresponding data nodes in $sm(s)$.

The links in the graph correspond to the properties in the ontology and are labeled with property URIs ($e \in links \Rightarrow l_e = property\_uri$). In general, a semantic model may contain multiple nodes and links labeled with the same URI. The semantic model 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*).

The problem of building semantic models can be stated as follows. Let O be the domain ontology. Given a new source s, our goal is to semi-automatically compute $sm(s)$ with minimal user input. Clearly, many models are well-formed semantic models, but only one or a few capture the intended meaning of the data contained in s.

In general, it is not always possible to automatically compute the desired mapping between a source and an ontology since there may not be enough information in the source to determine the mapping. So, the automated process computes the most succinct mapping (Section 2.2 and Section 2.3), and the user interface allows the user to guide the process towards the desired interpretation (Section 2.4).

2.2 Learning Semantic Types of Source Attributes

When presented with a new source s whose semantic model is unknown, the first step is recognizing semantic types of its data. We formally define a semantic type to be either an ontology class $\langle class\_uri \rangle$ or a pair consisting of a domain class and one of its data properties $\langle class\_uri, property\_uri \rangle$. 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 domain/data property pair as a semantic type for attributes containing literal values. For example, the semantic type for the first attribute of s, *name*, is $\langle Person, name\rangle$. Semantic labeling is the task of assigning semantic types to source attributes.

While syntactic information about data sources such as attribute names or attribute types (string, int, date, ...) may give the system some hints to discover semantic types, they are often not sufficient, e.g., the name of the first field in the source s (Figure 2.1) is *name* and we do not know whether this is a name of a person, organization, book, or a movie. Moreover, in many cases, attribute names are used in abbreviated forms, e.g., *dob* rather than *birthdate*.

We employ the technique proposed by Krishnamurthy et al. [Krishnamurthy et al., 2015] to learn semantic types of source attributes. Their approach focuses on learning the semantic types from the data rather than the attribute names. It learns a semantic labeling function from a set of sources that have been manually labeled. When presented with a new source, the learned semantic labeling function can automatically assign semantic types to each attribute of the new source. The training data consists of a set of semantic types and each semantic type has a set of data values and attribute names associated with it. Given a new set of data values from a new source, the goal is to predict the top k candidate semantic types along with confidence scores using the training data.

If the data values associated with a source attribute a_i are textual data, the labeling algorithm uses the *cosine similarity* between TF/IDF vectors of the labeled documents and the input document to predict candidate semantic types. The set of data values associated with each textual semantic type in the training data is treated as a document, and the input document consists of the data values associated with a_i. For attributes with numeric data, the algorithm uses *statistical hypothesis testing* [Lehmann and Romano, 2005] to analyze the distribution of numeric values. The intuition is that the distribution of values in each semantic type is different. For example, the distribution of temperatures is likely to be different from the distribution of weights. The training data here consists of a set of numeric semantic types and each semantic type has a sample of numeric data values. At prediction time, given a new set of numeric data values (query sample), the algorithm performs statistical hypothesis tests between the query sample and each sample in the training data.

Once we apply this labeling method, it generates a set of candidate semantic types for each source attribute, each with a confidence value. Our algorithm then picks the top semantic type for each attribute as an input to the next step of the process. Thus, if the new source s has n attributes denoted by $attributes(s) = \{a_1, \cdots, a_n\}$, the output of the semantic labeling step is $types(s) = \{t_1, \cdots, t_n\}$ where t_i is the semantic type of a_i. For example, for s with $attributes(s) = \{name$, $birthdate$, $city$, $state$, $workplace\}$, we will have $types(s) = \{\langle Person, name\rangle$, $\langle Person, birthDate\rangle$, $\langle City, name\rangle$, $\langle State, name\rangle$, $\langle Organization, name\rangle\}$.

2.3 A Graph-Based Approach to Extract the Implicit Relationships

The next step in modeling a source is to represent the relationships between the source attributes in terms of the properties defined by the ontology. To extract the relationships, we construct a graph that defines the space of all possible mappings between the source and the ontology. At a high level, the nodes in the graph represent classes in the ontology, and the edges represent properties that relate these classes. The mapping from the ontology to the graph is not one-to-one given that, for example, several columns may contain instances of the same class.

2.3.1 Building A Graph from the Semantic Types and Domain Ontology

The central component of our method is a directed weighted graph $G = (V, E)$ built on top of the inferred semantic types $types(s)$ and expanded using the domain ontology O. Similar to a semantic model, G contains both class nodes (V_c) and data nodes (V_d) and we have $V = V_c \cup V_d$. The links in G correspond to the properties in O and are weighted.

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 for building the graph has three steps: adding semantic types, adding closure of the class nodes, and connecting the class nodes.

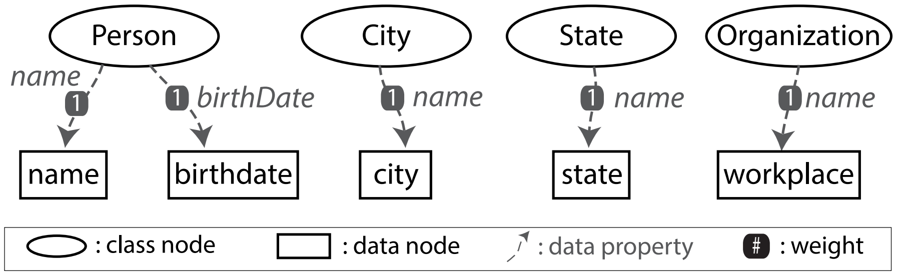


Figure 2.2: The graph G after adding the semantic types.

Adding Semantic Types: We start with an empty graph G. A semantic type, as aforementioned, is either $\langle class\_uri \rangle$ or $\langle class\_uri, property\_uri \rangle$. For each semantic type of the former type in $types(s)$, we add a class node u ($l_u = class\_uri$), a data node v ($l_v = attribute\_name$), and a link e from u to v with a weight equal to 1 and the label uri ($l_e =$ "uri", $w_e = 1$). For each semantic type of the latter kind, we add a class node u ($l_u = class\_uri$), a data node v ($l_v = attribute\_name$), and a link e with weight of 1 from u to v ($l_e = property\_uri$, $w_e = 1$). In both cases, if G already includes a class node with the same label, we do not create a new class node. For example, when adding the semantic type $\langle Person, birthdate \rangle$, we do not add the class node *Person*, because it has already been added to G after visiting the semantic type $\langle Person, name \rangle$. Figure 2.2 shows the graph G after adding the semantic types.

Adding the Closure of the Class Nodes: In addition to the nodes that are mapped from the semantic types, we find the nodes in the ontology that relate those semantic types. We search the ontology and create a class node in the graph for every class in the ontology having a path to the classes corresponding to the semantic types. In other words, for every class node v, we compute $closure(l_v)$ and then for each $class\_uri$ in $\bigcup_{v \in V_c} closure(l_v)$, we add a new class node to G with the label $class\_uri$. We ignore adding a class node if another class node with the same label already exists in G. In the example, after computing the closure of the class nodes, two new class nodes will be added to G: *Place* and *Event*. *Place* is in the closure of both *City* and *State*, and *Event* is in the closure of *Person*, *City*, and *State*.

Connecting the Class Nodes: The final step in constructing the graph is adding the links to express the relationships among the nodes. We connect two class nodes in the graph if there is an object property or $subClassOf$ relationship that connects their corresponding classes in the ontology. More precisely, for each pair of nodes $u, v \in V_c$, we compute $relations(l_u, l_v)$ from the ontology O, and then for each *property_uri* in the resulting set, a link with the label *property_uri* will be established from u to v in G.

To weight a link e from a class node u to another class node v, we look to the corresponding property in the ontology O. Let c_u and c_v be the classes in O that correspond to the class nodes u and v ($c_u = l_u$, $c_v = l_v$) and p be the property in O that corresponds to the link e ($p = l_e$).

- If p is an object property and it is a *direct property* from c_u to c_v, we assign a weight equal to 1 to the link e. The object property p is a direct object property from c_u to c_v if its definition in O explicitly declares c_u as domain and c_v as range.

- If p is an object property and it is an *inherited property* from c_u to c_v, we assign a weight equal to $1 + \epsilon$ to the link e. We say p is an inherited object property if it is not a direct object property but its domain contains one of the superclasses of c_u (at any level) and its range contains one of the superclasses of c_v (at any level). We assign a slightly higher weight to inherited properties, because we want to prioritize direct properties in the next step when we want to generate semantic models.

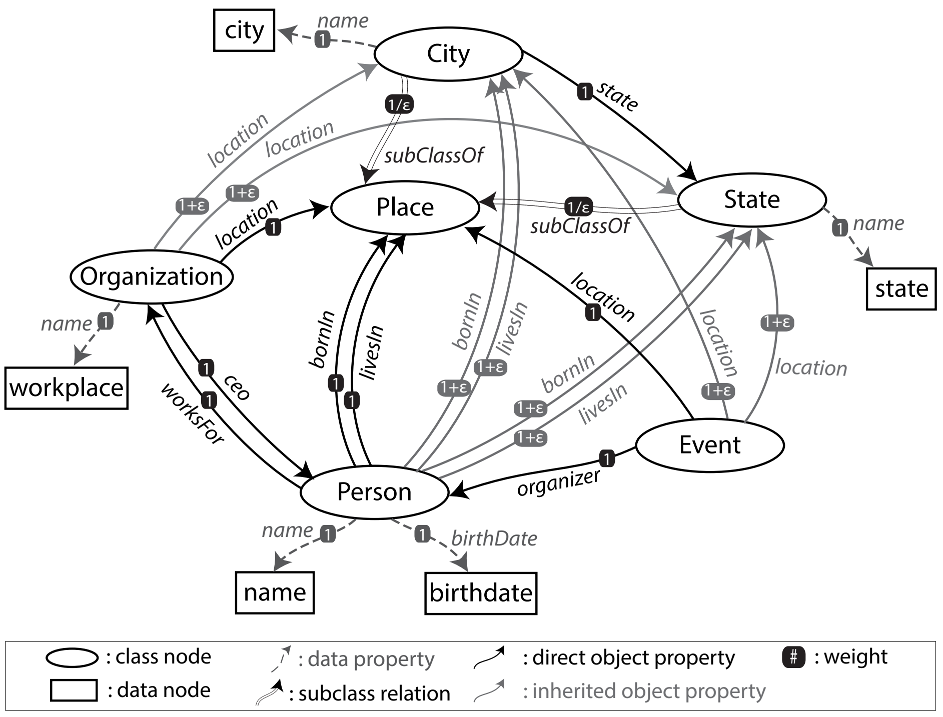


Figure 2.3: The graph G after adding the closure and connecting the class nodes. The properties *nearby* and *isPartOf* of the ontology are ignored to make the figure more readable.

- If p is a direct or inherited *rdfs:subClassOf* property from the c_u to c_v (c_u is subclass of c_v), we assign a weight equal to $1/\epsilon$ to e. Subclass links have a large weight so that relationships mapped from object properties will be preferred over the relationships through the class hierarchy.

In cases where G consists of more than one connected components, we add a class node with the label *owl:Thing* to the graph and connect the class nodes that do not have any parent to this root node using a *rdfs:subClassOf* link. This converts the original graph to a graph with only one connected component. The final graph is illustrated in Figure 2.3. Since this graph includes only one connected component, we do not need to add any class node with the label *owl:Thing*.

2.3.2 Generating an Initial Semantic Model

The graph we constructed explicitly represents all possible relationships among the semantic types. We construct a semantic model as the minimal tree that connects the semantic types. The minimal tree corresponds to the most succinct model that relates all the attributes in a source, and this is a good starting point for refining the model. To compute the minimal tree, we use one of the variations of the well-known Steiner Tree algorithm [Winter, 1987]. Given an edge-weighted graph and a subset of the vertices, called Steiner nodes, the goal is to find the minimum-weight tree in the graph that spans all Steiner nodes. In our graph, the Steiner nodes are the data nodes (V_d).

The general Steiner tree problem is NP-complete, however, there are several approximation algorithms [Winter, 1987; Takahashi and Matsuyama, 1980; Kou et al., 1981; Mehlhorn, 1988] that can be used to gain a polynomial runtime complexity. We use a heuristic algorithm [Kou et al., 1981] with an approximation ratio bounded by $2(1-1/l)$, where l is the number of leaves in the optimal Steiner tree. The time complexity of the algorithm is $O(|V_d||V|^2)$ in which V_d is the set of data nodes and V_c is the set of class nodes in G.

It is possible that multiple minimal trees exist, or that the correct interpretation of the data is specified by a non-minimal tree. For example, the output of the Steiner tree algorithm is the tree shown in Figure 2.4, which is not the correct interpretation of the data (Figure 2.1). In these cases, the user imposes constraints on the algorithms through a graphical user interface to create the correct semantic model.

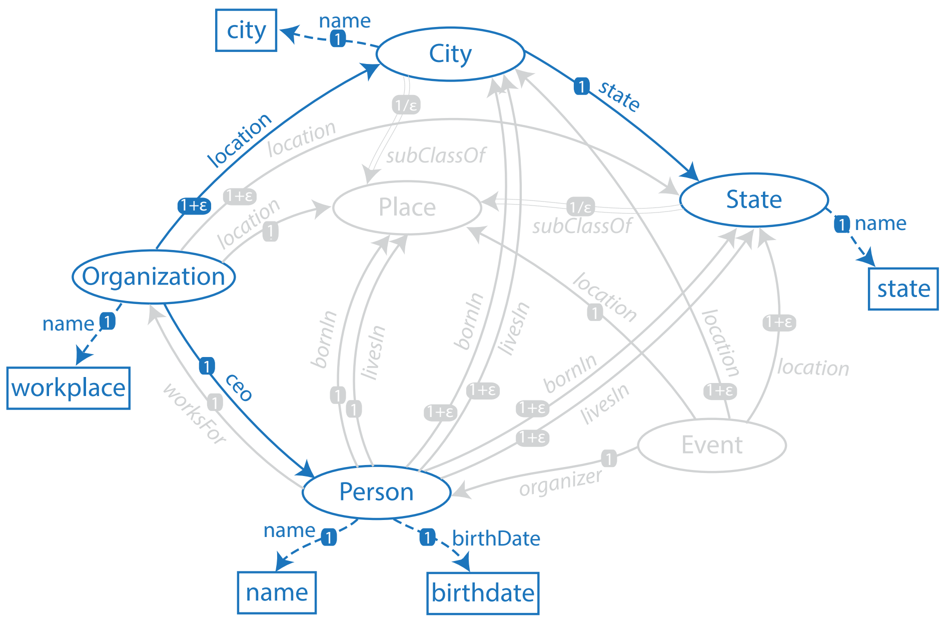
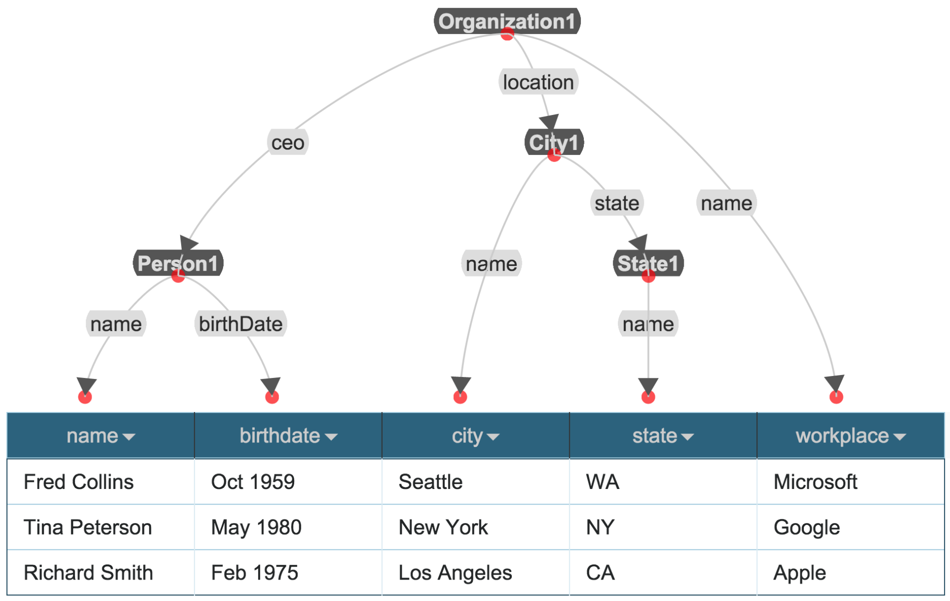


Figure 2.4: The initial Steiner tree (semantic model) suggested by the system (blue color). Our algorithm incorrectly suggests that *Person* is CEO of *Organization* and *City* is where *Organization* is located.

2.4 Refining Semantic Models

Our approach to model data sources is a semi-automatic approach. The user supervises different steps of the process to interactively build the desired semantic model. To bring the user in the loop, we integrated our approach into Karma[1] [Knoblock et al., 2012], a general information integration tool that supports many of the steps required for data integration. Karma allows the user to import data from a variety of sources including relational databases, spreadsheets, XML, and JSON. It provides support for cleaning and normalizing the data and integrating it using data integration operators such as join and union, and for publishing it in a variety of formats including RDF.

[1]`http://karma.isi.edu`



name	birthdate	city	state	workplace
Fred Collins	Oct 1959	Seattle	WA	Microsoft
Tina Peterson	May 1980	New York	NY	Google
Richard Smith	Feb 1975	Los Angeles	CA	Apple

Figure 2.5: The initial Steiner tree in Karma.

In Karma, the user initiates the modeling process by loading a data source and importing an ontology (the domain ontology). Karma shows the source data in a table and visualizes the semantic model as a tree of nodes displayed above the column headings (Figure 2.5). However, it is possible that the automatically suggested semantic model does not capture the correct meaning of the data. This is where the user interacts with the system and refines the model. During modeling a source, the user can interact with Karma in three ways:

Edit the semantic types: If the correct semantic type is not assigned to a source column, the user can browse the ontology through a user friendly interface to find the appropriate type. Karma automatically re-trains the classifier after this manual assignment. The system removes the class node and the data node corresponding to the old semantic type and adds new nodes to G. Then the Steiner tree is re-computed to generate a new semantic model.

In assigning or changing a semantic type, the user has the option to add a new instance of an ontology class that is already present in the model. In our example, the semantic labeling function has assigned $\langle Person, name\rangle$ and $\langle Person, birthDate\rangle$ to the first two attributes of s (*name* and *birthdate*). Thus, in the graph, we have two outgoing links from *Person* to *name* and *birthdate*, indicating that *name* and *birthdate* are different attributes of the same person. Now, consider the case in which the first two attributes of s are referring to different individuals. This means that in the semantic model, there should be two instances of the class *Person* that are separately connected to the data nodes *name* and *birthdate*. Assume that the user has already assigned the semantic type $\langle Person, name\rangle$ to the first column *name*. When assigning a semantic type to the second column *birthdate*, Karma allows the user to choose the same *Person* used to label the first column or choose a new instance of the class *Person*. If the user selects a new instance of *Person*, the system adds a new class node with the label *Person* to G, adds the new Person to the Steiner nodes, and replicates all the links of the existing *Person* for the new *Person*. Then, the Steiner tree is re-computed to produce the correct model.

Edit the links: Users can edit the model to adjust the relationships between the source columns. For instance, in the model shown in Figure 2.5, *Person* is related to *Organization* through the property *ceo*. However, in the correct model of the data, *Person* is connected to *Organization* using the property *worksFor*. To fix this, the user clicks on the source (*Organization*) or target (*Person*) of the wrong link (*ceo*) and Karma shows a list of possible relationships between the selected node and other nodes. For example, if the user clicks on the node *Person*, Karma extracts all the outgoing and incoming links of the class node *Person* from G and lists them in a menu. Then, the user can select a new link from the menu.

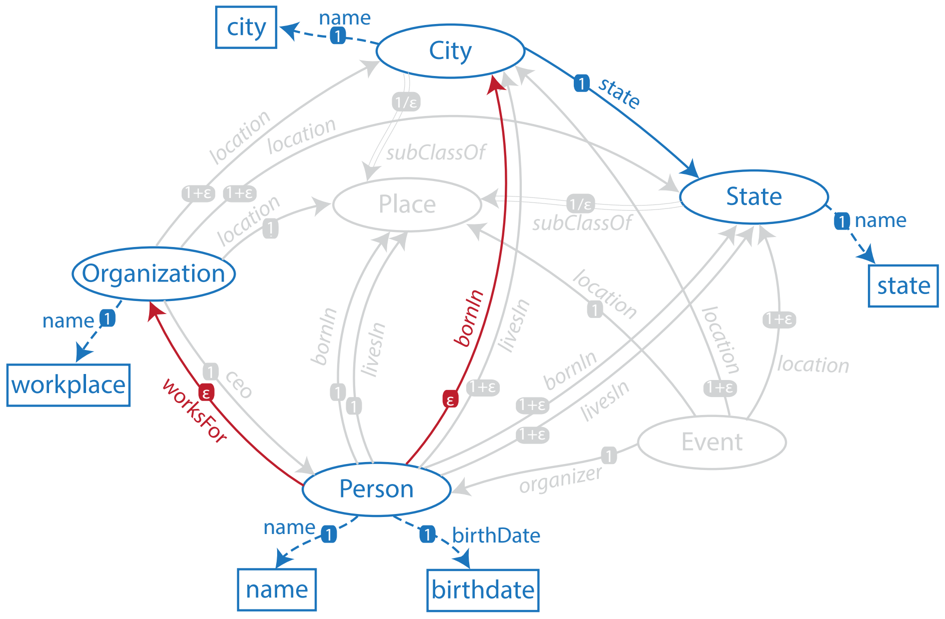
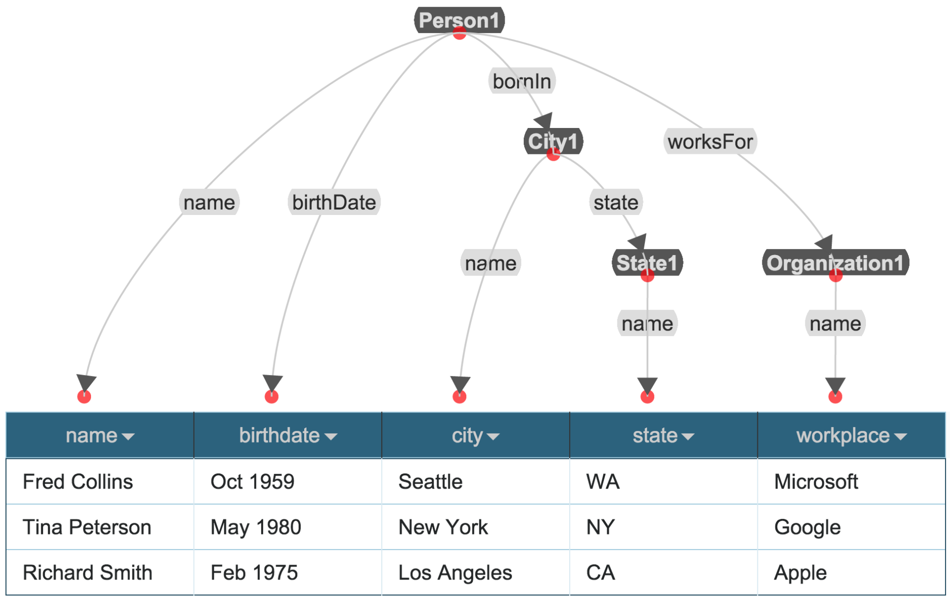


Figure 2.6: The user can refine the model by changing the relationships. The system changes the weights of the user-selected links (red links) to ϵ and re-computes the Steiner tree.

To force the Steiner tree algorithm to select the new link, we first add the source (*Person*) and the target (*Organization*) of the user-selected link (*worksFor*) to the Steiner nodes. Then, we reduce the weight of the user link to ϵ. These steps guarantee that the user link will be chosen by the Steiner tree algorithm. Figure 2.6 illustrates the new G and Steiner tree after adjusting the relationships by the user and Figure 2.7 shows the visualization of the final model in Karma.

2.5 Evaluation

We evaluated our approach on a dataset of 29 museum data sources in CSV, XML, or JSON format containing data from different art museums in the US. We used



name	birthdate	city	state	workplace
Fred Collins	Oct 1959	Seattle	WA	Microsoft
Tina Peterson	May 1980	New York	NY	Google
Richard Smith	Feb 1975	Los Angeles	CA	Apple

Figure 2.7: The visualization of the correct semantic model in Karma.

Karma to model these sources according to Europeana Data Model (EDM),[2] a well-known *data model* in the museum domain. A data model standardizes how to map the data elements in a domain to a set of domain ontologies. The domain ontologies that were used to model the sources are EDM[3], AAC[4], SKOS[5], Dublin Core Metadata Terms[6], FRBR[7], FOAF, ORE[8], and ElementsGr2[9]. Table 2.1 provides more information about the evaluation dataset. The dataset including

[2]`http://pro.europeana.eu/page/edm-documentation`

[3]`http://www.europeana.eu/schemas/edm`

[4]`http://www.americanartcollaborative.org/ontology`

[5]`http://www.w3.org/2008/05/skos#`

[6]`http://purl.org/dc/terms`

[7]`http://vocab.org/frbr/core.html`

[8]`http://www.openarchives.org/ore/terms`

[9]`http://rdvocab.info/ElementsGr2`

Table 2.1: The evaluation datasets.

number of data source	29
number of classes in the domain ontologies	119
number of properties in the domain ontologies	351
number of nodes in the gold-standard models	473
number of data nodes in the gold-standard models	329
number of class nodes in the gold-standard models	141
number of links in the gold-standard models	441

the sources, the domain ontologies, and the gold standard models are available on GitHub.[10] The source code of our approach is integrated into Karma which is available as open source.[11]

The objective of the evaluation was to assess the ability of our approach to produce semantic models for the given data sources. We measured effort in Karma by counting the number of user actions (number of menu choices to select correct semantic types or adjust paths in the graph) that the user had to perform to construct the correct semantic model for a source.

Table 2.2 shows the number of actions required to map all the data sources. The *Choose Type* column shows the number of times that the correct semantic type of a column was not present in the top four semantic types learned by our labeling function and we had to manually browse the ontology to select the correct semantic type. We started this evaluation with no training data for the semantic type identification. Out of the 56 manual assignments, 19 were for specifying semantic types that the system had never seen before, and 37 to fix incorrectly inferred types. The data sources were mapped in the order listed in the table, and

[10]`https://github.com/taheriyan/phd-thesis`

[11]`https://github.com/usc-isi-i2/Web-Karma`

Table 2.2: Evaluation results for modeling the evaluation dataset using Karma.

Source	**# Columns**	**# User Actions**			**Time (min)**
		Choose Type	Change Link	Total	
s1	7	7	1	8	3
s2	12	5	2	7	6
s3	4	0	0	0	2
s4	17	5	7	12	8
s5	14	4	6	10	7
s6	18	4	4	8	7
s7	14	1	4	5	6
s8	6	0	4	4	3
s9	4	1	0	1	2
s10	11	3	4	7	5
s11	6	1	0	1	2
s12	9	0	3	3	4
s13	10	1	3	4	4
s14	9	1	4	5	5
s15	13	0	3	3	5
s16	5	0	3	3	2
s17	12	0	4	4	4
s18	5	0	1	1	2
s19	17	0	6	6	5
s20	9	2	3	5	3
s21	28	8	8	16	11
s22	8	1	3	4	3
s23	18	0	4	4	5
s24	10	1	3	4	3
s25	13	1	4	5	4
s26	14	2	5	7	5
s27	12	2	3	5	4
s28	15	4	3	7	5
s29	9	2	1	3	3
Total	**329**	**56**	**96**	**152**	**128**
		Avg. # User Actions/Column = 152/329 = 0.46			

we can see that the number of Choose Type actions decreases as we move down the table. This suggests that Karma is able to learn semantic types from the examples it has seen before.

The *Change Link* column shows the number of times we had to select alternative relationships using a menu. Unlike learning the semantic types, our semi-automatic approach does not learn from the changes the user makes in the relationships. For example, in the first source, the Steiner tree algorithm suggests the link *aac:sitter* between *aac:CulturalHeritageObject* and *aac:Person*. The correct link between these two nodes in the gold standard model is *dcterms:creator*, and thus, the user has to change the initial link in Karma. However, our semi-automatic approach does not remember this adjustment. When the user starts modeling the second source, the system still proposes *aac:sitter* as the link from *aac:CulturalHeritageObject* to *aac:Person*. We address this problem in Section 3.

The whole process is done in a visual user interface, requiring 152 user actions, about 5.2 per data source and 0.46 per column. It took 128 minutes of interaction with Karma for a user familiar with the sources and the ontology to model all the sources. That means the user spent on average 4.4 minutes to create a semantic model for each source, a small effort compared to writing mapping rules by hand using standards such as R2RML [Das et al., 2012]. Moreover, manually writing semantic descriptions for sources requires some degree of expertise in Semantic Web technologies in addition to the domain knowledge. Karma plays a significant role by providing a user interface to rapidly build semantic models, making the complexity of formal semantic descriptions transparent to the user.

Chapter 3

Learning Semantic Models of Data Sources

In previous chapter, we presented a semi-automatic approach to interactively map a source to an ontology. The system uses learned semantic types and a Steiner tree algorithm to propose semantic models to the user, who can refine them as needed. One problem with this approach is that although it learns the new semantic types assigned by the user, it does not learn the refinements done by the user to adjust the relationships. Thus, it does not learn from the structure of previously modeled sources and always suggests a random minimal tree as the initial model of a new source, and most of the times user intervention is required to build the correct semantic model from the initially suggested model.

In this chapter, 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 [Taheriyan et al., 2013, 2014, 2015a]. 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 semantic models for new sources. The main contribution of our work are the techniques to leverage attribute relationships in known source models to hypothesize attribute relationships for new sources, and capturing them in semantic models.

3.1 Example

We explain the problem of learning semantic models by giving an example that will be used throughout this paper to illustrate different steps of our approach. In this example, the goal is to model a set of museum data sources using EDM, AAC, SKOS, Dublin Core Metadata Terms, FRBR, FOAF, ORE, and ElementsGr2 ontologies and then use the created semantic models to publish their data as RDF [Szekely et al., 2013]. Suppose that we have three data sources. The first source is a table containing information about artworks in the Dallas Museum of Art[1] (Figure 3.1a). We formally write the signature of this source as *dma(title, creationDate, name, type)* where *dma* is the name of the source and *title*, *creationDate*, *name*, and *type* are the names of the source attributes (columns). The second source, *npg*, is a CSV file including the data of some of the portraits in the National Portrait Gallery[2] (Figure 3.1b), and the third data source, *dia*, has the data of the artworks in the Detroit Institute of Art[3] (Figure 3.1c).

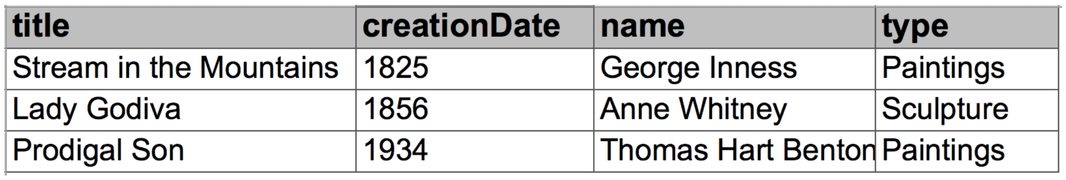
Figure 3.2 shows the correct semantic model of the sources *dma*, *npg*, and *dia* created by experts in the museum domain. The links in the semantic model are associated with ontology properties. The particular link *karma:uri* from a class node, which represents an ontology class, to a data node, which represents a source attribute, denotes that the attribute values are the URIs of the class instances. For instance, in Figure 3.2b, the values of the column *image* in the source *npg* are the URIs of the instances of the class *edm:WebResource*.

As discussed earlier, automatically building the semantic models is challenging. Machine learning methods can help us in assigning semantic types to the attributes

[1]`http://www.dma.org`

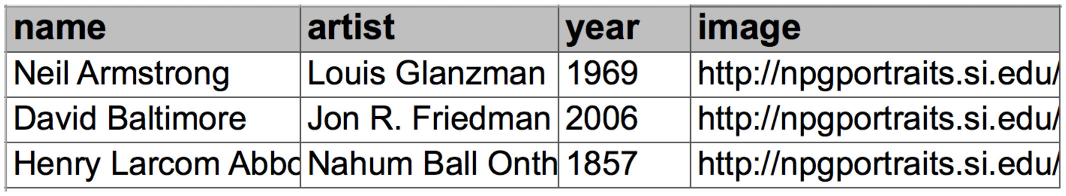
[2]`http://www.nationalportraitgallery.org`

[3]`http://www.dia.org`



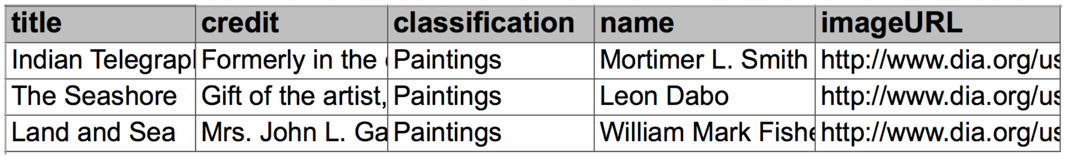
title	creationDate	name	type
Stream in the Mountains	1825	George Inness	Paintings
Lady Godiva	1856	Anne Whitney	Sculpture
Prodigal Son	1934	Thomas Hart Benton	Paintings

(a) *dma(title, creationDate, name, type)*



name	artist	year	image
Neil Armstrong	Louis Glanzman	1969	http://npgportraits.si.edu/
David Baltimore	Jon R. Friedman	2006	http://npgportraits.si.edu/
Henry Larcom Abbc	Nahum Ball Onth	1857	http://npgportraits.si.edu/

(b) *npg(name, artist, year, image)*

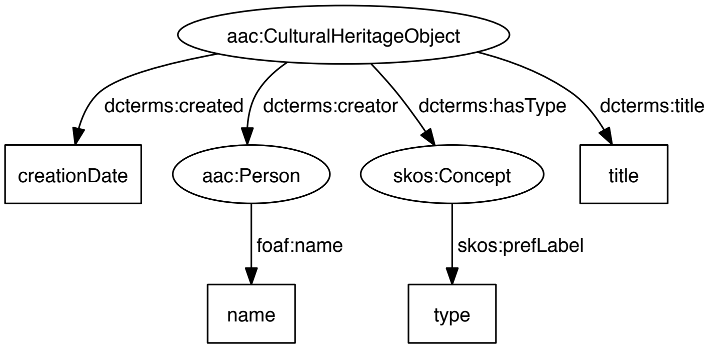


title	credit	classification	name	imageURL
Indian Telegrap	Formerly in the (	Paintings	Mortimer L. Smith	http://www.dia.org/us
The Seashore	Gift of the artist,	Paintings	Leon Dabo	http://www.dia.org/us
Land and Sea	Mrs. John L. Ga	Paintings	William Mark Fishe	http://www.dia.org/us

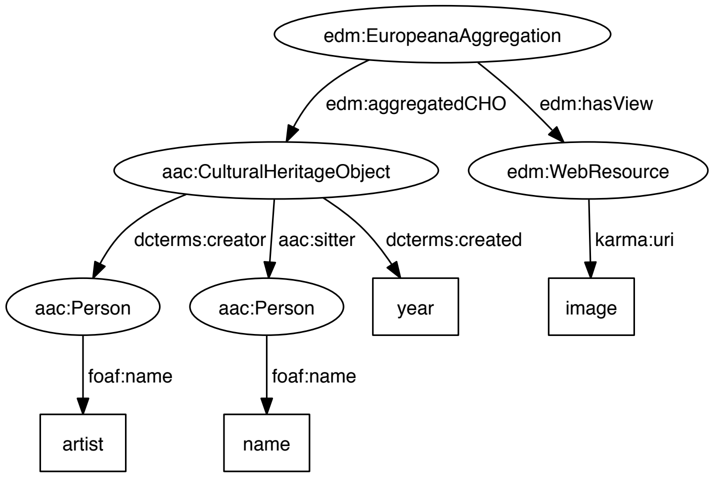
(c) *dia(title, credit, classification, name, imageURL)*

Figure 3.1: Sample data from three museum sources: (a) Dallas Museum of Art, (b) National Portrait Gallery, and (c) Detroit Institute of Art.

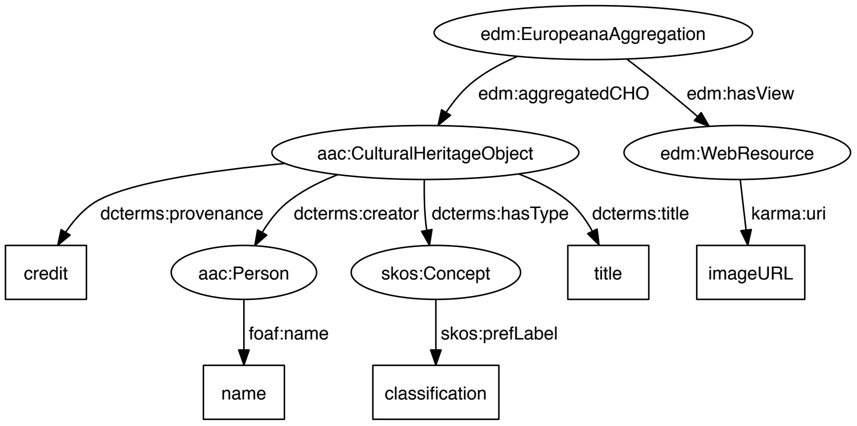
by looking into the attributes values, however, these methods are error prone when similar data values have different semantic types. For example, from just the data values of the attribute *creationDate* in the source *dma*, it is hard to say whether it is the creation date of *aac:CulturalHeritageObject* or it is the birthdate of a *aac:Person*. 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 captures the intended meaning of the data. For instance, there are several paths in the domain ontology connecting *aac:CulturalHeritageObject* to *aac:Person*, but in the context of the source *dma*, only the link *dcterms:creator* represents the correct meaning of the source. As another example, the attributes *artist* and *name* in the source *npg* are both labeled with name of *Person*, nevertheless, how can we decide whether these two attributes



(a) *sm(dma)*: semantic model of the source *dma*



(b) *sm(npg)*: semantic model of the source *npg*



(c) *sm(dia)*: semantic model of the source *dia*

Figure 3.2: Semantic models of the example data sources created by experts in the museum domain.

are different names of one person or they belong to two distinct individuals? In general, the ontology defines a large space of possible semantic models and without additional context, we do not know which one describes the source more precisely.

Now, assume that the correct semantic models of the sources *dma* and *npg* are given. Can we leverage these known semantic models to build a semantic model for a new source such as *dia*? In the next section, we present a scalable and automated approach that exploits the known semantic models *sm(dma)* and *sm(npg)* to limit the search space and learn a semantic model *sm(dia)* for the new source *dia*.

3.2 Learning Semantic Models

We now formally state the problem of learning semantic models of data sources. Let O be the domain ontology[4] 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 $s(a_1, a_2, \cdots, a_m)$ called the *target source*, in which $\{a_1, a_2, \cdots, a_m\}$ are the source attributes, our goal is to automatically compute a semantic model $sm(s)$ that captures the intended meaning of the source s. In our example, *sm(dma)* and *sm(npg)* are the known semantic models, and the source *dia* is the new source for which we want to automatically learn a semantic model.

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. One of the metrics helping us to infer relationships between the attributes of a new source is the popularity of the links between the semantic types in the set of known models. Nevertheless, simply using link popularity to connect a set of nodes would

[4]O can be a set of ontologies.

lead to myopic decisions that select links that appear frequently in other models without taking into account how these nodes are connected to other nodes in the given models. Suppose that we have a set of 5 known semantic models. One of these models contains the link *painter* between *Artwork* and *Person* and the link *museum* between *Artwork* and *Museum*. The other 4 models do not contain the type *Artwork*, but they include the link *founder* from *Museum* to *Person*. If a given new source contains the types *Artwork*, *Museum*, and *Person*, just using the link popularity yields to an incorrect model. Our approach takes into account the coherence of the patterns in addition to their popularity, and this is more complicated to do.

Our approach to learn a semantic model for a new source has four steps: (1) Using sample data from the new source, learn the semantic types of the source attributes. (2) Construct a graph from the known semantic models, augmented with nodes and links corresponding to the learned semantic types and ontology paths connecting nodes of the graph. (3) Compute the candidate mappings from the source attributes to the nodes of the graph. (4) Finally, build candidate semantic models for the candidate mappings, and rank the generated models.

3.2.1 Learning Semantic Types of Source Attributes

Semantic labeling is the first step to model a source, i.e., we label the source attributes with semantic types from the domain ontology. As we described in Section 2.2, a semantic type can be either an ontology class ⟨*class_uri*⟩ or a pair consisting of a data property and its domain class ⟨*class_uri,property_uri*⟩. 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 domain/data property

Table 3.1: Top two learned semantic types for the attributes of the source *dia*.

attribute	**candidate semantic types**
title	$\langle$*aac:CulturalHeritageObject,dcterms:title*$\rangle^{0.49}$ $\langle$*aac:CulturalHeritageObject,rdfs:label*$\rangle^{0.28}$
credit	$\langle$*aac:CulturalHeritageObject,dcterms:provenance*$\rangle^{0.83}$ $\langle$*aac:Person,ElementsGr2:note*$\rangle^{0.06}$
classification	$\langle$*skos:Concept,skos:prefLabel*$\rangle^{0.58}$ $\langle$*skos:Concept,rdfs:label*$\rangle^{0.41}$
name	$\langle$*aac:Person,foaf:name*$\rangle^{0.65}$ $\langle$*foaf:Person,foaf:name*$\rangle^{0.32}$
imageURL	$\langle$*foaf:Document*$\rangle^{0.47}$ $\langle$*edm:WebResource*$\rangle^{0.40}$

pair as a semantic type for attributes containing literal values. For example, the semantic types of the attributes *imageURL* and *classification* in the source *dia* are respectively $\langle$*edm:WebResource*$\rangle$ and $\langle$*skos:Concept,skos:prefLabel*$\rangle$.

To learn semantic types of source attributes, we use the same method that we used in our semi-automatic approach in previous chapter (Section 2.2). The only difference is that we take into account the uncertainty of the machine learning algorithm when labeling the source attributes. Our approach in previous chapter only uses the semantic type with the highest confidence value and ignores the other suggested semantic types. This is a strong assumption, because in many cases, the learning algorithm cannot distinguish the types of the source attributes that have similar data values, e.g., *birthDate* and *deathDate*. To overcome this limitation, we consider the top k candidate semantic types for each attribute rather than the top one semantic type per attribute. Thus, the output of the labeling step for $s(a_1, a_2, \cdots, a_m)$ is $T = \{(t_{11}^{p_{11}}, \cdots, t_{1k}^{p_{1k}}), \cdots, (t_{m1}^{p_{m1}}, \cdots, t_{mk}^{p_{mk}})\}$, 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. Table 3.1 lists the candidate semantic types for the source *dia* considering *k=2*.

As we can see in Table 3.1, the semantic labeling method prefers $\langle$*foaf:Document*$\rangle$ for the semantic type of the attribute *imageURL*, while according to the correct model (Figure 3.2c), $\langle$*edm:WebResource*$\rangle$ 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.

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

So far, we have tagged the attributes of *dia* with a set of candidate 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 candidate 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. This graph is different than the graph we built in our semi-automatic approach (Section 2.3.1) in the sense that this graph incorporates the known semantic models in addition to the learned semantic types and the domain ontology. Algorithm 3.1 shows the steps to build the graph. The algorithm has three parts: (1) adding the known semantic models, $sm(dma)$ and $sm(npg)$ (Algorithm 3.2); (2) adding the semantic types learned for the target source (Algorithm 3.3); and (3) expanding the graph using the domain ontology O (Algorithm 3.4).

Adding Known Semantic Models: Suppose that we want to add $sm(s_i)$ to the graph. If the graph is empty, we simply add all the nodes and links in $sm(s_i)$

```
Algorithm 3.1 Construct Graph G=(V,E)
    Input:
        - Known Semantic Models $M = \{sm_1, \cdots, sm_n\}$,
        - Attributes(s) $A = \{a_1, \cdots, a_m\}$
        - Semantic Types $T = \{(t_{11}^{p_{11}}, \cdots, t_{1k}^{p_{1k}}), \cdots, (t_{m1}^{p_{m1}}, \cdots, t_{mk}^{p_{mk}})\}$
        - Ontology $O$
    Output: Graph $G = (V, E)$

1: AddKnownModels($G$,$M$)
2: AddSemanticTypes($G$,$T$)
3: AddOntologyPaths($G$,$O$)

    return $G$
```

to G, otherwise we merge the nodes and links of $sm(s_i)$ into G by adding the nodes and links that do not exist in G. When adding a new node or link, we tag it with a unique identifier (e.g., s_i, name of the source) indicating that the node/link exist in $sm(s_i)$. If a node or link already exists in the graph, we just add the identifier s_i to its tags. The nodes and the links that are added in this step are shown with the black color in Figure 3.3. In order to easily refer to the nodes of the figure in the text, we assign a unique name to each node. The name of a node is written with small font at the left side of the node. For example, the node with the label *edm:EuropeanaAggregartion* is named n_1. The orange and green tags below the labels of the black links are the identifiers indicating the semantic model(s) supporting the links. For instance, the link *dcterms:creator* from n_2 (*aac:CulturalHeritageObject*) to n_7 (*aac:Person*) is tagged with both *dma* and *npg*, because it exists in both $sm(dma)$ and $sm(npg)$. For readability, we have not put the tags of the nodes in Figure 3.3.

Although merging a semantic model into G looks straightforward, there are difficulties when the semantic model or the graph include multiple class nodes

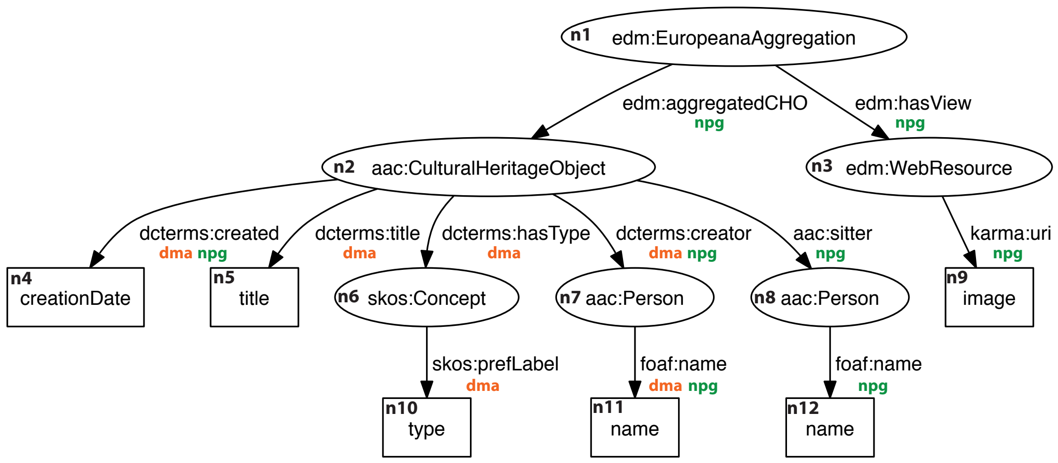


Figure 3.3: The graph G after adding the known semantic models $sm(dma)$ and $sm(npg)$.

with the same label. Suppose that G already includes two class nodes v_1 and v_2 both labeled with *Person* connected by the link *isFriendOf*. Now, we want to add a semantic model including the link *worksFor* from *Person* to *Organization*. Assuming G does not have a class node with the label *Organization*, we add a new class node v_3 to G. Now, the question is where to put the link *worksFor*, between v_1 and v_3, or v_2 and v_3. One option is to duplicate the link by adding a link between each pair and then assign different tags to the added links. This approach slows down the process of building the graph, and because it can yield a graph with a large number of links, our algorithm to compute the candidate semantic models would be inefficient too. Therefore, we adopt a different strategy; if there is more than one node in the graph matching a node in the semantic model, we select the one having more tags. This heuristic creates a more compact graph and makes the whole algorithm faster, while not having much impact on the results.

Algorithm 3.2 illustrates the details of our method to add a semantic model $sm(s_i)$ to G:

Algorithm 3.2 *Add Known Semantic Models to G*

```
1: function AddKnownModels($G$,$M$)
2:     $H = HashMap\langle node, node\rangle$
       ▷ H keys: nodes in $sm_i$
       ▷ H values: matched nodes in $G$
3:     for each $sm_i \in M$ do
4:         for each class node $v$ in $sm_i$ do
5:             $l_v \leftarrow$ label of $v$
6:             $c_1 \leftarrow$ number of class nodes in $sm_i$ with label $l_v$
7:             $c_2 \leftarrow$ number of class nodes in $G$ with label $l_v$
8:             add $(c_1 - c_2)$ class nodes with label $l_v$ to $G$
9:             $matched\_nodes \leftarrow$ class nodes with label $l_v$ in $G$
10:            $unmapped\_nodes \leftarrow matched\_nodes - values(H)$
11:            $v' \leftarrow$ the node with largest tag set in $unmapped\_nodes$
12:            add $\langle v, v'\rangle$ to $H$
13:        end for
14:        for each link $e$ from a class node $u$ to a data node $v$ in $sm_i$ do
15:            $l_e \leftarrow$ label of $e$
16:            $u' \leftarrow H(u)$
17:            if $u'$ has an outgoing link with label $l_e$ then
18:                $v' \leftarrow$ target of the link with label $l_e$
19:            else
20:                add a new data node $v'$ to $G$
21:            end if
22:            add $\langle v, v'\rangle$ to $H$
23:        end for
24:        $w_l \leftarrow 1$
25:        for each link $e$ from $u$ to $v$ in $sm_i$ do
26:            $u' \leftarrow H(u)$
27:            $v' \leftarrow H(v)$
28:            if there is $e'$ from $u'$ to $v'$ with $l_{e'} = l_e$ in $G$ then
29:                $tags_{e'} \leftarrow tags_{e'} \cup sm_i$
30:                $weight(e') \leftarrow w_l - |tags_{e'}|/(i+1)$
31:            else
32:                add the link $e'$ from $u'$ to $v'$ with $l_{e'} = l_e$ to $G$
33:                $tags_{e'} \leftarrow sm_i$
34:                $weight(e') \leftarrow w_l$
35:            end if
36:        end for
37:    end for
38: end function
```

1. [line 2] Let H be a HashMap keeping the mappings from the nodes in $sm(s_i)$ to the nodes in G. The key of each entry in H is a node in $sm(i)$ and its value is a node in G.

2. [lines 3-13]: For each class node v in $sm(s_i)$, we search the graph to see if G includes a class node with the same label. If no such node exists in the graph, we simply add a new node to the graph. It is possible that $sm(s_i)$ contains

multiple class nodes with the same label, for instance, a model including the link *isFriendOf* from one *Person* to another *Person*. In this case, we make sure that G also has at least the same number of class nodes with that label. For example, if G only has one *Person*, we add another class node with the label *Person*. Once we added the required class nodes to the graph, we map the class nodes in the model to the class nodes in the graph. If G has multiple class nodes with the same label, we select the one that is tagged by larger number of known semantic models. We add an entry to H with v as the key and the mapped node (v') as the value.

3. [lines 14-23]: For each link $e = (u, v)$ in $sm(s_i)$ where v is a data node, we search the graph to see if there is a match for this pattern. We first use H to find the node u' in G to which the node u is mapped. If u' does not have any outgoing link with a label equal to the label of e, we add a new data node v' to G. We add v and its mapped data node v' to H.

4. [lines 24-37]: For each link $e = (u, v)$ in $sm(s_i)$, we find the nodes in G to which u and v are mapped (say u' and v'). If G includes a link with the same label as the label of e between u' and v', we only add s_i to the tags associated with the link. Otherwise, we add a new link to the graph and tag it with s_i.

Adding Semantic Types: Once the known semantic models are added to G, we add the semantic types learned for the attributes of the target source. As mentioned before, we have two kinds of semantic types: ⟨*class_uri*⟩ for attributes whose data values are URIs and ⟨*class_uri,property_uri*⟩ for attributes that have literal data. For each learned semantic type t, we search the graph to see whether G includes a *match* for t.

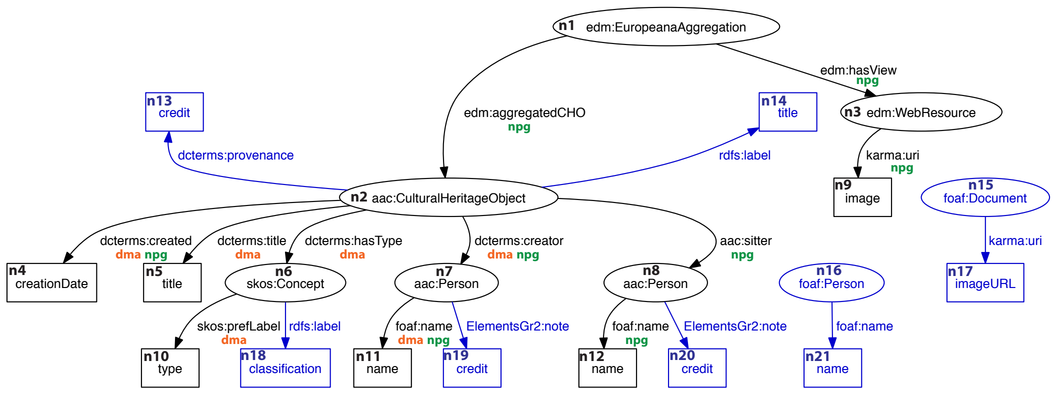


Figure 3.4: The graph G after adding the nodes and the links corresponding to the semantic types (shown in blue).

- t=⟨*class_uri*⟩: We say (u, v, e) is a match for t 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 *karma:uri*. For example, in Figure 3.3, (n_3,n_9,*karma:uri*) is a match for the semantic type ⟨*edm:WebResource*⟩.

- t=⟨*class_uri,property_uri*⟩: We say (u, v, e) is a match for t if u is a class node labeled with *class_uri*, v is a data node, and e is a link from u to v labeled with *property_uri*. In Figure 3.3, (n_6,n_{10},*skos:prefLabel*) is a match for the semantic type ⟨*skos:Concept,skos:prefLabel*⟩.

We say t=⟨*class_uri*⟩ or t=⟨*class_uri,property_uri*⟩ 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 ⟨*skos:Concept,rdfs:label*⟩ only has a partial match in G, because G contains a class node labeled with *skos:Concept* (n_6), but this class node does not have an outgoing link with the label *rdfs:label*.

Algorithm 3.3 shows the function that adds the learned semantic types to the graph G. 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

Algorithm 3.3 *Add Semantic Types to G*

```
function AddSemanticTypes($G$,$T$)

    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
            if $t_{ij} = \langle class\_uri \rangle$ then
                $l_v \leftarrow class\_uri$
                $l_e \leftarrow$ "$karma\colon uri$"
            else if $t_{ij} = \langle class\_uri, property\_uri \rangle$ then
                $l_v \leftarrow class\_uri$
                $l_e \leftarrow property\_uri$
            end if
            if no node in $G$ has the label $l_v$ then
                add a new node $v$ with the label $l_v$ in $G$
            end if
            $V_{match} \leftarrow$ all the class nodes with the label $l_v$
            $w_h \leftarrow |E|$
            for each $v \in V_{match}$ do
                if $v$ does not have an outgoing link labeled $l_e$ then
                    add a data node $w$ with the label $a_i$ to $G$
                    add a link $e = (v, w)$ with the label $l_e$
                    $weight(e) \leftarrow w_h$
                end if
            end for
        end for
    end for

end function
```

matches. Consider the semantic types learned for the source *dia* (Table 3.1). Figure 3.4 illustrates the graph G after adding the semantic types. The nodes and the links that are added in this step are depicted with the blue color. For ⟨*aac:CulturalHeritageObject, dcterms:title*⟩, we do not need to change G, because the graph already contained one match: *(*n_2*,*n_5*,dcterms:title).* The semantic type ⟨*skos:Concept,rdfs:label*⟩ only had one partial match (n_6), thus, we add one data node (n_{18} with a label equal to the name of the corresponding attribute) and one link (*rdfs:label* from n_6 to n_{18}) in order to complete the existing partial match.

The semantic type ⟨*foaf:Document*⟩ had neither a match nor a partial match. We add a class node (n_{15}), a data node (n_{17}), and a link between them (*karma:uri* from n_{15} to n_{17}) to create a match.

Adding Paths from the Ontology: We use the domain ontology to find all the paths that relate the current class nodes in G (Algorithm 3.4). The goal is to connect class nodes of G using the direct paths or the paths inferred through the subclass hierarchy in O. The final graph is shown in Figure 3.5. We connect two class nodes in the graph if there is an object property or *subClassOf* relationship that connects their corresponding classes in the ontology. For instance, in Figure 3.5, there is the link *ore:aggregates* from n_1 to n_2. This link is added because the object property *ore:aggregates* is defined with *ore:Aggregation* as domain and *ore:AggregatedResource* as range, and *edm:EuropeanaAggregation* is a subclass of the class *ore:Aggregation* and *aac:CulturalHeritageObject* is a subclass of *edm:ProvidedCHO*, which is in turn a subclass of the class *ore:AggregatedResource*. As another example, the reason why n_1 is connected to n_{15} is that the property *foaf:page* is defined from *owl:Thing* to *foaf:Document* in the FOAF ontology. Thus, a link with the label *foaf:page* would exist from each class node in G to n_{15} since all classes are subclasses of the class *owl:Thing*. Depending on the size of the ontology, many nodes and links may be added to the graph in this step. To make the figure readable, only a few of the added nodes and links are illustrated in Figure 3.5 (the ones with the red color).

In cases where G consists of disconnected components, we add a class node with the label *owl:Thing* to the graph and connect the class nodes that do not have any parent to this root node using a *rdfs:subClassOf* link. This converts the original graph to a graph with only one connected component.

Algorithm 3.4 *Add Ontology Paths to G*

```
1: function AddOntologyPaths($G$,$O$)

2:     for each pair of class nodes $u$ and $v$ in $G$ do
3:         $c_1 \leftarrow$ ontology class with $uri = l_u$
4:         $c_2 \leftarrow$ ontology class with $uri = l_v$
5:         $P_{(c_1,c_2)} \leftarrow$ all the direct and inferred properties (including
       rdfs:subClassOf) from $c_1$ to $c_2$ in $O$
6:         $w_h \leftarrow |E|$
7:         for each property $p \in P$ do
8:             $l_e \leftarrow uri$ of the property $p$
9:             if there is no link with label $l_e$ from $u$ to $v$ then
10:                add a link $e = (u, v)$ with label $l_e$ to $G$
11:                $weight(e) \leftarrow w_h$
12:            end if
13:        end for
14:    end for

15: end function
```

The links in the graph G are weighted. Assigning weights to the links of the graph is very important in our algorithm. We can divide the links in G into two categories. The first category includes the links that are associated with the known semantic models (black links in Figure 3.5). The other group consists of the links added from the learned semantic types or the ontology (blue and red links) which are not tagged with any identifier. The basis of our weighting function is to assign a much lower weight to the links in the former group compared to the links in the latter group. If w_l is the default weigh of a link in the first group and w_h is the default weight of a link in the second group, we will have $w_l \ll w_h$. The intuition behind this decision is to produce more coherent models in the next step when we are generating minimum-cost semantic models (Section 3.2.4). Our goal is to give more priority to the models containing larger segments from the known patterns. One reasonable value for w_h is $w_l * |E|$ in which $|E|$ is the number of links in G.

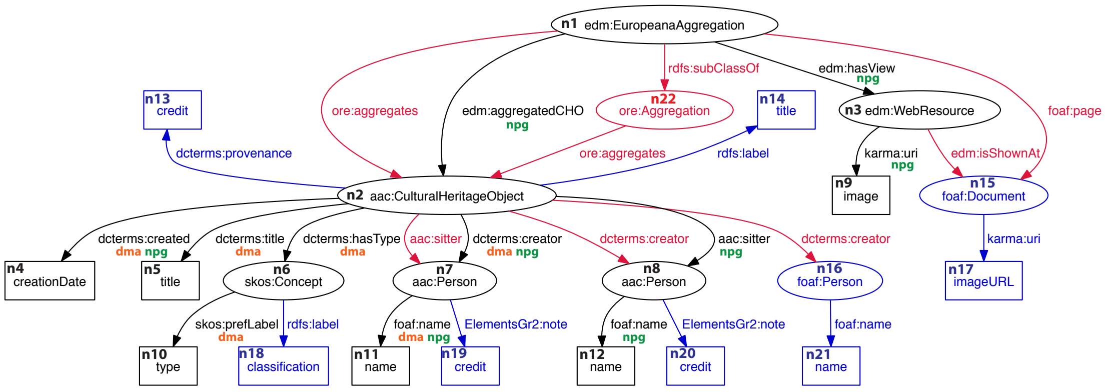


Figure 3.5: The final graph G after adding the paths from the domain ontologies. For legibility, only a few of all the possible paths between the class nodes are shown (drawn with the red color).

This formula ensures that even a long pattern from a known semantic model will cost less than a single link that does not exist in any known semantic model.

One factor that we consider in weighting the links coming from the known semantic models (black links) is the popularity of the links, i.e., the number of known semantic models supporting that link. We assign $(w_l - x/(n+1))$ to each black link where n is the number of known semantic models and x is the number of identifiers the link is tagged with. Suppose that we use $w_l = 1$ in our example. Since our graph in Figure 3.5 has a total of 26 links, we will have $w_h = w_l * |E| = 26$. In Figure 3.5, the link *edm:hasView* from n_1 to n_3 will be weighted with 0.66 because it is only supported by $sm(npg)$ (n=*2*, x=*1*). The weight of the link *dcterms:creator* from n_2 to n_7 will be 0.33 since both $sm(dma)$ and $sm(npg)$ contain that link (the link has two tags).

We assign w_h to the links that are not associated with the known models (blue and red links, which do not have a tag). There is only a small adjustment for the links coming from the ontology (red links). We prioritize direct properties over inherited properties by assigning a slightly higher weight $(w_h + \epsilon)$ to the inherited ones. The rationale behind this decision comes from this observation that the direct properties (more specific) are more likely to be used in the semantic models than the inherited properties (more general). For instance, the red link *aac:sitter* from n_2 to n_7 will be weighted with $w_h = 26$, because its definition in the ontology AAC has *aac:CulturalHeritageObject* as domain and *aac:Person* as range. In other hand, the weight of the link *ore:aggregates* from n_1 to n_2 will be 26.01 (assume $\epsilon = 0.01$) since the domain of *ore:aggregates* in the ontology ORE is the class *ore:Aggregation* (which is a superclass of *edm:EuropeanaAggregation*) and its range is the class *ore:AggregatedResource* (which is a superclass of *aac:CulturalHeritageObject*).

3.2.3 Mapping Source Attributes to the Graph

We use the graph built in the previous step to infer the relationships between the source attributes. First, we find mappings from the source attributes to a subset of the nodes of the graph. Then, we use these mappings to generate and rank candidate semantic models. In this section, we describe the mapping process, and in Section 3.2.4, we talk about computing candidate semantic models.

To map the attributes of a source to the nodes of G (Figure 3.5), we search G to find the nodes matching the semantic types associated with the attributes. For example, the attribute *classification* in *dia* maps to $\{n_6, n_{10}\}$ and $\{n_6, n_{18}\}$, corresponding to the semantic types ⟨*skos:Concept,skos:prefLabel*⟩ and ⟨*skos:Concept, rdfs:label*⟩, respectively.

Since each attribute has been annotated with k semantic types and also each semantic type may have more than one match in G (e.g., ⟨*aac:Person,foaf:name*⟩ maps to $\{n_7, n_{11}\}$ and $\{n_8, n_{12}\}$), more than one mapping m might exist from the source attributes to the nodes of G. Generating all the mappings is not feasible in cases where we have a data source with many attributes and the learned semantic types have many matches in the graph. The problem becomes worse when we generate more than one candidate semantic type for each attribute. Suppose that we are modeling the source s consisting of n attributes and we have generated k semantic types for each attribute. If there are r matches for each semantic type, we will have $(k * r)^n$ mappings from the source attributes to the nodes of G.

We present a heuristic 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 most promising mappings. The algorithm scores the mappings after processing each attribute and removes the low score ones. Our scoring function takes into account the confidence values of the semantic types, the coherence of the nodes in the mappings, and the size of the mappings. The inputs to the algorithm are the learned semantic types $T = \{(t_{11}^{p_{11}}, \cdots, t_{1k}^{p_{1k}}), \cdots, (t_{m1}^{p_{m1}}, \cdots, t_{mk}^{p_{mk}})\}$ for the attributes of the source $s(a_1, a_2, \cdots, a_m)$ and the graph G, and the output is a set of candidate mappings m from the source attributes to a subset of the nodes in G. The key idea is that instead of generating all the mappings (which is not feasible), we score the partial mappings after processing each attribute and prune the mappings with lower scores. In other words, 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 (*branching factor*) after mapping each attribute.

Algorithm 3.5 shows our mapping process. The heart of the algorithm is the scoring function we use to rank the partial mappings (line 22 in Algorithm 3.5). We compute three functions for each mapping m: *confidence*(m), *coherence*(m), and *sizeReduction*(m). Then, we calculate the final score *score*(m) by combining the values of these three functions. We explain these functions using an example. Suppose that the maximum number of the mappings we expand in each step is

Algorithm 3.5 *Generate Candidate Mappings*

```
Input:
        - $G(V, E)$,
        - $attributes(s) = \{a_1, \cdots, a_m\}$
        - $T = \{(t_{11}^{p_{11}}, \cdots, t_{1k}^{p_{1k}}), \cdots, (t_{m1}^{p_{m1}}, \cdots, t_{mk}^{p_{mk}})\}$
        - $branching\_factor$: max number of mappings to expand
        - $num\_of\_candidates$: number of candidate mappings
Output: a set of candidate mappings $m$ from $attributes(s)$ to $S \subset V$

 1: $mappings \leftarrow \{\}$
 2: $candidates \leftarrow \{\}$
 3: for each $a_i \in attributes(s)$ do
 4:     for each $t_{ij}^{p_{ij}} \in (t_{i1}^{p_{i1}}, \cdots, t_{ik}^{p_{ik}})$ do
 5:         $matches \leftarrow$ all the $(u, v, e)$ in $G$ matching $t_{ij}$
 6:         if $mappings = \{\}$ then
 7:             for each $(u, v, e) \in matches$ do
 8:                 $m \leftarrow (\{a_i\} \rightarrow \{u, v\})$
 9:                 $mappings \leftarrow mappings \cup m$
10:             end for
11:         else
12:             for each $m : X \rightarrow Y \in mappings$ do
13:                 for each $(u, v, e) \in matches$ do
14:                     $m' \leftarrow (X \cup \{a_i\} \rightarrow Y \cup \{u, v\})$
15:                     $mappings \leftarrow mappings \cup m'$
16:                 end for
17:                 remove $m$ from $mappings$
18:             end for
19:         end if
20:     end for
21:     if $|mappings| > branching\_factor$ then
22:         compute $score(m)$ for each $m \in mappings$
23:         sort items in $mappings$ descending based on their score
24:         keep top $branching\_factor$ mappings and remove others
25:     end if
26: end for
27: $candidates \leftarrow$ top $num\_of\_candidates$ items from $mappings$
    return $candidates$
```

2 (*branching_factor* = *2*). After mapping the second attribute of the source *dia* (*credit*), we will have: $mappings = \{$

$m_1 : \{title, credit\} \rightarrow \{(n_2, n_5), (n_2, n_{13})\},$

$m_2 : \{title, credit\} \rightarrow \{(n_2, n_5), (n_7, n_{19})\},$

$m_3 : \{title, credit\} \rightarrow \{(n_2, n_5), (n_8, n_{20})\},$

$m_4 : \{title, credit\} \rightarrow \{(n_2, n_{14}), (n_2, n_{13})\}$,

$m_5 : \{title, credit\} \rightarrow \{(n_2, n_{14}), (n_7, n_{19})\}$,

$m_6 : \{title, credit\} \rightarrow \{(n_2, n_{14}), (n_8, n_{20})\}$

}

There are two matches for the attribute *title*: (n_2, n_5) for the semantic type ⟨*aac:CulturalHeritageObject, dcterms:title*⟩ and (n_2, n_{14}) for the semantic type ⟨*aac: CulturalHeritageObject,rdfs:label*⟩; and three matches for the attribute *credit*: (n_2, n_{13}) for the semantic type ⟨*aac:CulturalHeritageObject, dcterms:provenance*⟩ and (n_7, n_{19}) and (n_8, n_{20}) for the semantic type ⟨*aac:Person, ElementsGr2:note*⟩. This yields $2 * 3 = 6$ different mappings. Since *branching_factor* $= 2$, we have to eliminate four of these mappings. Now, we describe how the algorithm ranks the mappings.

Confidence: We define confidence as the arithmetic mean of the confidence values associated with a mapping. For example, m_1 is consisting of the matches for the semantic types ⟨*aac:CulturalHeritageObject, dcterms:title*⟩$^{0.49}$ and ⟨*aac: CulturalHeritageObject,dcterms:provenance*⟩$^{0.83}$. Thus, $confidence(m_1) = 0.66$.

Coherence: This function measures the largest number of nodes in a mapping that belong to the same known semantic model. Like the links, the nodes in G are also tagged with the model identifiers although we have not shown them in Figure 3.5. We calculate coherence as the maximum number of the nodes in a mapping that have at least one common tag. For instance, $coherence(m_1) = 0.66$ because two nodes out of the three nodes in m_1 (n_2 and n_5) are from $sm(dma)$, and $coherence(m_2) = 1.0$ because all the nodes of m_2 are from the same semantic model $sm(dma)$. The goal of defining the coherence is to give more priority to the models containing larger segments from the known patterns.

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

Score(m): The final score is the combination the values *confidence(m)*, *coherence(m)*, and *sizeReduction(m)*, which are all in the range $[0, 1]$. We assign a weight to each of these values and then compute the final score as the weighted sum of them: $\textit{score}(m) = w_1\ \textit{confidence}(m) + w_2\ \textit{coherence}(m) + w_3\ \textit{sizeReduction}(m)$, where w_1, w_2, and w_3 are the weights, decimal values in the range $[0, 1]$ summing up to 1. The proper values of the weights can be tuned by experiments. In our evaluation (Section 3.3), we obtained better results when all the three functions contributed equally to the final score. That is, $\textit{score}(m)$ is calculated as the arithmetic mean of $\textit{confidence}(m)$, $\textit{coherence}(m)$, and $\textit{sizeReduction}(m)$ ($w_1 = w_2 = w_3 = 1/3$).

In our example, if we use arithmetic mean to compute the final score, the scores of the 6 mappings we mentioned before are as follows: $\textit{score}(m_1) = 0.60$, $\textit{score}(m_2) = 0.42$, $\textit{score}(m_3) = 0.42$, $\textit{score}(m_4) = 0.46$, $\textit{score}(m_5) = 0.39$,

$score(m_6) = 0.39$. Therefore, m_2, m_3, m_5, and m_6 will be removed from the *mappings* (line 24), and the algorithm continues to the next iteration, which is mapping the next attribute of the source *dia* (*classification*) to the graph. At the end, we will have maximum *branching_factor* 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 (Algorithm 3.5 line 27).

3.2.4 Generating and Ranking Semantic Models

Once we generated candidate mappings from the source attributes to the nodes of the graph, we compute and rank candidate semantic models. To compute a semantic model for a mapping m, we find the minimum-cost tree in G that connects the nodes of m. The cost of a tree is the sum of the weights on its links. This problem is known as the Steiner Tree problem [Winter, 1987]. 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, however, there are several approximation algorithms [Winter, 1987; Takahashi and Matsuyama, 1980; Kou et al., 1981; Mehlhorn, 1988] that can be used to gain a polynomial runtime complexity.

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 as a candidate semantic model for the source. For example, for the source *dia* and the mapping *m:* {*title,credit, classification,name,imageURL*}$\rightarrow$ {(n_2,n_5),(n_2,n_{13}),(n_6,n_{10}),(n_7,n_{11}),(n_3,n_9)}, the resulting Steiner tree will be exactly as what is shown in Figure 3.2c, which is the correct semantic model of the source *dia*. The algorithm to compute the minimal tree prefers the links that appear in the known semantic models (links with

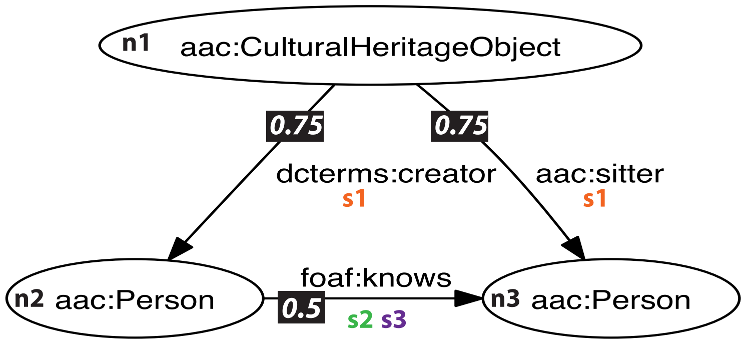


Figure 3.6: A small part of an example graph constructed using three known models.

tags) because they have a much lower weight than the other links in G. Additionally, since the weight of a link with tags has inverse relation with its number of tags (number of known semantic models containing the link), the semantic model obtained by computing the minimal tree will contain the links that are more popular in the known semantic models.

Selecting more popular links does not always yield the correct semantic model. Suppose that we have three known semantic models $\{sm(s_1), sm(s_2), sm(s_3)\}$. One of them connects *aac:CulturalHeritageObject* to two instances of *aac:Person* using the links *dcterms:creator* and *aac:sitter* (similar to $sm(npg)$). The other two semantic models do not contain the class node *aac:CulturalHeritageObject*, but they have two class nodes *aac:Person* connected using the link *foaf:knows*. Figure 3.6 shows a small part of the graph constructed using these known models. The black labels on the links represent the weights of the links. For instance, the link *dcterms:creator* from n_1 to n_2 has a weight equal to 0.75 because it is only supported by $sm(s_1)$ ($w_l - x/(n+1) = 1 - 1/(1+3) = 0.75$).

Now, assume that we have a new source s_4 with three attributes $\{a_1, a_2, a_3\}$ annotated with *aac:CulturalHeritageObject*, *aac:Person*, and *aac: Person*. Computing the minimal tree for the mapping *m:* $\{a_1, a_2, a_3\} \rightarrow \{n_1, n_2, n_3\}$ will result a tree that consists of the link *foaf:knows* between n_2 to n_3 and either *dcterms:creator* from n_1 to n_2 or *aac:sitter* between n_1 and n_3. Nonetheless, this is not the correct semantic model for the source. When s_4 includes *aac:CulturalHeritageObject* in addition to those two *aac:Person*, it is more likely that the source is describing the relations between the cultural heritage objects and the people and not the relations between the people.

We solve this problem by taking into account the coherence of the patterns. Instead of just the minimal Steiner tree, we compute the *top-k* Steiner trees and rank them first based on the coherence of their links and then their cost. In the example shown in Figure 3.6, the top-3 results assuming n_1, n_2, and n_3 as the Steiner nodes are:

T_1=\{$(n_1,n_3,aac{:}sitter),(n_2,n_3,foaf{:}knows)$\}

T_2=\{$(n_1,n_2,dcterms{:}creator),(n_2,n_3,foaf{:}knows)$\}

T_3=\{$(n_1,n_2,dcterms{:}creator),(n_1,n_3,aac{:}sitter)$\}

where $cost(T_1)=cost(T_2)=1.25$ and $cost(T_3)=1.5$. Once we computed the *top-k* trees, we sort them according to their coherence. The coherence here means the percentage of the links in the Steiner tree that are supported by the same semantic model. It is computed similar to the coherence of the nodes with the difference that we use the tags on the links instead of the tags on the nodes. In our example, the coherence of T_1 and T_2 will be 0.5 because their links do not belong to the same known semantic model, and the coherence of T_3 will be 1.0 since both of its links are tagged with s_1. Therefore, T_3 will be ranked higher than T_1 and T_2, although it has higher cost than T_1 and T_2.

We use a customized version of the BANKS algorithms [Bhalotia et al., 2002] to compute the *top-k* Steiner trees. The original BANKS algorithm is developed for the problem of the keyword-based search in relational databases, and because it makes specific assumptions about the topology of the graph, applying it directly to our problem eliminates some of the trees from the results. For instance, if two nodes are connected using two links with different weights, it only considers the one with the lower weight and it never generates a tree including the link with the higher weight. We customized the original algorithm to support more general cases.

The BANKS algorithm creates one iterator for each of the nodes corresponding to the semantic types, and then the iterators follow the incoming links to reach a common ancestor. The algorithm uses the iterator's distance to its starting point to decide which link should be followed next. Because our weights have an inverse relation with their popularity, the algorithm prefers more frequent links. To make the algorithm converge to more coherent models first, we use a heuristic that prefers the links that are parts of the same pattern (known semantic model) even if they have higher weights. Suppose that $sm_1 : v_1 \xrightarrow{e_1} v_2$ and $sm_2 : v_1 \xrightarrow{e_2} v_2 \xrightarrow{e_3} v_3$ are the only known models used to build the graph G, and the weight of the link e_2 is higher than e_1. Assume that v_1 and v_3 are the semantic labels. The algorithm creates two iterators, one starting from v_1 and one from v_3. The iterator that starts from v_3 reaches v_2 by following the incoming link $v_2 \xrightarrow{e_3} v_3$. At this point, it analyzes the incoming links of v_2 and although e_1 has lower weight than e_2, it first chooses e_2 to traverse next. This is because e_2 is part of the known model sm_2 which includes the previously traversed link e_3.

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 top semantic types are not the correct ones. For example, for the source *dia*, the mapping *m:* $\{$*title,credit, classification,name,imageURL*$\} \rightarrow \{(n_2,n_5),(n_2,n_{13}),(n_6,n_{10}),(n_7,n_{11}),(n_3,n_9)\}$, which maps the attribute *imageURL* to (n_3, n_9) using the type ⟨*edm:WebResource*⟩, will be scored higher than the mapping *m′:* $\{$*title,credit,classification,name,imageURL*$\}$ $\rightarrow \{(n_2,n_5),(n_2,n_{13}),(n_6,n_{10}),(n_7,n_{11}),(n_{15},n_{17})\}$, which maps *imageURL* to the nodes (n_{15}, n_{17}) using the semantic type ⟨*foaf:Document*⟩. The mapping m has lower confidence value than m', but is 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.

3.3 Evaluation

We evaluated our approach on two datasets, each including a set of data sources and a set of domain ontologies that will be used to model the sources. Both of these datasets have the same set of data sources, 29 museum sources in CSV, XML, or JSON format containing data from different art museums in the US, however, they include different domain ontologies. The goal is to learn the semantic models of the data sources with respect to two well-known data models in the museum domain: Europeana Data Model (EDM) [Hennicke et al., 2011], and CIDOC Conceptual Reference Model (CIDOC-CRM) [Doerr, 2003]. EDM[5] and CIDOC-CRM[6] data

[5]`http://pro.europeana.eu/page/edm-documentation`

[6]`http://www.cidoc-crm.org`

Table 3.2: The evaluation datasets ds_{edm} and ds_{crm}.

	ds_{edm}	ds_{crm}
#data source	29	29
#classes in the domain ontologies	119	147
#properties in the domain ontologies	351	409
#nodes in the gold-standard models	473	812
#data nodes in the gold-standard models	329	418
#class nodes in the gold-standard models	141	394
#links in the gold-standard models	441	785

models use different domain ontologies to represent knowledge in the museum domain.

The first dataset, ds_{edm} (the same dataset that we used for evaluation in Section 2.5), contains the EDM, AAC, SKOS, Dublin Core Metadata Terms, FRBR, FOAF, ORE, and ElementsGr2 ontologies, and the second dataset, ds_{crm}, includes the CIDOC-CRM and SKOS ontologies. The reason why we used two data models is to evaluate how our approach performs with respect to different representations of knowledge in a domain. We applied our approach on both datasets 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. Table 3.2 shows more details of the evaluation datasets. The datasets including the sources, the domain ontologies, and the gold standard models are available on GitHub.[7] The source code of our approach is integrated into Karma which is available as open source.[8]

In each dataset, we applied our method to learn a semantic model for a target source s_i, $sm(s_i)$, assuming that the semantic models of the other sources are

[7]`https://github.com/taheriyan/phd-thesis`

[8]`https://github.com/usc-isi-i2/Web-Karma`

known. To investigate how the number of the known models influences the results, we used variable number of known models as input. Suppose that $\mathrm{M_j}$ is a set of known semantic models including j models. Running the experiment with $\mathrm{M_0}$ means that we do not use any knowledge other than the domain ontology and running it with $\mathrm{M_{28}}$ means that the semantic models of all the other sources are known ($\mathrm{M_{28}}$ is leave-one-out cross validation). For example, for s_1, we ran the code 29 times using $\mathrm{M_0}$=$\{\}$, $\mathrm{M_1}$=$\{sm(s_2)\}$, $\mathrm{M_2}$=$\{sm(s_2), sm(s_3)\}$, $\cdots$, $\mathrm{M_{28}}$=$\{sm(s_2), \cdots, sm(s_{29})\}$.

In learning the semantic types of a source s_i, we use the data of the sources whose semantic models are known as training data. More precisely, when we are running our labeling algorithm on source s_i with $\mathrm{M_j}$ setting, the training data is the data of all the sources $\{s_k | k = 1, .., j \textit{ and } k != i\}$ and the test data is the data of the target source s_i. Using $\mathrm{M_0}$ means that there is no training data and thus the labeling function will not be able to suggest any semantic type for the source attributes. To evaluate the labeling algorithm, we use *mean reciprocal rank* (MRR) [Craswell, 2009], which is useful when we consider top k semantic types. MRR helps to analyze the ranking of predictions made by any semantic labeling approach using a single measure rather than having to analyze top-1 to top-k prediction accuracies separately, which is a cumbersome task. In learning the semantic types of a source s_i with n attributes, MRR is computed as:

$$MRR = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{rank_i}$$

where $rank_i$ is the rank of the correct semantic type in the top k predictions made for the attribute a_i. It is obvious that if we only consider the top semantic type predictions, the value of MRR is equal to the accuracy. In our example in Table 3.1,

$MRR = 1/5(1/1 + 1/1 + 1/1 + 1/1 + 1/2) = 0.9$ (the correct semantic type for the attribute $imageURL$ is ranked second).

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

$$precision = \frac{|rel(sm) \cap rel(sm')|}{|rel(sm')|}$$

$$recall = \frac{|rel(sm) \cap rel(sm')|}{|rel(sm)|}$$

where $rel(sm)$ is the set of triples (u, v, e) in which e is a link from the node u to the node v in the semantic model sm. For example, for the semantic model in Figure 3.2c, *rel(sm)*=$\{$*(edm:EuropeanaAggregation, aac:CulturalHeritageObject, edm:aggregatedCHO), (edm:EuropeanaAggregation, edm:WebResource, edm: hasView),(aac:CulturalHeritageObject, aac:Person, dcterms:creator),* $\cdots\}$.

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 $sm = sm'$. For example, the two semantic models exemplified in Figure 3.7 have the same set of triples although they do not convey the same semantics. In Figure 3.7a, the creator of the artwork knows another person while the semantic model in Figure 3.7b states that the creator of the artwork is known by another person. Many sources in our datasets have models that include two or more instances of an ontology class.

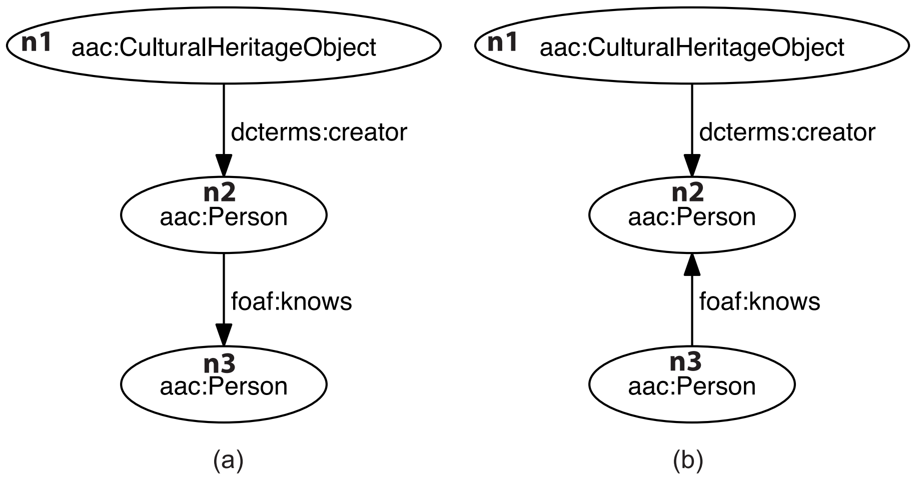


Figure 3.7: These two semantic models are not equivalent.

To have a more accurate evaluation, we number the nodes and then use the numbered labels in measuring the precision and recall. Assume that the model in Figure 3.7a is the correct semantic model (*sm*) and the one in 3.7b is the model learned by our approach (sm'). We change the labels of the nodes n_1, n_2 and n_3 in *sm* to *aac:CulturalHeritageObject1*, *aac:Person1* and *aac:Person2*. After this change, we will have *rel(sm)*={*(aac:CulturalHeritageObject1, aac:Person1, dcterms:creator), (aac:Person1, aac:Person2, foaf: knows)*}. Then, we try all the permutations of the numbering in the learned model sm' and report the precision and recall of the one that generates the best *F1-measure*.[9] For instance, if we number the nodes n_2 and n_3 in sm' with *aac:Person1* and *aac:Person2*, we will have *rel(sm′)*={*(aac:CulturalHeritageObject1, aac:Person1, dcterms:creator), (aac:Person2, aac: Person1, foaf:knows)*}, which yields *precision=recall=0.5*. If we label n_2 with *aac:Person2* and n_3 with *aac:Person1*, we will have *rel(sm′)*=

[9] *F1-measure*$=2*(precision \times recall)/(precision + recall)$

Table 3.3: The overlap between the pairs of the semantic models in the datasets ds_{edm} and ds_{crm}.

	ds_{edm}	ds_{crm}
minimum overlap	0.04	0.03
maximum overlap	1	1
median overlap	0.45	0.46
average overlap	0.43	0.46

{*(aac:CulturalHeritageObject1, aac:Person2, dcterms:creator), (aac:Person1, aac: Person2, foaf:knows)*}, which still has *precision=recall=0.5*.

One of the factors influencing the results of our method is the overlap between the known semantic models and the semantic model of the target source. To see how much two semantic models overlap each other, we define the *overlap* metric as the Jaccard similarity between their relationships:

$$overlap = \frac{|rel(sm) \cap rel(sm')|}{|rel(sm) \cup rel(sm')|}$$

Table 3.3 reports the minimum, maximum, median, and average overlap between the semantic models of each dataset. Overall, the higher the overlap between the known semantic models and the semantic model of the target source, the more accurate models can be learned.

In our mapping algorithm (Algorithm 3.5), we used 50 as cut-off (*branching_factor=50*) and then considered all the generated mappings as the candidate mappings (*num_of_candidates=50*). We justify this choice in Section 3.3.2 by analyzing the impact of the branching factor on the accuracy of the results and the running time of the algorithm. To score a mapping m, we assigned equal weights to the functions *confidence*(m), *coherence*(m), and *sizeReduction*(m). We tried different combinations of weights, and although our algorithm generated more precise models for a few sources in some of these weight systems, the average results were better when each of these functions contributed equally to the final score. Once we found the candidate mappings, we generated the top 10 Steiner trees for each of them (*k=10* in *top-k* Steiner tree algorithm). Finally, we ranked the candidate semantic models (at most 500) and compared the best one with the correct model of the source. We ran two experiments with different scenarios that will be explained next.

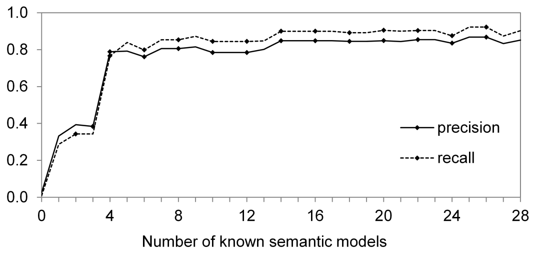
3.3.1 Scenario 1

In the first scenario, we assumed that each source attribute is annotated with its correct semantic type. The goal was to see how well our approach learns the attribute relationships using the correct semantic types. Figure 3.8 illustrates the average precision and recall of all the learned semantic models ($sm'(s_1), \cdots,$ $sm'(s_{29})$) for each $\mathrm{M_j}$ ($j \in [0..28]$) for each dataset. Since the correct semantic types are given, we excluded their corresponding triples in computing the precision and recall. That is, we compared only the links between the class nodes in the gold standard models with the links between the class nodes in the learned models. We call such links *internal links*, the links that are established between the class nodes in semantic models. The total number of the links in the dataset ds_{edm} is 441, and 329 of these links corresponds to the source attributes (there are 329 data nodes). Thus, ds_{edm} has 112 internal links (441-329=112). Following the same rationale, ds_{crm} has 367 internal links.

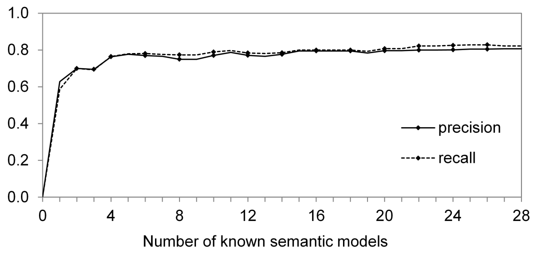
The results show that the precision and recall increase significantly even with a few known semantic models. An interesting observation is that when there is no known semantic model and the only background knowledge is the domain ontology (baseline, $\mathrm{M_0}$), the precision and recall are close to 0. This low accuracy comes from the fact that there are multiple links between each pair of class nodes in the graph G, and without additional information, we cannot resolve the ambiguity. Although we assign lower weights to direct properties to prioritize them over inherited ones, it cannot help much because for many of the class nodes in the correct models, there is no object property in the ontology that is explicitly defined with the corresponding classes as domain and range. In fact, most of the properties that have been used in the correct models are either inherited properties or defined without a domain or/and range in the ontology.

To evaluate the running time of the approach, we measured the running time of the algorithm starting from building the graph until ranking the results on a single machine with a Mac OS X operating system and a 2.3 GHz Intel Core i7 CPU. Figure 3.9 shows the average time (in seconds) of learning the semantic models. The reason why there is some fluctuations in the timing diagram of ds_{crm} (Figure 3.9b) is related to the topology of the graph built on top of the known models and also the details of our implementation. While one expects to see linear increase in time when the number of known semantic models grows, sometimes adding a new semantic model changes the structure of the graph in a way that the Steiner tree algorithm finds k candidate trees faster.

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 scratch. We just need to merge the new known model to the existing graph and update the links.

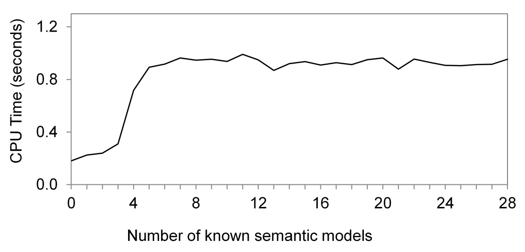


(a) ds_{edm}

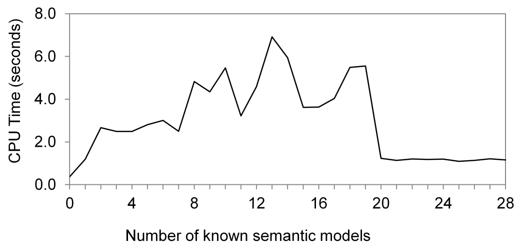


(b) ds_{crm}

Figure 3.8: Average precision and recall for the learned semantic models when the attributes are labeled with their correct semantic types.



(a) ds_{edm}

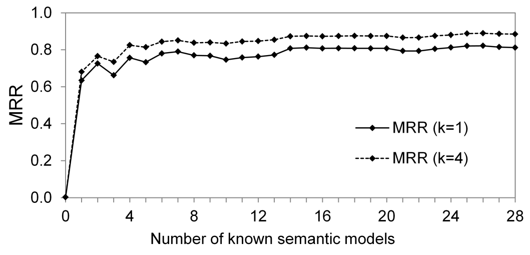


(b) ds_{crm}

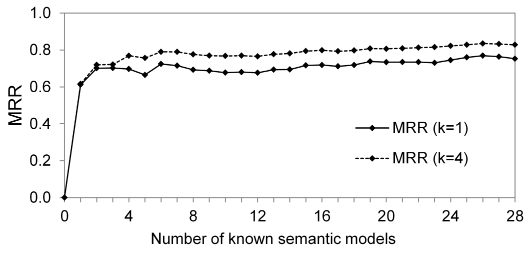
Figure 3.9: Average semantic model learning time when the attributes are labeled with their correct semantic types.

3.3.2 Scenario 2

In the second scenario, we used our semantic labeling algorithm to learn the semantic types. We trained the labeling classifier on the data of the sources whose semantic models are already known and then applied the learned labeling function to the target source to assign a set of candidate semantic types to each source attribute. Figure 3.10 shows the MRR diagram for ds_{edm} and ds_{crm} in two cases: (1) only the top semantic type (the type with the highest confidence value) is considered (k=1), (2) the top four learned semantic types are taken into account as the candidate semantic types (k=4). Note that, when k=1, the MRR value is equal to the accuracy, i.e., how many of the attributes are labeled with their correct semantic types.

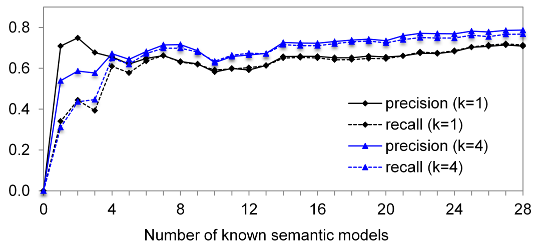


(a) ds_{edm}

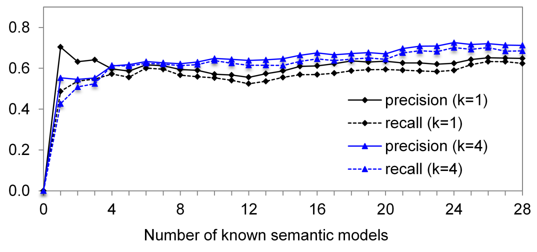


(b) ds_{crm}

Figure 3.10: MRR value of the learned semantic types when only the top learned semantic types are considered (k=1); and the top four suggested types are considered (k=4).



(a) ds_{edm}



(b) ds_{crm}

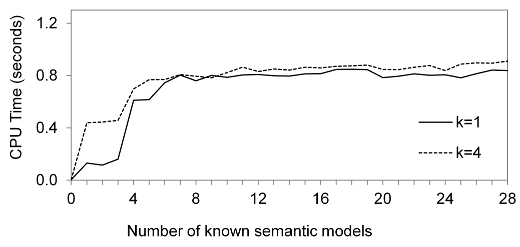
Figure 3.11: Average precision and recall for the learned semantic models for k=1 and k=4.

Once the labeling is done, we feed the learned semantic types to the rest of algorithm to learn a semantic model for each source. The average precision and recall of the learned models are illustrated in Figure 3.11. The black color shows the precision and recall for k=1, and the blue color illustrates the precision and recall for k=4. In this experiment, we computed precision and recall for all the links including the links from the class nodes to the data nodes (these links are associated with the learned semantic types). The results show that using the known semantic models as background knowledge yields in a remarkable improvement in both precision and recall compared to the case in which we only consider the domain ontology (M_0).

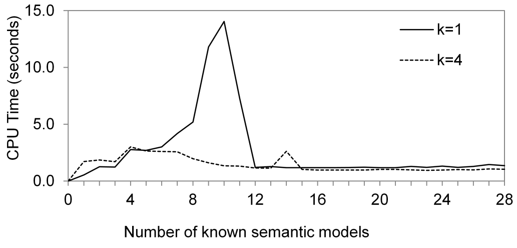
We provide an example to help in understanding the correlation between the MRR and the precision values, i.e., how the accuracy of the learned semantic types affects the accuracy of the learned semantic models. The average MRR value for ds_{crm} when we use k=1 and M_{28} (leave-one-out setting) is 0.75 (Figure 3.10b). This means that our labeling algorithm can learn the correct semantic types for only 75% of the attributes. From Table 3.2, we know that the gold standard models for ds_{crm} have totally 418 data nodes, and thus, 418 links in the gold standard models correspond to the source attributes. Since 75% of the attributes are labeled correctly, 313 links out of 418 links corresponding to the source attributes will be correct in the learned semantic models. Even if we predict all the internal links correct (785-418=367 links), the maximum precision would be 86% ((367+313)/785). However, the input to the Steiner tree algorithm are the nodes coming from the learned semantic types (leaves of the tree), and incorrect semantic types may prompt the Steiner tree algorithm to select incorrect links in the higher levels (internal links). As we see in Figure 3.11b, in the k=1 and M_{28} setting, the average precision of the learned semantic models is 65%.

When considering the top four semantic types (k=4) instead of only the top one semantic type (k=1), our algorithm recovers some of the correct semantic types even if they are not the top predictions of the labeling function. For example, in the dataset ds_{crm}, using k=4 rather than k=1 when we have 28 known models (M_{28}), improves the precision by 6% and the recall by 7% (Figure 3.11b). This improvement is mainly because of the coherence factor we take into account in scoring the mappings and also ranking the candidate semantic models.

The running time of the algorithm in the second scenario is displayed in Figure 3.12. This time does not include the labeling step. The work done by Krishnamurthy et al. [Krishnamurthy et al., 2015] contains a detailed analysis of the



(a) ds_{edm}

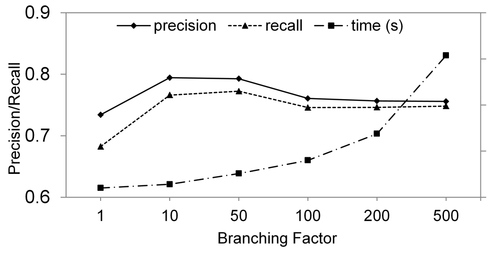


(b) ds_{crm}

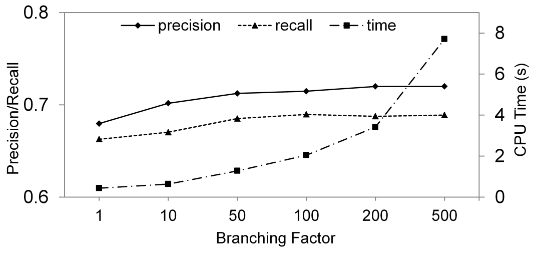
Figure 3.12: Average semantic model learning time when the attributes are labeled with their correct semantic types.

performance of the labeling algorithm. As we can see in Figure 3.12b, the running time of the algorithm is higher at M_8, M_9, M_{10}, and M_{11} when k=1. This is because computing top 10 Steiner trees takes longer once we add semantic models of s_8, s_9, s_{10}, and s_{11} to the graph. When adding more semantic models, the algorithm runs faster. For example, the average time at M_{11} is 7.29 seconds while it is 1.21 seconds at M_{12}. This is the result of a combination of several reasons. First, there is more training data in learning the semantic types of a source s_i at M_{12}, and this affects the output of the mapping algorithm (Algorithm 3.5). Second, the structure of the graph is different at M_{12} and this results in different mappings between the source attributes and the graph. Finally, the new semantic model $sm(s_{12})$ adds new paths to the graph allowing the Steiner tree algorithm to find the top 10 trees faster.

We mentioned earlier that we used 50 as the value of the branching factor in mapping the source attributes to the graph (line 21 of Algorithm 3.5). The branching factor is essential to the scalability of our mapping algorithm. This value can be configured by trying some sample values and then choosing a value yielding good accuracy while keeping the running time of the algorithm reasonably low. This can be different for each dataset. In our evaluation, using *branching_factor=50*



(a) ds_{edm}



(b) ds_{crm}

Figure 3.13: impact of branching factor on precision, recall, and running time for k=4 and M_{28}.

worked well for both datasets. Figure 3.13 illustrates how changing the value of the branching factor affects the precision, recall, and running time of the algorithm in a setting where we considered 4 candidate semantic types (k=4) and the semantic models of all the other sources were known (M_{28}). In this experiment, we fixed the value of *num_of_candidates* (line 27 of Algorithm 3.5) equal to the value of *branching_factor*. This means that all the generated mappings will be given to the Steiner tree algorithm as the candidate mappings. As we can see in Figure 3.13b, increasing the value of the branching factor from 50 to 200 for ds_{crm} provides 1% improvement in the precision, however, it increases the average running time by 2.14 seconds. We chose to ignore this insignificant increase in the precision and used 50 as the branching factor to gain a better running time.

3.3.3 User Effort

Manually constructing semantic models, in addition to being time-consuming and error-prone, requires a thorough understanding of the domain ontologies. Karma [Knoblock et al., 2012] provides a user friendly graphical interface enabling users to interactively build the semantic models. Yet, building the models in Karma without any automation requires significant user effort. Our automatic approach learns accurate semantic models that can be transformed to the gold standard models by only a few user actions.

To measure how much our new approach, which learns from the known semantic models, facilitates the modeling task in Karma, we integrated our approach into Karma and repeated the same experiment we did in Section 2.5 (where we only used the knowledge from the domain ontology to build the semantic models). We used Karma to model the dataset ds_{edm} according to the EDM data model and then counted the number of user actions required to convert the predicted models to the correct models.

The results are shown in Table 3.4. The *Choose Type* column shows the number of times that the correct semantic type of a column was not present in the top four semantic types learned by our labeling function and we had to manually browse the ontology to select the correct semantic type. Since we employed the same labeling algorithm that was used in the experiment in Section 2.5, the values of this column are exactly similar to the values in Table 2.2.

The *Change Link* column shows the number of times we had to add/delete links to/from the suggested models or change one of the predicted links. If we compare the values of this column with the values of the same column in Table 2.2, we observe a significant reduction in the number of user actions (reducing from 96 to 19). This is because the approach we introduced in this chapter learns popular and coherent patterns from the semantic models of previously modeled sources and uses them to hypothesize the attribute relationships for a new source.

Comparing the results of the experiment in this section with the results of the experiment in Section 2.5, we see that our new approach decreases the total number of user actions from 152 to 77 (from 5.2 per data source to 2.6 per data source). The overall time the user took to model all the sources also dropped from 128

Table 3.4: The evaluation results for modeling the dataset ds_{edm} using Karma.

Source	**# Columns**	**# User Actions**			**Time (min)**
		Choose Type	Change Link	Total	
s1	7	7	1	8	3
s2	12	5	1	6	5
s3	4	0	0	0	2
s4	17	5	6	11	7
s5	14	4	2	6	5
s6	18	4	1	5	5
s7	14	1	0	1	4
s8	6	0	0	0	2
s9	4	1	0	1	2
s10	11	3	0	3	4
s11	6	1	0	1	2
s12	9	0	0	0	2
s13	10	1	0	1	3
s14	9	1	0	1	3
s15	13	0	0	0	4
s16	5	0	0	0	2
s17	12	0	0	0	3
s18	5	0	0	0	1
s19	17	0	0	0	3
s20	9	2	0	2	2
s21	28	8	6	14	10
s22	8	1	0	1	2
s23	18	0	1	1	4
s24	10	1	0	1	3
s25	13	1	0	1	3
s26	14	2	1	3	4
s27	12	2	0	2	4
s28	15	4	1	5	4
s29	9	2	1	3	3
Total	**329**	**56**	**19**	**77**	**101**
		Avg. # User Actions/Column = 77/329 = 0.23			

minutes to 101 minutes (from 4.4 minutes per source to 3.4 minutes per source), and most of this time was for assigning the correct semantic types by browsing the ontology. The results show that leveraging the known semantic models in additions to the domain ontology significantly reduces the number of user actions compared to the case where we only use the domain ontology.

Chapter 4

Leveraging Linked Data to Infer Semantic Relations

In previous chapter, we presented an automatic approach that exploits the known semantic models to learn a semantic model for a new unknown source. However, in many domains, only few, if any, known semantic models are available. In this chapter, we present an approach that leverages the knowledge from Linked Open Data (LOD) to learn semantic models. This approach complements the work in our previous chapter in cases where the number of known semantic models is limited in a domain.

LOD is a vast and growing collection of semantic data that has been published by various data providers. The current estimate is that the LOD cloud contains over 30 billion RDF triples. Even the New York Times is now publishing all of their metadata as Linked Open Data.[1] Given the growing availability of this type of data, LOD will provide an invaluable source of semantic content that we can exploit as background knowledge.

The semantic data in the LOD cloud can be leveraged to learn how instances of different classes are linked to each other. Once we have identified the semantic types of the source attributes, we can search for corresponding classes in LOD and analyze which properties are connecting them. Suppose that a source with two attributes is labeled with *dbpedia:City* and *dbpedia:State*. We can search LOD for

[1]See `http://data.nytimes.com`

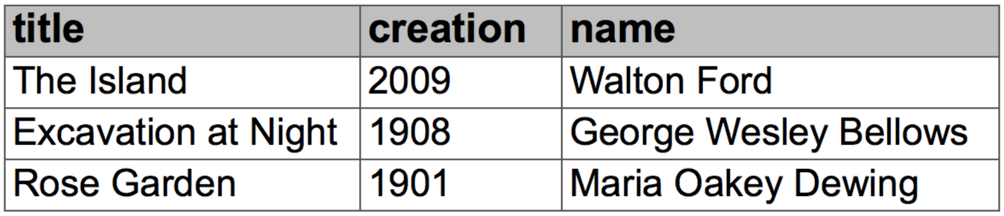
possible properties between these two classes and find that the properties *dbpedia:state* and *dbpedia:isPartOf* are better candidates than other properties that ontology suggests, e.g., *dbpedia:closeTo*. By combining the information we extract for each pair of classes, we can narrow the search to those classes and properties that commonly occur together.

The main contribution of the work we introduce in this chapter is exploiting the graph patterns occurring in the linked data to disambiguate the relationships between the source attributes [Taheriyan et al., 2015b]. The main difference between the algorithm in this chapter and the one in Chapter 3 is the way we create the graph. Here, instead of the known semantic models, we use the patterns extracted from LOD to build the graph. The rest of the two approaches are similar. We use SPARQL to extract graph patterns with different lengths occurring in the linked data. We combine these patterns into one graph and expand the resulting graph using the paths inferred from the domain ontology. Then, we search the graph starting from the semantic labels of the source and heuristically find the top k semantic models connecting all the labels.

4.1 Example

In this section, we provide an example from the museum domain to explain how the knowledge from LOD can help us to infer semantic relations within structured sources. In this example, we want to model a data source using the CIDOC-CRM ontology and then use the created semantic model to publish the source data as RDF. This source is a table containing information about artworks in the Crystal Bridges Museum of American Art[2] (Figure 4.1). We formally write the signature

[2] `http://crystalbridges.org`



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

Figure 4.1: Sample data from the Crystal Bridges Museum of American Art.

of this source as *s(title, creation, name)* where *s* is the name of the source and *title*, *creation*, and *name* are the names of the source attributes (columns). The correct semantic types for the columns *title*, *creation*, and *name* are ⟨*E35_Title,label*⟩, ⟨*E52_Time-Span, P82_at_some_time_within*⟩, and ⟨*E82_Actor_Appellation,label*⟩.

Figure 4.2 shows the correct semantic model of the source *s*. As we can see in the figure, none of the semantic types corresponding to the source columns are directly connected to each other, which makes the problem of finding the correct semantic model more complex. There are many paths in the CIDOC-CRM ontology connecting the assigned labels. For instance, we can use the classes *E39_Actor* and *E67_Birth* to relate the semantic types *E82_Actor_Appellation* and *E52_Time-Span*:

(*E39_Actor*, *P1_is_identified_by*, *E21_Actor_Appellation*)
(*E39_Actor*, *P98i_was_born*, *E67_Birth*)
(*E67_Birth*, *P4_has_time-span*, *E52_Time-Span*)

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

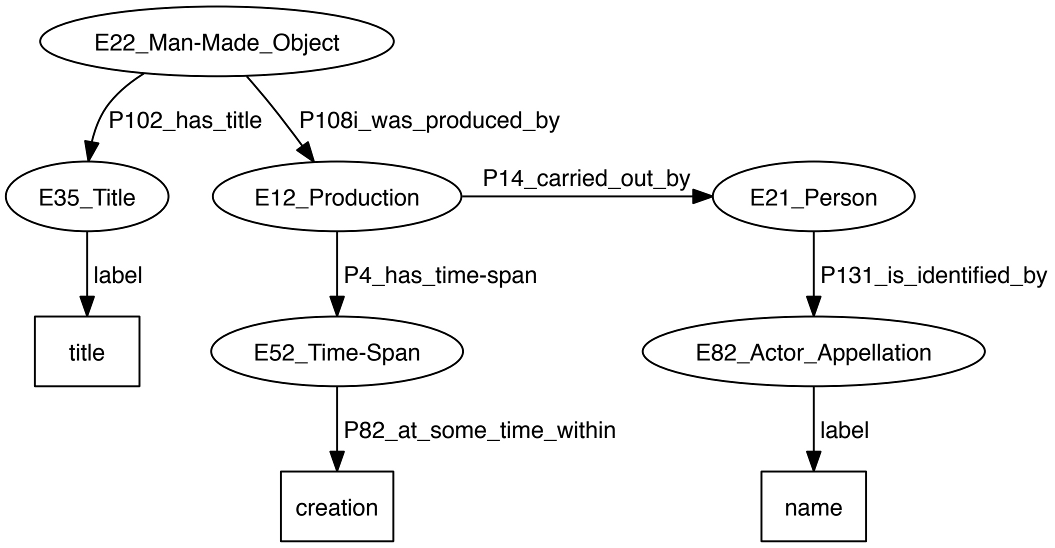


Figure 4.2: The semantic model of the source *s*.

Now assume that we have access to a repository of linked data containing the RDF triples published by some other museums. We can exploit this linked data to bias the search space to prefer those models that are used for related source. Once we have identified the semantic types of the source attributes, we can search the linked data to find the frequent patterns connecting the corresponding classes. For example, by querying the linked data, we find out that *P131_is_identified_by* is more popular than *P1_is_identified_by* to connect instances of *E82_Actor_Appellation* and instances of *E21_Person*, and this makes sense when we investigate the definitions of these two properties in the ontology. The property *P1_is_identified_by* describes the naming or identification of any real world item by a name or any other identifier, and *P131_is_identified_by* is a specialization of *P1_is_identified_by* that identifies a name used specifically to identify an instance of *E39_Actor* (superclass of *E21_Person*). We can query the linked data to find longer paths between entities. For instance, by inspecting the paths with length two between the instances of *E22_Man-Made_Object* and *E21_Person*, we observe that the best way to connect these two classes is through the path: $\mathit{E22\_Man\text{-}Made\_Object} \xrightarrow{\mathit{P108i\_was\_produced\_by}}$ $\mathit{E12\_Production} \xrightarrow{\mathit{P14\_is\_carried\_out\_by}} \mathit{E21\_Person}$.

4.2 Inferring Semantic Relations

In this section, we explain our approach to automatically deduce the attribute relationships within a data source. Since we have described our labeling algorithm before (Chapter 2), in this section, we assume that source attributes are already labeled with semantic types and we focuses on learning the relationships. The input to our approach are the domain ontology, a repository of linked data in the same domain, and a data source whose attributes are already labeled with semantic types. The output is a semantic model expressing how the assigned labels are connected.

4.2.1 Extracting Patterns from Linked Open Data

The Linked Open Data (LOD) cloud includes a vast and growing collection of semantic content published by various data providers in many domains. When modeling a source in a particular domain, we can exploit the linked data published in that domain to hypothesize attribute relationships within the source. We assume that the source attributes are labeled with ⟨*class_uri,property_uri*⟩ pairs.

We use SPARQL to query the linked data in order to extract the graph patterns connecting the instances of the classes corresponding to the semantic types. Each pattern consists of some nodes and links. The nodes correspond to ontology classes and the links correspond to ontology properties. Suppose that we want to find the patterns connecting three classes c_1, c_2, and c_3. Figure 4.3 exemplifies some of the possible patterns to connect these classes. Depending on

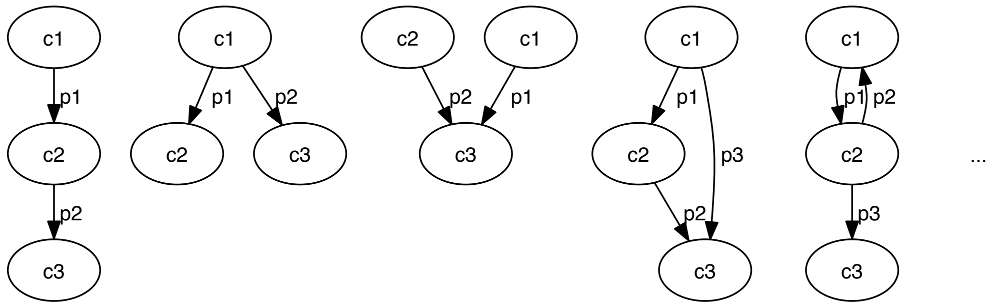


Figure 4.3: Sample graph patterns connecting the classes c_1, c_2, and c_3 using the ontology properties p_1, p_2, and p_3.

the structure of the domain ontology, there might be a large number of possible patterns for any given number of ontology classes. For simplicity, in this paper, we only extract the tree patterns, the patterns in which the number of properties is exactly one less than the number of classes (the first three patters in Figure 4.3). That said, we define the length of a pattern as the number of links (ontology properties) in a pattern. For example, a pattern with length one is in the form of $c_1 \xrightarrow{p} c_2$ indicating that at least one instance of the class c_1 is connected to an instance of the class c_2 with the property p. The following SPARQL query extracts the patterns with length one along with their frequencies from the linked data:

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

```
ORDER BY DESC(?count);
```

Querying a triple store containing a huge amount of linked data to mine long patterns is not efficient. In our experiments, we only extracted the patterns with length one and two. We will see later that even small graph patterns provide enough evidence to infer rich semantic models.

4.2.2 Merging LOD Patterns into a Graph

Once we extracted the LOD patterns, we combine them into a graph G that will be used to infer the semantic models. Building the graph has three parts: (1) adding the LOD patterns, (2) adding the semantic labels assigned to the source attributes, and (3) expanding the graph with the paths inferred from the ontology. Algorithm 4.1 shows the pseudocode of building the graph.

The graph G is a weighted directed graph in which nodes correspond to ontology classes and links correspond to ontology properties. The algorithm to construct the graph is straightforward. However, we adopt a subtle approach to weight the links. We assign a much lower weight to the links added from the LOD patterns comparing to the links added from the ontology. Since we are generating minimum-cost models in the next section, this weighting strategy gives more priority to the links used in the linked data. The weight of the links coming from the LOD patterns has an inverse relation with the frequency of the patterns.

The other important feature of the links in the graph is their *tags*. We assign an identifier to each pattern added to the graph and annotate the links with the identifiers of the supporting patterns. Suppose that we are adding two patterns $m_1 : c_1 \xrightarrow{p_1} c_2 \xrightarrow{p_2} c_3$ and $m2 : c_1 \xrightarrow{p_1} c_2 \xrightarrow{p_3} c_4$ to G. The link p_1 from c_1 to c_2 will be tagged with $\{m_1, m_2\}$, the link p_2 from c_2 to c_3 will have only $\{m_1\}$ as its tag

Algorithm 4.1 Construct Graph G

```
    Input: LOD Patterns, Semantic Types, Domain Ontology
    Output: Graph $G$

    ▷ Add LOD patterns
 1: sort the patterns descending based on their length
 2: exclude the patterns contained in longer patterns
 3: merge the nodes and links of the remaining patterns into $G$

    ▷ Add Semantic Types
 4: for each semantic type ⟨class_uri,property_uri⟩ do
 5:     add the class to the graph if it does not exist in $G$
 6: end for

    ▷ Add Ontology Paths
 7: for each pair of classes $c_i$ and $c_j$ in $G$ do
 8:     find the directed and inherited properties between $c_i$ and $c_j$ in the ontology
 9:     add the properties that do not exist in $G$
10: end for

    return $G$
```

set, and the link p_3 from c_2 to c_4 will be tagged with $\{m_2\}$. We use the link tags later to prioritize the semantic models containing larger segments from the LOD patterns.

4.2.3 Mapping Source Attributes to the Graph

This part of the algorithm is exactly similar to what we explained in Section 3.2.3. The goal is to map the semantic types assigned to the attributes to the nodes of the graph. If there is only one instance of each class in the graph, the mapping will be one to one. However, we might have patterns that include two instances of the same ontology class, e.g., *E82_Actor_Appellation* $\xrightarrow{P106\_is\_composed\_of}$ *E82_Actor_Appellation*. In this case, the graph will have two nodes with the label *E82_Actor_Appellation*, and if one of the source attributes is labeled with

E82_Actor_Appellation, we will have two ways to map this attribute to the graph. We score the possible mappings and then select the more promising mappings as the input to the next step (similar to Algorithm 3.5).

4.2.4 Generating and Ranking Semantic Models

The final part of our approach is to compute the semantic models from the graph, and this step is also the same as the technique presented in 3.2.4. Once we mapped the semantic types to the nodes of the graph, we find the top k minimal trees connecting those nodes. We rank the resulting models first based on their coherence and then their cost (sum of the weights of the links). The coherence metric gives priority to the models that contain longer patterns. For example, a model that includes one pattern with length 3 will be ranked higher than a model including two patterns with length 2, and the latter in turn will be preferred over a model with only one pattern with length 2.

4.3 Evaluation

To evaluate our approach, we used the dataset ds_{crm} that was also used in previous chapter. The linked data that we used as the background knowledge is the RDF data published by the Smithsonian American Art Museum.[3] The museum has made use of the CIDOC Conceptual Reference Model (CIDOC-CRM) to map out the concepts and relationships that exist within the artwork collection. This repository includes more than 3 million triples (3,398,350). We injected the data into a Virtuoso triple store and then used SPARQL to extract patterns of length

[3] `http://americanart.si.edu/collections/search/lod/about`

Table 4.1: The evaluation results for modeling the dataset ds_{crm}. The correct semantic types are given as input.

background knowledge	**precision**	**recall**	**time (s)**
domain ontology	0.07	0.05	0.17
domain ontology + patterns of length one	0.65	0.55	0.75
domain ontology + patterns of length one and two	0.78	0.70	0.46

one and two. There were 68 distinct patterns with length one (two nodes and one link) and 634 distinct patterns with length two (three nodes and two links).

We assumed that the correct semantic labels for the source attributes are known. The goal was to see how well our approach learns the attribute relationships having the correct semantic types. We performed three experiments. First, we only used the domain ontology to build a graph on top of the semantic labels and then computed the semantic model connecting those labels. In the second experiment, we took into account the patterns with length one extracted from the linked data, and in the third experiment, we used the patterns of both length one and two. We measured the accuracy of the computed semantic models by comparing them with the gold standard models in terms of precision and recall. Since the correct semantic types are given, we excluded their corresponding triples in computing the precision and recall. For example, in learning the semantic model of the source s (Figure 4.2), we do not consider *(E35_Title, label, title)* in our evaluation. From the 785 links in the correct models, 418 links correspond to the semantic types because we have 418 attributes, and the rest are the internal links. Therefore, our evaluation in this scenario measures the precision and recall of inferring the 367 internal links.

Table 4.1 shows the average precision and recall for all 29 sources. An interesting observation is that when we do not use the linked data patterns, the precision and recall are close to zero. This low accuracy comes from the fact that in most of the gold standard models, the attributes are not directly connected and there are multiple paths between each pair of classes in the ontology (and thus in our graph), and without additional information, we cannot resolve the ambiguity. When we exploit the patterns with length one, there is a boost in precision and recall. Since we are using the pattern frequencies in assigning the weights to the links of the graph, using patterns of length one means that we are only taking into account the popularity of the links in computing the semantic models. Once we added the patterns with length two, our approach achieved 13% improvement in precision and 15% increase in recall. This means that considering coherence (even to a small extent) in addition to the link popularity empowers our approach to derive more accurate semantic models.

We can compare the results in Table 4.1 (inferring relationships from LOD) with the results shown in Figure 3.8b in Section 3.3 (inferring relationships from known semantic models). The target dataset is the same in both experiments, ds_{crm}, and we assume that the correct semantic types are given as input in both cases, however, the precision and recall in Table 4.1 are lower than the leave-one-out setting (M_{28}) in Figure 3.8b (78% precision and 70% recall compared to 81% precision and 82% recall). Two reasons explain the difference between the results of our experiment in this section and the experiment in Section 3.3. First, we used different kinds of background knowledge in these two experiments. In this section, we used LOD (the linked data published by Smithsonian American Art Museum) to infer the relationships between the attributes of a source, but in Section 3.3, we used the semantic models of other sources to learn the semantic relations within each data source. Thus, we can expect to observe different outputs because the overlap between the semantic model of a target source and the background knowledge is not the same. The second reason why the accuracy is lower in the case of using LOD is that we only used patterns of length one and two in our experiment. On the other hand, our approach in Section 3.3 exploits complete semantic models of previously modeled sources, which are more coherent than the small graph patterns we used in this section. We believe that both the precision and recall will be further improved if we extract longer patterns from LOD and incorporate them into our approach.

The column time in Table 4.1 shows the running time of our algorithm on a single machine with a Mac OS X operating system and a 2.3 GHz Intel Core i7 CPU. This is the time from combining pre-extracted LOD patterns into a graph until generating and ranking candidate semantic models. The algorithm is much faster when we only use the domain ontology as the background knowledge, because adding LOD patterns will be excluded from the graph construction process (lines 1-3 in Algorithm 4.1). The reason why more time is required when we use patterns of length one comparing to the case where we use patterns of both length one and two is related to details of our algorithm to compute top k minimal trees. Although it takes longer to create the graph when we add patterns of length two, the algorithm to generate the candidate models finds top k trees faster reducing the total running time.

Our evaluation results support the theory that more accurate models can be constructed when longer graph patterns from LOD are used. Although these patterns can be pre-computed, using SPARQL queries to extract long patterns from a large number of triples is challenging. For example, the Virtuoso response time to the SPARQL query to extract patterns of length two with a chain shape ($c_1 \xrightarrow{p_2} c_2 \xrightarrow{p_3} c_3$) was approximately 90 seconds, and it was roughly 1 hour for the query to extract patterns of length two having a V-shape ($c_1 \xrightarrow{p_2} c_2 \xleftarrow{p_3} c_3$). We were only able to collect a few patterns with length three and could not extract any pattern with length four from our Virtuoso server in a 5-hour timeout. Efficiently mining more complex patterns from the linked data is one direction of our future work.

Chapter 5

Related Work

The problem of describing semantics of data sources is at the core of data integration [Doan et al., 2012] and exchange [Arenas et al., 2010]. The main approach to reconcile the semantic heterogeneity among sources consists of defining logical mappings between the source schemas and a common target schema. One way to define these mappings is local-as-view (LAV) descriptions where every source is defined as a view over the domain schema [Doan et al., 2012]. The semantic models that we generate are graphical representation of LAV rules, where the domain schema is the domain ontology. Although the logical mappings are declarative, defining them requires significant technical expertise, so there has been much interest in techniques that facilitate their generation.

In traditional data integration, the mapping generation problem is usually decomposed in a *schema matching* phase followed by *schema mapping* phase [Bellahsene et al., 2011]. Schema matching [Rahm and Bernstein, 2001] finds correspondences between elements of the source and target schemas. For example, iMAP [Dhamankar et al., 2004] discovers complex correspondences by using a set of special-purpose searchers, ranging from data overlap, to machine learning and equation discovery techniques. This is analogous to the semantic labeling step in our work [Krishnamurthy et al., 2015], where we learn a labeling function 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).

Schema mapping defines an appropriate transformation that populates the target schema with data from the sources. Mappings may be arbitrary 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 [Fagin et al., 2009], which provided a practical system and furthered the theoretical foundations of data exchange [Fagin et al., 2005] to more recent systems that support additional schema constraints [Marnette et al., 2011]. Alexe et al. [Alexe et al., 2011] generate schema mappings from examples of source data tuples and the corresponding tuples over the target schema.

An et al. [An et al., 2007] generate declarative mapping expressions between two tables with different schemas starting from element correspondences. They create a graph from the conceptual model (CM) of each schema and then suggest plausible mappings by exploring low-cost Steiner trees that connect those nodes in the CM graph that have attributes participating in element correspondences. This is similar to our semi-automatic approach to build the semantic models where we derive a graph from the domain ontology and the learned semantic types (Chapter 2). We exploit the knowledge from the ontology to assign weights to the links based on their types, e.g., direct properties get lower weight than inherited properties, because we want to give more priority to more specific relations. We also allows the user to correct the mappings interactively.

Our work on learning semantic models of structured sources (Chapter 3) is complementary to these schema mapping techniques. Instead of focusing on satisfying schema constraints, we analyze known source models to propose mappings that capture more closely the semantics of the target source in ways that schema constraints could not disambiguate. For example, by suggesting that a *dcterms:creator* relationship is more likely than *dbpedia:owner* in a given domain. Moreover, our algorithm can incrementally refine the mappings based on user feedback and learn from this feedback to improve future predictions.

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 a domain ontology. There are many studies on mapping data sources to ontologies. Several approaches have been proposed to generate semantic web data from databases and spreadsheets [Sahoo et al., 2009].

D2R [Bizer, 2003; Bizer and Cyganiak, 2006] and D2RQ [Bizer and Seaborne, 2004] are mapping languages that enable the user to define mapping rules between tables of relational databases and target ontologies in order to publish semantic data in RDF format. R2RML [Das et al., 2012] is a another mapping language, which is a W3C recommendation for expressing customized mappings from relational databases to RDF datasets. Writing the mapping rules by hand is a tedious task. The users need to understand how the source table maps to the target ontology. They also need to learn the syntax of writing the mapping rules.

RDOTE [Vavliakis et al., 2010] is a tool that provides a graphical user interface to facilitate mapping relational databases into ontologies. The developers of RDOTE have said they will incorporate an export/import mechanism for D2RQ compliant mapping files, as well as a query builder graphical user interface to hasten the mapping creation process. RDF123 [Han et al., 2008] and XLWrap [Langegger and Wöß, 2009] are other tools to define mappings from spreadsheets to RDF graphs. Although these tools can facilitate the mapping process, the users still need to manually define the mappings between the source and target ontologies.

In recent years, there are some efforts to automatically infer the implicit semantics of tables. Polfliet and Ichise [Polfliet and Ichise, 2010] use string similarity between the column names and the names of the properties in the ontology to find a mapping between the table columns and the ontology. Wang et al. [Wang et al., 2012] detect the header of Web tables and use them along with the values of the rows to map the columns to the attributes of the corresponding entity in a rich and general purpose taxonomy of worldly facts built from a corpus of over one million Web pages and other data. This approach can only deal with the tables containing information of a single entity type.

Limaye et al. [Limaye et al., 2010] used YAGO[1] to annotate web tables and generate binary relationships using machine learning approaches. However, this approach is limited to the labels and relations defined in the YAGO ontology (less than 100 binary relationships). Venetis et al. [Venetis et al., 2011] presented a scalable approach to describe the semantics of tables on the Web. To recover the semantics of tables, they leverage a database of class labels and relationships automatically extracted from the Web. They attach a class label to a column if a sufficient number of the values in the column are identified with that label in the database of class labels, and analogously for binary relationships. Although these approaches are very useful in publishing semantic data from tables, they are limited in learning the semantics relations. Both of these approaches only infer individual binary relationships between pair of columns. They are not able to find the relation between two columns if there is no direct relationship between the values of those columns. Our approach to learn semantic models can connect one column to another one through a path in the ontology. For example, suppose that we have a table including two columns *person* and *city*, where the city is

[1]`http://www.mpi-inf.mpg.de/yago-naga/yago`

the location of the company the person is working for. Our approach can learn a semantic model that connects the class *Person* to the class *City* through the chain $Person \xrightarrow{worksFor} Organization \xrightarrow{location} City$.

There is also work that exploits the data available in the Linked Open Data (LOD) cloud to capture the semantics of the tables and publish their data as RDF. Munoz et al. [Muñoz et al., 2013] mine RDF triples from the Wikipedia tables by linking the cell values to the resources available in DBPedia [Auer et al., 2007]. This approach is limited to Wikipedia tables because of its simple linking algorithm. If a cell value contains a hyperlink to a Wikipedia page, the Wikipedia URL maps to a DBpedia entity URI by replacing the namespace `http://en.wikipedia.org/wiki/` of the URL with `http://dbpedia.org/resource/`.

In other work, Mulwad et al. [Mulwad et al., 2013] used *Wikitology* [Syed and Finin, 2011], an ontology which combines some existing manually built knowledge systems such as DBpedia and Freebase [Bollacker et al., 2008], to link cells in a table to Wikipedia entities. They query the background LOD to generate initial lists of candidate classes for column headers and cell values and candidate properties for relations between columns. Then, they use a probabilistic graphical model to find the correlation between the columns headers, cell values, and relation assignments. The quality of the semantic data generated by this category of work is highly dependent to how well the data can be linked to the entities in LOD. While for most popular named entities there are good matches in LOD, many tables contain domain-specific information or numeric values (e.g., temperature and age) that cannot be linked to LOD. Moreover, these approaches are only able to identify individual binary relationships between the columns of a table. However, an integrated semantic model is more than fragments of binary relationships between the columns. In a complete semantic model, the columns may be connected through a path including the nodes that do not correspond to any column in the table.

Semantic annotation of services have also received attention. Annotating the input and output parameters of Web services and Web APIs is useful for automatic service discovery and composition. SAWSDL vocabulary [Farrell and Lausen, 2007] allows adding semantic meta-data to service descriptions. Service inputs and outputs can be annotated by the concepts and properties using an attribute called *modelReference*. SWEET [Maleshkova et al., 2009] is a tool that supports users in creating semantic descriptions of RESTful services. There has been some work on classifying Web services into different domains [Heß et al., 2003] and automatically labeling the input and outputs of Web services [Lerman et al., 2006; Saquicela et al., 2011]. This work provides useful knowledge for service discovery, but not sufficient for automating service integration. In our work, we are able to learn more expressive descriptions of Web services that describe how the attributes of a service relate to one another.

Parundekar et al. [Parundekar et al., 2012] previously developed an approach to automatically generate conjunctive and disjunctive mappings between the ontologies of linked data sources by exploiting existing linked data instances. However, the system does not model arbitrary sources such as we present in this paper. Carman and Knoblock [Carman and Knoblock, 2007] use known source descriptions to learn a semantic description that precisely describes the relationship between the inputs and outputs of a source, expressed as a Datalog rule. However, their approach is limited in that it can only learn sources whose models are subsumed by the models of known sources. That is, the description of a new source is a conjunctive *combination* of known source descriptions. 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.

In recent years, ontology matching has received much attention in the Semantic Web community [Kalfoglou and Schorlemmer, 2003; Pavel and Euzenat, 2013]. Ontology matching (or ontology alignment) finds the correspondence between semantically related entities of different ontologies. This problem is analogous to schema matching in databases. Both schemas and ontologies provide a vocabulary of terms that describe a domain of interest. However, schemas often do not provide explicit semantics for their data. Our work benefits from some of the techniques developed for ontology matching. For example, *instance-based ontology matching* exploits similarities between instances of ontologies in the matching process. Our semantic labeling algorithm adopts the same idea to map the data of a new source to the classes and properties of a target ontology. The algorithm computes the similarity (cosine similarity between TF/IDF vectors) between the data of the new source and the data of the sources whose semantic models are known.

Ontology matching is different than the problem we addressed in this paper in the sense that in our work the data that is being mapped to a target ontology is not bound to any source ontology. This makes our problem more complicated since no explicit semantics is necessarily attached to data sources. Moreover, most of the work on ontology matching only finds simple correspondences such as equivalence and subsumption between ontology classes and properties. Therefore, the explicit relationships within the data elements are often missed in aligning the source data to the target ontology. Suppose that we want to find the correspondences between a source ontology O_s and a target ontology O_t. Using ontology matching, we find that the class A_s in O_s maps to the class A_t in O_t and the class B_s in O_s maps to the class B_t in O_t. Assume that there is only one property connecting A_s to B_s in O_s, but there are multiple paths connecting A_t to B_t in O_t. If we align the source data to the target ontology O_t using the correspondences found by ontology matching, the instances of A_s will be mapped to the class A_t and the instances of B_s will be mapped to the class B_t. However, this alignment does not tell us which path in O_t captures the correct meaning of the source data.

In our approach presented in Chapter 2 [Knoblock et al., 2012], we build a graph from learned semantic types and a domain ontology and use this graph to map a source to the ontology *interactively*. Karma, which is an open source data integration tool, employs our semi-automatic modeling approach to propose models to the user, who can correct them as needed. Szekely et al. [Szekely et al., 2013] used Karma to model the data from Smithsonian American Art Museum[2] and then publish it into the Linked Open Data cloud. Karma is also able to build semantic models for Web services and then exploits the created semantic models to build APIs that directly communicate at the semantic level [Szekely et al., 2011; Taheriyan et al., 2012a,b].

Most of the work mentioned earlier is manual or automates learning semantic types for services parameters or table columns, but is limited in learning relationships. Our automatic approach explained in Chapter 3 [Taheriyan et al., 2013, 2014, 2015a] exploits the known semantic models to learn a semantic model for a new unknown source. We analyze known semantic models to hypothesize semantic models that capture more closely the semantics of the new source. Integrating our algorithm into Karma, enables the user to refine the automatically learned models resulting in more accurate predictions for future data sources.

[2]`http://americanart.si.edu`

Our work in Chapter 4 [Taheriyan et al., 2015b] complements our method in Chapter 3 in cases where few, if any, known semantic models are available. In fact, the number of known semantic models is limited in many domains, however, there may be a huge amount of semantic data published in those domain. We leverage the small graph patterns from the available linked data to infer the semantic relationships within data sources.

Chapter 6

Discussion

In this dissertation, we presented a novel approach to learn semantic models of structured data sources. Such models are the key ingredients to automate tasks such as source discovery and data integration. They also automate the process of publishing semantic data. In this chapter, I first summarize the contributions of my thesis. Next, I discuss some of the applications areas and highlight some limitations of the presented approach. Finally, I list possible directions for future work.

6.1 Contributions

The key contributions of this work are threefold. In Chapter 2, we use a domain ontology as the background knowledge to semi-automatically build semantic models. In Chapter 3, we exploit the knowledge from known semantic models in addition to the domain ontology to automatically learn semantic models. Finally, in Chapter 4, we leverage the knowledge from Linked Open Data (LOD) to learn the semantic models. We summarize each part in the following paragraphs.

In Chapter 2, we presented a semi-automatic approach to build a semantic model of a new source as a mapping from the source to a domain ontology (Chapter 2). The system attempts to exploit the knowledge from the domain ontology to automate as much of the source modeling as possible and relies on the user to help disambiguate the semantic models when there is insufficient information to fully automate the modeling task.

We use an existing machine learning technique to label the attributes of the new source with semantic types, classes and/or properties of the ontology. Second, we exploit the knowledge inferred from the domain ontology to build a graph that models the possible relationships between the learned semantic types. Then, we find an initial semantic model by computing the minimal tree connecting the semantic types in the graph. If the suggested model is not the right interpretation of the source data, the user can interactively impose constraints on the algorithm to refine the model.

In Chapter 3, we presented a scalable approach to make the process of building semantic models more automated. The core idea is to exploit the knowledge of previously learned semantic models to learn a plausible semantic model for a new source. The first step in learning semantic models is learning semantic types which is done exactly the same as our semi-automatic approach. The output of the labeling step is a set of candidate semantic types and their confidence values rather than one fixed semantic type. Taking into account the uncertainty of the labeling algorithm 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 also adding the paths inferred from the ontology. Our algorithm to construct the graph consolidates the overlapping segments of the known semantic models, makes it scalable to a huge number of known semantic models.

The next step is mapping the source attributes to the nodes of the graph where we use a search algorithm that enables the system to do the mapping even when the source has many attributes. The algorithm, after processing each source attribute, prunes 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. It takes into account the confidence scores associated with the semantic types and the coherence of the nodes in each mapping in order to calculate the top k promising mappings.

The final part of the approach is computing the minimal tree that connects the nodes of the candidate mappings. We build a minimal tree over each mapping to generate k candidate models. We score the candidates to output a ranked list of the most plausible semantic models. Our evaluation shows that our approach learns rich semantic models with minimal user input. It also shows that exploiting both the domain ontology and the known semantic models yields in semantic models that are significantly more accurate than the models built by only using the knowledge from the domain ontology.

In many domains, there is none or very limited number of known semantic models available. However, there may be a huge amount of semantic data published in those domain. In Chapter 4, we presented a new approach that mines the small graph patterns from the available linked data to infer the semantic relationships within data sources. We use SPARQL to extract graph patterns with different lengths occurring in the linked data. We treat these patterns as known models. We combine them into one graph and expand the resulting graph using the paths inferred from the domain ontology. Then, we compute the top k semantic models from the graph.

Our work provides a basis to learn the semantic models of structured information sources. The learned semantic models explicitly represent the relationships between the source attributes in addition to their semantic types. Such precise models of data sources make it possible to automatically integrate the data across sources and provides rich support for source discovery. They also make it possible to convert sources into RDF and publish them in the Linked Data cloud.

Our learning algorithms play an important role in making the Karma interactive user interface easy to use, a key design goal given that many of our users are domain experts, but are not Semantic Web experts. Our experience observing users is that they can understand and critique models when displayed in our interactive user interface. They can easily verify that models accurately capture the semantics of a source, and can easily spot errors or controversial modeling decisions. Users can click on the corresponding elements on the screen and do local modifications such as replacing the property of a link or changing the source or destination of a link.

We also observe that it is much harder for users to model a source from scratch, as is necessary in tools such as Open Refine.[1] Even though the user interface is easy to use, the task of filling a blank page with a model is daunting for many users. Karma helps these users because it gives them an almost-correct model as a starting point. Users can easily find the elements they do not agree with, and can easily change them. A possible direction for future work is to perform user evaluations to measure the quality of the models produced using learning algorithms. Although time to create models is important, we hypothesize that most users, such as our museum users, are primarily concerned with producing correct models, and time to model is a secondary concern for them. By using previous models, users are more likely to model sources in a correct way.

[1]`http://openrefine.org/`

A large portion of the data in the LOD cloud is published directly from existing databases using tools such as D2R [Bizer, 2003; Bizer and Cyganiak, 2006]. This conversion process uses the structure of the data as it is organized in the database, which may not be the most useful structure of the information in RDF. But either way, there is often no explicit semantic description of the contents of a source and it requires a significant effort if one wants to do more than simply convert a database into RDF. The result of the ease with which one can publish data into the LOD cloud is that there is lots of data published in RDF and remarkably little in the way of semantic descriptions of much of this data. Our approach brings the semantics into the conversion process by producing a semantic model that maps a data source to a common domain ontology. This model can then be used to generate RDF triples that are linked to an ontology and to provide a SPARQL end point that converts the data on the fly into RDF with respect to a given ontology.

Our work also plays a role in helping communities to produce consistent Linked Data so that sources containing the same type of data use the same classes and properties when published in RDF. Often, there are multiple correct ways to model the same type of data. For example, users can use Dublin Core and FOAF to model the creator relationship between a person and an object (*dcterms:creator* and *foaf:maker*). A community is better served when all the data with the same semantics is modeled using the same classes and properties. Our work encourages consistency because our learning algorithms bias the selection of classes and properties towards those used more frequently in existing models.

6.2 Applications

Our research addresses the fundamental question of how an intelligent system can learn things of which it has limited knowledge. This capability will allow people and systems to better exploit the massive amount of data available today on the Internet and provide a tool to keep up with its growth. Within the bioinformatics world, for example, the amount of data continues to grow rapidly, and the ability to find and structure this data will have a significant impact on our ability to fully exploit all of this information to solve biomedical research questions, such as finding more effective treatments for cancer. Today in the bioinformatics service repository, called BioCatalogue,[2] there are approximately 2,500 services available with limited semantic descriptions of the capabilities provided by these services. The work in this thesis will provide an important step in the ability to automatically create semantic models of services, which will make it possible for researchers to quickly find relevant services as well as to automatically compose services to solve specific technical problems.

We have integrated our approach into Karma [Knoblock et al., 2012], our data integration framework. Users integrate information in Karma by modeling it according to an ontology of their choice using a graphical user interface that automates much of the process. The modeling component of Karma is an implementation of the approaches we presented in Chapter 2 and Chapter 3. Karma learns to recognize the mapping of data to ontology classes and then uses the ontology as well as the previously modeled sources to propose a model that ties together these classes. Users then interact with the system to adjust the automatically generated model. During this process, users can transform the data as needed

[2]`http://www.biocatalogue.org`

to normalize data expressed in different formats and to restructure it. Once the model is complete, users can published the integrated data as RDF or store it in a database. The modeling framework we have developed is domain independent, so it can be applied across a variety of domains. Sometimes these applications provide surprising solutions to problems in other fields that we had not anticipated.

Our tool helped the museum community to provide more detailed and more easily accessed information about artworks. We collaborated with Smithsonian American Art Museum (SAAM) to model their data using the museum standard ontologies and map their data to Linked Open Data (LOD).[3] This enables online users to access records of more than 40,000 artworks in a structured format designed to be easier to interlink both inside and outside of the museums online presence. We are now working with the museum community to grow the project and create the American Art Collaborative. This effort will integrate the data about artwork across a set of American museums.

We also used Karma in data integration research within the Southern California Clinical Translational Science Institute.[4] One of the many tasks of the SC-CTSI is to create a social network of participating researchers. To that end, the SC-CTSI evaluated several tools including VIVO.[5] One of the major challenges of using this tool by SC-CTSI staff was loading the existing personnel/faculty data from multiple departmental databases and HR systems into the RDF-based VIVO tool, according to the VIVO ontology (which describes researchers, institutions, publications, etc). We realized that our Karma modeling tool was well suited to

[3]`http://viterbi.usc.edu/news/news/2014/creating-the-art.htm`

[4]`http://sc-ctsi.org`

[5]`http://www.vivoweb.org`

facilitate this process. Using Karma we were able to transform the data from multiple databases and populate the VIVO knowledge base quickly and with minimum effort. We presented these results at the annual VIVO conference and it was very well received. Karma is currently in use by the VIVO community for mapping their data sources into the VIVO knowledge base.

Karma also plays a vital role in the DIG system,[6] which is developed as part of the DARPA Memex program,[7] an effort to create the next generation of domain-specific search technologies. DIG harnesses state-of-the-art open source software combined with an open architecture and flexible set of APIs to facilitate the integration of a variety of extraction and analysis tools. DIG uses Karma to build rich models for data sources in a domain and then builds a graph of the entities and relationships within the domain using scalable extraction and linking technologies. DIG also includes a faceted content search interface for users to query DIGs and visualize information on maps, timelines, and tables. Szekely et al. have used DIG in the problem of combating human trafficking and have successfully deployed it to six law enforcement agencies and several non-governmental organizations to assist them with finding traffickers and helping victims in a matter of months [Szekely et al., 2015].

Thus, our current research has already contributed to several areas that we had not expected. We expect new efficiencies and capabilities will be generated when we and others apply Karma to novel fields. This effect should be multiplied by the fact that our code is available to the public as open source, so that others can easily extend and adapt its capabilities for new domains.

[6]`http://dig.isi.edu`

[7]`http://www.darpa.mil/program/memex`

6.3 Limitations

The work presented in Chapter 3 exploits known semantic models to map a given source to a domain ontology. One limitation of this approach is that in many domains, there are not many known semantic models available. Therefore, our algorithm only relies on the domain ontology to hypothesize a semantic model for the input source. The same thing happens when there is not any (or enough) overlap between the new source and the previously modeled sources. As the evaluation showed (Section 3.3), using only the domain ontology as the background knowledge does not yield accurate semantic models. The reason is that the domain ontology often defines a large space of possible semantic models and without additional knowledge we cannot resolve the ambiguity. Fortunately, our approach allows users to give feedback on the proposed suggestions and uses that feedback to improve further suggestions. However, in cases where there are not enough known semantic models available, the user often needs to make many changes to the initially learned model in order to build the correct model.

A limitation of our approach to infer semantic relationships from LOD (Chapter 4) is that it heavily depends on the linked data at hand. It assumes that there is sufficient amount of linked data available in the same domain that we are modeling the target data sources. The other assumption is that the linked data contains relationship instances between the instances of the classes assigned to the source attributes as the semantic types. Our approach cannot leverage enough useful patterns when no or sparse data is available, and this negatively impacts the accuracy of the generated semantic models.

6.4 Future Work

There are some future directions for this work that will allow our approach to be applied more broadly. One direction is to improve the techniques used for semantic labeling of source attributes. Semantic types are very important in our approach. The input to the Steiner tree algorithm that computes the candidate semantic models are the nodes coming from the learned semantic types (leaves of the tree). Incorrect semantic types may prompt the Steiner tree algorithm to select incorrect links in the higher levels (internal links).

In Chapter 4, we use the LOD patterns to infer relationships between the source attributes. Nonetheless, we make no use of LOD in learning the semantic types for the source attributes. One possible direction for future work is to leverage LOD in the semantic labeling step of our approach. Given the huge repository of data available in LOD, for any given set of values provided by a new source, we can search for classes that provide or even subsume all of the data for a given property of a source. For example, if we have a set of values for people names or temperature, we are likely to find some classes in LOD that provides that same set of values. We will not require a perfect overlap between the set of values from the source and a class in LOD, but rather a statistically significant overlap, similar to what is done by Parundekar et al. [Parundekar et al., 2012]. An important challenge here is how to efficiently find the classes that most closely match the set of attribute values and how to handle the problem that the classes that match the best may come form different ontologies.

Another direction for future work is to improve the quality of the models learned from LOD. Our evaluation results (Section 4.3) support the theory that more accurate models can be constructed when longer graph patterns from LOD are used. Although these patterns can be pre-computed, using SPARQL queries to extract long patterns from a large number of triples is challenging. For example, for our Virtuoso repository including 3,398,350 RDF triples, the response time to the SPARQL query to extract patterns of length two with a chain shape ($c_1 \xrightarrow{p_2} c_2 \xrightarrow{p_3} c_3$) was approximately 90 seconds, and it was roughly 1 hour for the query to extract patterns of length two having a V-shape ($c_1 \xrightarrow{p_2} c_2 \xleftarrow{p_3} c_3$). We were only able to collect a few patterns with length three and could not extract any pattern with length four from our Virtuoso server in a 5-hour timeout. A promising path for future study includes efficiently mining more complex patterns from the linked data and incorporating those patterns in learning semantic models.

As discussed earlier, the strength of our work in Chapter 3 is limited when there are not enough known semantic models. This is the motivation of our approach in Chapter 4, which exploits the published semantic data instead of the known semantic models as the background knowledge. The drawback of this approach is that it cannot generate accurate models when no or sparse data is available. One direction of future work is to combine the work in Chapter 3 (learning semantic models from the known semantic models) and the work in Chapter 4 (learning semantic models from the LOD patterns). The combined approach will use the known semantic models, LOD patterns, and domain ontology to learn a semantic model for a new source. We believe leveraging the knowledge from both the known semantic models and the LOD patterns can help to learn more precise semantic models.

This thesis focuses on learning semantic models of data sources. These semantic models are the key to automatically populate knowledge graphs (e.g., the LOD cloud) with semantic content (e.g., RDF). However, publishing semantic data according to a shared vocabulary is just the first step to build a knowledge graph. The second step is to link the information at the instance level. Linking entities distributed across different published datasets is an active research area in the Semantic Web community. Adding the capability of linking the data at the instance level empowers our approach to automate the entire workflow of publishing linked data on the Web.

Bibliography

Alexe, Bogdan; ten Cate, Balder; Kolaitis, Phokion G., and Tan, Wang-Chiew. Designing and Refining Schema Mappings via Data Examples. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 133–144, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4.

An, Yuan; Borgida, Alexander; Miller, Renée J., and Mylopoulos, John. A Semantic Approach to Discovering Schema Mapping Expressions. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE)*, pages 206–215, Istanbul, Turkey, 2007.

Arenas, Marcelo; Barcelo, Pablo; Libkin, Leonid, and Murlak, Filip. *Relational and XML Data Exchange*. Morgan & Claypool, San Rafael, CA, 2010.

Auer, Sören; Bizer, Christian; Kobilarov, Georgi; Lehmann, Jens; Cyganiak, Richard, and Ives, Zachary. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 722–735, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-76297-3, 978-3-540-76297-3.

Bellahsene, Zohra; Bonifati, Angela, and Rahm, Erhard. *Schema Matching and Mapping*. Springer, 1st edition, 2011. ISBN 9783642165177.

Bhalotia, Gaurav; Hulgeri, Arvind; Nakhe, Charuta; Chakrabarti, Soumen, and Sudarshan, S. Keyword Searching and Browsing in Databases Using BANKS. In *Proceedings of the 18th International Conference on Data Engineering*, pages 431–440, 2002.

Bizer, Christian. D2R MAP - A Database to RDF Mapping Language. In *WWW (Posters)*, 2003.

Bizer, Christian and Cyganiak, Richard. D2R Server - Publishing Relational Databases on the Semantic Web. In *Poster at the 5th International Semantic Web Conference*, 2006.

Bizer, Christian and Seaborne, Andy. D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. In *ISWC2004 (posters)*, November 2004.

Bizer, Christian; Heath, Tom, and Berners-Lee, Tim. Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst.*, 5(3):122, 2009.

Bollacker, Kurt; Evans, Colin; Paritosh, Praveen; Sturge, Tim, and Taylor, Jamie. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-102-6.

Carman, Mark J. and Knoblock, Craig A. Learning Semantic Definitions of Online Information Sources. *Journal of Artificial Intelligence Research*, 30(1):1–50, September 2007. ISSN 1076-9757.

Craswell, Nick. Mean reciprocal rank. In *Encyclopedia of Database Systems*, page 1703. 2009.

Das, Souripriya; Sundara, Seema, and Cyganiak, Richard. R2RML: RDB to RDF Mapping Language, W3C Recommendation 27 September 2012. http://www.w3.org/TR/r2rml/, 2012.

Dhamankar, Robin; Lee, Yoonkyong; Doan, AnHai; Halevy, Alon, and Domingos, Pedro. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *International Conference on Management of Data (SIGMOD)*, pages 383–394, New York, NY, 2004.

Ding, Li; DiFranzo, Dominic; Graves, Alvaro; Michaelis, James; Li, Xian; McGuinness, Deborah L., and Hendler, James A. TWC Data-gov Corpus: Incrementally Generating Linked Government Data from data.gov. In Rappa, Michael; Jones, Paul; Freire, Juliana, and Chakrabarti, Soumen, editors, *WWW*, pages 1383–1386. ACM, 2010. ISBN 978-1-60558-799-8.

Doan, Anhai; Halevy, Alon, and Ives, Zachary. *Principles of Data Integration*. Morgan Kauffman, 2012.

Doerr, Martin. The CIDOC Conceptual Reference Module: An Ontological Approach to Semantic Interoperability of Metadata. *AI Mag.*, 24(3):75–92, September 2003. ISSN 0738-4602.

Fagin, Ronald; Kolaitis, Phokion G.; Miller, Renée J., and Popa, Lucian. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science*, 336 (1):89 – 124, 2005.

Fagin, Ronald; Haas, Laura M.; Hernández, Mauricio; Miller, Renée J.; Popa, Lucian, and Velegrakis, Yannis. Clio: Schema Mapping Creation and Data Exchange. In *Conceptual Modeling: Foundations and Applications*, 2009.

Farrell, Joel and Lausen, Holger. Semantic Annotations for WSDL and XML Schema, August 2007. W3C Recommendation.

Han, Lushan; Finin, Tim; Parr, Cynthia; Sachs, Joel, and Joshi, Anupam. RDF123: From Spreadsheets to RDF. pages 451–466. 2008.

Hennicke, Steffen; Olensky, Marlies; Boer, Viktor De; Isaac, Antoine, and Wielemaker, Jan. A Data Model for Cross-domain Data Representation. The Europeana Data Model in the Case of Archival and Museum Data. In *Schriften zur Informationswissenschaft 58, Proceedings des 12. Internationalen Symposiums der Informationswissenschaft (ISI 2011)*, pages 136–147, 2011.

Heß, Andreas; Kushmerick, Nick, and Kushmerick, Nicholas. Learning to Attach Semantic Metadata to Web Services. In *Proceedings of the 2nd International Semantic Web Conference (ISWC)*, pages 258–273. Springer, 2003.

Kalfoglou, Yannis and Schorlemmer, Marco. Ontology Mapping: The State of the Art. *Knowl. Eng. Rev.*, 18(1):1–31, January 2003. ISSN 0269-8889.

Knoblock, Craig; Szekely, Pedro; Ambite, José Luis; Goel, Aman; Gupta, Shubham; Lerman, Kristina; Muslea, Maria; Taheriyan, Mohsen, and Mallick, Parag. Semi-Automatically Mapping Structured Sources into the Semantic Web. In *Proc. 9th Extended Semantic Web Conference*, 2012.

Kou, Lawrence T.; Markowsky, George, and Berman, Leonard. A Fast Algorithm for Steiner Trees. *Acta Informatica*, 15:141–145, 1981. ISSN 0001-5903.

Krishnamurthy, Ramnandan; Mittal, Amol; Knoblock, Craig A., and Szekely, Pedro. Assigning Semantic Labels to Data Sources. In *Proceedings of the 12th Extended Semantic Web Conference (ESWC)*, May 2015.

Langegger, Andreas and Wöß, Wolfram. XLWrap - Querying and Integrating Arbitrary Spreadsheets with SPARQL. In Bernstein, Abraham; Karger, David R.; Heath, Tom; Feigenbaum, Lee; Maynard, Diana; Motta, Enrico, and Thirunarayan, Krishnaprasad, editors, *International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*, pages 359–374. Springer, 2009. ISBN 978-3-642-04929-3.

Lehmann, Erich L. and Romano, Joseph P. *Testing Statistical Hypotheses*. Springer Texts in Statistics. Springer, New York, third edition, 2005. ISBN 0-387-98864-5.

Lerman, Kristina; Plangrasopchok, Anon, and Knoblock, Craig A. Semantic Labeling of Online Information Sources. *IJSWIS, special issue on Ontology Matching*, 2006.

Limaye, Girija; Sarawagi, Sunita, and Chakrabarti, Soumen. Annotating and Searching Web Tables Using Entities, Types and Relationships. *PVLDB*, 3(1): 1338–1347, 2010.

Maleshkova, Maria; Pedrinaci, Carlos, and Domingue, John. Semantically Annotating RESTful Services with SWEET. In *8th International Semantic Web Conference (ISWC2009)*, October 2009.

Marnette, Bruno; Mecca, Giansalvatore; Papotti, Paolo; Raunich, Salvatore, and Santoro, Donatello. ++Spicy: an OpenSource Tool for Second-Generation Schema Mapping and Data Exchange. In *Procs. VLDB*, pages 1438–1441, Seattle, WA, 2011.

Mehlhorn, Kurt. A Faster Approximation Algorithm for the Steiner Problem in Graphs. *Information Processing Letters*, 27(3):125 – 128, 1988. ISSN 0020-0190.

Muñoz, Emir; Hogan, Aidan, and Mileo, Alessandra. Triplifying Wikipedia's Tables. In Gentile, Anna Lisa; Zhang, Ziqi; d'Amato, Claudia, and Paulheim, Heiko, editors, *LD4IE@ISWC*, volume 1057 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

Mulwad, Varish; Finin, Tim, and Joshi, Anupam. Semantic Message Passing for Generating Linked Data from Tables. In *The Semantic Web - ISWC 2013*, pages 363–378. Springer, 2013.

Parundekar, Rahul; Knoblock, Craig A., and Ambite, José Luis. Discovering Concept Coverings in Ontologies of Linked Data Sources. In *Proceedings of the 11th International Semantic Web Conference (ISWC)*, Boston, MA, 2012.

Pavel, Shvaiko and Euzenat, Jérôme. Ontology Matching: State of the Art and Future Challenges. *IEEE Trans. on Knowl. and Data Eng.*, 25(1):158–176, January 2013. ISSN 1041-4347.

Polfliet, Simeon and Ichise, Ryutaro. Automated Mapping Generation for Converting Databases into Linked Data. In Polleres, Axel and Chen, Huajun, editors, *ISWC Posters&Demos*, volume 658 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

Rahm, Erhard and Bernstein, Philip A. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4), 2001.

Sahoo, Satya S.; Halb, Wolfgang; Hellmann, Sebastian; Idehen, Kingsley; Jr, Ted Thibodeau; Auer, Sören; Sequeda, Juan, and Ezzat, Ahmed. A Survey of Current Approaches for Mapping of Relational Databases to RDF, 01 2009.

Saquicela, Victor; Blázquez, Luis Manuel Vilches, and Óscar Corcho, . Lightweight Semantic Annotation of Geospatial RESTful Services. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC)*, pages 330–344, 2011.

Sheth, Amit P.; Gomadam, Karthik, and Ranabahu, Ajith. Semantics Enhanced Services: METEOR-S, SAWSDL and SA-REST. *IEEE Data Eng. Bulletin*, 31 (3):8–12, 2008.

Syed, Zareen and Finin, Tim. Creating and Exploiting a Hybrid Knowledge Base for Linked Data. In *Agents and Artificial Intelligence*, pages 3–21. Springer, 2011.

Szekely, Pedro; Knoblock, Craig A.; Gupta, Shubham; Taheriyan, Mohsen, and Wu, Bo. Exploiting Semantics of Web Services for Geospatial Data Fusion. In *Proceedings of the SIGSPATIAL International Workshop on Spatial Semantics and Ontologies (SSO 2011)*, Chicago, IL, 2011.

Szekely, Pedro; Knoblock, Craig A.; Yang, Fengyu; Zhu, Xuming; Fink, Eleanor; Allen, Rachel, and Goodlander, Georgina. Connecting the Smithsonian American Art Museum to the Linked Data Cloud. In *Proceedings of the 10th Extended Semantic Web Conference (ESWC)*, pages 593–607, Montpellier, May 2013.

Szekely, Pedro; Knoblock, Craig A.; Slepicka, Jason; Philpot, Andrew; Singh, Amandeep; Yin, Chengye; Kapoor, Dipsy; Natarajan, Prem; Marcu, Daniel; Knight, Kevin; Stallard, David; Karunamoorthy, Subessware S.; Bojanapalli, Rajagopal; Minton, Steven; Amanatullah, Brian; Hughes, Todd; Tamayo, Mike; Flynt, David; Artiss, Rachel; Chang, Shih-Fu; Chen, Tao; Hiebel, Gerald, and Ferreira, Lidia. Building and Using a Knowledge Graph to Combat Human Trafficking. In *Proceedings of the 14th International Semantic Web Conference (ISWC 2015)*, 2015.

Taheriyan, Mohsen; Knoblock, Craig A.; Szekely, Pedro, and Ambite, José Luis. Semi-Automatically Modeling Web APIs to Create Linked APIs. In *Proceedings of the Linked APIs for the Semantic Web Workshop (LAPIS)*, 2012a.

Taheriyan, Mohsen; Knoblock, Craig A.; Szekely, Pedro, and Ambite, José Luis. Rapidly Integrating Services into the Linked Data Cloud. In *ISWC*, pages 559–574, Boston, MA, 2012b.

Taheriyan, Mohsen; Knoblock, Craig A.; Szekely, Pedro, and Ambite, José Luis. A Graph-based Approach to Learn Semantic Descriptions of Data Sources. In *Procs. 12th International Semantic Web Conference (ISWC)*, 2013.

Taheriyan, Mohsen; Knoblock, Craig A.; Szekely, Pedro, and Ambite, José Luis. A Scalable Approach to Learn Semantic Models of Structured Sources. In *Semantic Computing (ICSC), 2014 IEEE International Conference on*, pages 183–190, June 2014.

Taheriyan, Mohsen; Knoblock, Craig; Szekely, Pedro, and Ambite, José Luis. Learning the Semantics of Structured Data Sources. *Journal of Web Semantics Special Issue on Knowledge Graphs*, 2015a.

Taheriyan, Mohsen; Knoblock, Craig; Szekely, Pedro; Ambite, José Luis, and Chen, Yinyi. Leveraging Linked Data to Infer Semantic Relations within Structured Sources. In *Proceedings of the 6th International Workshop on Consuming Linked Data (COLD 2015)*, 2015b.

Takahashi, Hiromitsu and Matsuyama, Akira. An Approximate Solution for the Steiner Problem in Graphs. *Math.Japonica*, 24:573–577, 1980.

Vavliakis, Konstantinos N.; Grollios, Theofanis K., and Mitkas, Pericles A. RDOTE - Transforming Relational Databases into Semantic Web Data. In Polleres, Axel and Chen, Huajun, editors, *ISWC Posters & Demos*, volume 658 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

Venetis, Petros; Halevy, Alon; Madhavan, Jayant; Paşca, Marius; Shen, Warren; Wu, Fei; Miao, Gengxin, and Wu, Chung. Recovering Semantics of Tables on the Web. *Proc. VLDB Endow.*, 4(9):528–538, June 2011. ISSN 2150-8097.

Wang, Jingjing; Wang, Haixun; Wang, Zhongyuan, and Zhu, Kenny Qili. Understanding Tables on the Web. In Atzeni, Paolo; Cheung, David W., and Ram, Sudha, editors, *ER*, volume 7532 of *Lecture Notes in Computer Science*, pages 141–155. Springer, 2012. ISBN 978-3-642-34001-7.

Wiederhold, Gio. Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38–49, March 1992. ISSN 0018-9162.

Winter, Pawel. Steiner Problem in Networks - A Survey. *Networks*, 17:129–167, 1987.