Exploiting Web Tables and Knowledge Graphs for Creating Semantic Descriptions of Data Sources

by

Binh Le Thanh Vu

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

August 2024

Dedication

To my wife, Nhan, and my parents for their encouragement, love, and patience.

Acknowledgements

I would like to thank my Ph.D. advisor, Craig A. Knoblock, very much for his support and guidance on my research. He spent countless hours with me discussing research ideas and reviewing my drafts. Through him, I learned how to conduct research, from selecting meaningful problems to presenting my research ideas. He gave me opportunities to attend conferences and workshops related to my field and has always been willing to help me with things not limited to research. I am very grateful to him for making my Ph.D. journey an enjoyable experience.

I would also like to thank all members of my proposal and dissertation committees for providing valuable feedback on my dissertation: Professor Sven Koenig, Professor Daniel Edmund O'Leary, Professor Yolanda Gil, Professor Jay Pujara, and Professor Muhao Chen. Their feedback and support is greatly appreciated. I also want to thank Professor Pedro Szekely and Professor Jay Pujara for greatly contributing to my research. They attended my weekly research meetings and gave me insightful ideas for the problems.

I would like to thank my colleagues at the Information Sciences Institute (ISI). Thanks to my officemates, Basel Shbita, Fandel Lin, and Kexuan Sun, who are always supportive and available to talk. Thank you, Karen Rawlins, for your administrative help during my years at ISI. Thank you, Minh Pham, for brainstorming my research and spending time with me. You are like a brother to

me. Thanks to my other USC Vietnamese friends for helping me out during the first few years at USC and making me feel connected.

I would like to thank my wife, Nhan, for her support and patience during my long Ph.D. journey. I am also thankful to my Mom and Dad, as well as my brother and sister, for always loving and caring for me.

This research was supported in part by the Army Research Office and the Defense Advanced Research Projects Agency under contract number W911NF-18-1-0027, and was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR00112390132 and Contract No. 140D0423C0093. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office, Defense Advanced Research Projects Agency, United States Air Force, or U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Table of Contents

List of Tables

List of Figures

Abstract

There is an enormous number of tables available on the web, and they can provide valuable information for diverse applications. To harvest information from the tables, we need precise mappings, called semantic descriptions, of concepts and relationships in the data to classes and properties in a target ontology. However, creating semantic descriptions, or semantic modeling, is a complex task requiring considerable manual effort and expertise.

Much research has focused on automating this problem. However, existing supervised and unsupervised approaches both face various difficulties. The supervised approaches require lots of known semantic descriptions for training and, thus, are hard to apply to a new or large domain ontology. On the other hand, the unsupervised approaches exploit the overlapping data between tables and knowledge graphs; hence, they perform poorly on tables with lots of ambiguity or little overlapping data.

To address the aforementioned weaknesses, we present novel approaches for two main cases: tables that have overlapping data with a knowledge graph (KG) and tables that do not have overlapping data. Exploiting web tables that have links to entities in a KG, we automatically create a labeled dataset to learn to combine table data, metadata, and overlapping background knowledge (if available) to find accurate semantic descriptions. Our methods for the two cases together provide a comprehensive solution to the semantic modeling problem. In the evaluation, our approach

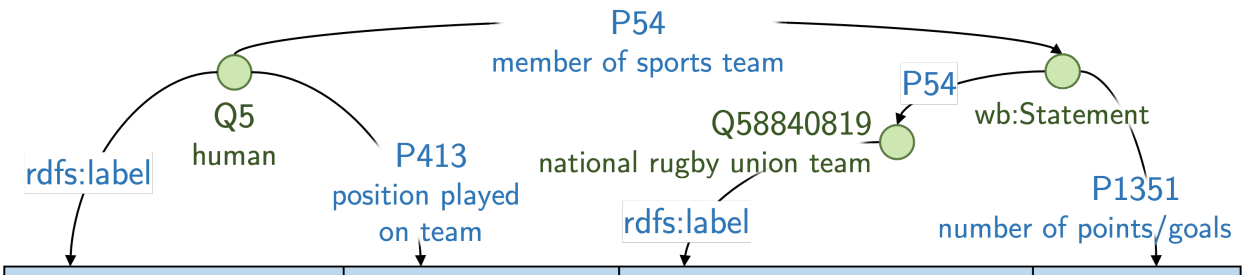
in the overlapping setting yields an improvement of approximately 5% in F_1 scores compared to the state-of-the-art methods. In the non-overlapping setting, our approach outperforms strong baselines up to 42% and 31% in F_1 scores for column type and relationship prediction, respectively.

Chapter 1

Introduction

Tables on the Web contain an immense amount of valuable information. However, they rarely provide descriptions of their content, making it difficult to use the information automatically. The semantic modeling task addresses this challenge by creating a semantic description that maps concepts and relationships in a table to classes and properties in a domain ontology. The resulting semantic description can be used both for data discovery and for publishing data to a knowledge graph [68].

A semantic description is expressed as a graph, where each node in the graph represents a column, an ontology class, or a literal (e.g., number, text, or date). Each edge presents an ontology predicate encoding relationships between the two nodes. Figure 1.1 shows an example table with its semantic description on top. For example, the edge `Q5 human` $\xrightarrow{\texttt{rdfs:label}}$ `Player` specifies that the `Player` column contains people's names. The edge `Q5 human` $\xrightarrow{\texttt{P413 position played on team}}$ `Position` provides the players' positions. More complex facts, such as the number of goals a player scored for their team (n-ary relationships), can easily be reified using a statement node `wb:Statement` that connects edges `P54 member of sports team` and `P1351 number of points` between the three columns `Player`, `Team`, and `Points`.



Player	Position	Team	Points
Dan Carter	Fly-half	New Zealand	1598
Ronan O'Gara	Fly-half	Ireland	1083
Percy Montgomery	Fullback	South Africa	893
Chris Paterson	Fullback	Scotland	809

Figure 1.1: Excerpt of a table of players of national rugby teams with its semantic description on top. Green circle nodes are ontology classes; blue cells on the first table row represent columns. The node `wb:Statement`, named statement node, is used to represent a ternary relationship between the players, their teams, and the number of points scored for the teams.

Since creating semantic descriptions requires significant effort and expertise, there is much research on automating this problem. In general, they can be classified into two groups. The first group is supervised methods trained on a set of known semantic descriptions with given domain ontologies [17, 59, 65]. Because the manually labeled dataset is expensive to obtain, these methods do not work well for new or large ontologies. The second group exploits the background knowledge in KGs to predict the semantic description [28, 37, 38, 41, 43, 50, 71]. First, they link table cells to entities in KGs. Then, they match the entities' property values with other table cells to find the semantic description. These approaches work for large ontologies and generally do not need to be retrained when the ontology is updated. They also tend to have only a few parameters to learn; hence, they typically need little to no training data. However, approaches in the second group tend to perform poorly on tables with lots of ambiguity such as having similar candidate entities or few matches. Due to the limited amount of training data, it is challenging for these

methods to learn the optimal parameters to combine the entity linking and data matching results to resolve the ambiguity.

In this work, we aim to address the aforementioned issues. We present three novel approaches for the semantic modeling problem for three progressively relaxed settings: (1) tables that already have links to entities in a knowledge graph (referred to as linked tables), (2) unlinked tables with overlapping data, and (3) unlinked tables without overlapping data with a knowledge graph.

1.1 Problem Definition

We define the problem of creating semantic descriptions as follows:

Definition 1.1 (Semantic Modeling Problem) *Given a target ontology $\mathcal{O}$ and a table $s(c_1, c_2, ..., c_n)$, in which c_i is a column, predict the semantic description $sm(s)$ of the table.*

Figure 1.1 shows a semantic description of the table using Wikidata's ontology. In this thesis, we generalize the concept of a table from 2-dimensional to data sources containing a list of tree-like records such as XML and JSON. Our methods and most existing approaches can be applied to these data sources without modification.

1.2 Thesis Statement

By exploiting knowledge from web tables and knowledge graphs, we can learn semantic descriptions of tables with little or no manually labeled training data.

1.3 Proposed Approach

Our work on semantic modeling can be divided into four main parts. First, we present a supervised approach that uses a probabilistic graphical model (PGM) [65]. A PGM allows us to express and capture the weak signals within the data. From a set of known semantic descriptions, we auto-generate positive and negative examples representing many possible descriptions of sources. Using this training set, the PGM is trained to distinguish between good and bad models. To predict the most probable semantic description for a data source, we use beam search to explore the space of possible models and evaluate the likelihood of the model using the trained PGM as a scoring function. However, the supervised approach suffers from the cold start problem: users need to label some tables to train the system first. Therefore, the remaining parts of our work aim to address this problem.

In the second part, we present GRAMS [66], a novel method for creating semantic descriptions of linked tables. Specifically, GRAMS constructs a candidate graph containing relationships between table columns and their context values by leveraging existing connections between data in the table and existing knowledge in a knowledge graph (KG) such as Wikidata. Then, incorrect relationships in the candidate graph are detected and removed using a Probabilistic Soft Logic (PSL) [2] model. Through collective inference, the PSL model favors links with high confidence, more informative, and consistent with constraints in the ontology and existing knowledge in the KG.

The third part of our work, GRAMS+, extends the previous one to support unlinked tables. GRAMS+ performs entity linking and creates a candidate graph of potential relationships using

the retrieved candidate entities. We train two neural networks (NN) using an automatically labeled dataset from Wikipedia tables to predict the likelihood of candidate entities and column relationships. Then, the two NNs are used to predict column types and column relationships to create a semantic description.

Finally, GRAMS and GRAMS+ both depend on the overlapping data between a table and a KG. To overcome this limitation, we present GRAMS++. GRAMS++ uses an NN model trained using distant supervision to estimate a similarity score of a given column and an ontology class or predicate. Then, the top K classes and predicates for table columns are combined to create a semantic description.

1.4 Contribution of the Research

In this thesis, we present comprehensive techniques for the semantic modeling problem under different settings and assumptions. Our research has the following main contributions:

- A supervised approach for creating semantic descriptions using previously modeled tables
- An unsupervised method for tables containing links to entities in a knowledge graph (KG)
- A distant supervised method for unlinked tables with overlapping data with a KG
- A distant supervised method for unlinked tables without overlapping data with a KG

The first two contributions are based on my earlier conference papers [65, 66]. The later contributions are based on my two latest papers; the former is submitted and under review, and the latter is under preparation.

1.5 Outline of the Thesis

The thesis is organized as follows. Chapter 2 introduces our supervised approach. Chapter 3presents our unsupervised approach for linked tables. Chapter 4 describes our distant supervised approach for tables that have some overlapping data with a knowledge graph (KG). Chapter 5 explains our solution for tables that do not have overlapping data with a KG. Chapter 6 discusses the related work on this problem. Finally, Chapter 7 summarizes the thesis and discusses important topics for future work.

In addition to the main chapters, we include in the appendices two auxiliary works that support the main research. Appendix 1 presents a novel data representation language [68] that can map a wide variety of format and layout heterogeneous data sources to a target format such as RDF [45]. The language is extensible and goes beyond the set of sources that existing mapping languages support. Appendix 2 shows an extensible user interface named SAND for creating and curating semantic descriptions semi-automatically. SAND is carefully designed to make it easy to integrate with different ontologies, knowledge graphs, and semantic modeling algorithms.

Chapter 2

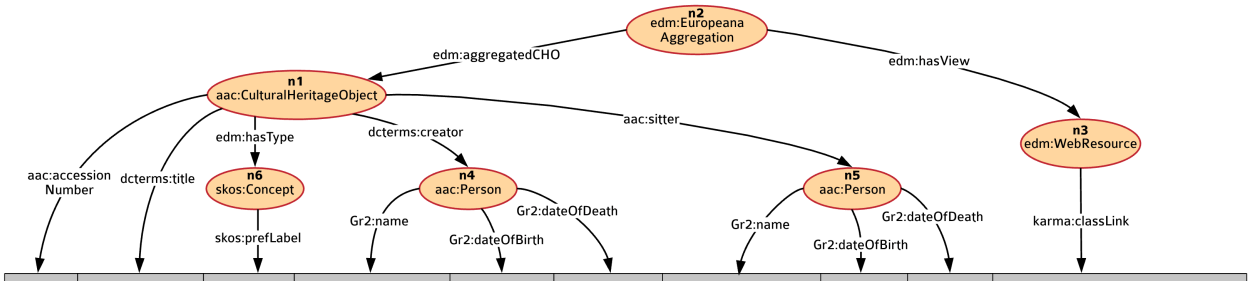
Learning Semantic Descriptions from Previously Modeled Tables

In this chapter, we introduce our supervised learning approach [65] for the semantic modeling problem (Definition 1.1). Our goal is to be able to create a complete semantic description to map a table to any chosen domain ontology.

Prior to this work, most approaches were limited to a specific ontology [38, 41, 63], or limited in their ability to infer relationships [38, 41, 46, 47, 51, 63] in a table. Previous work in the same setting [59, 62] creates the semantic description of a table by using predicted candidate semantic types of each column to build a minimum-weighted tree (Steiner Tree) that connects the columns together. A semantic type is a pair of ontology class and predicate. Heuristic functions and other machine learning methods, such as frequent pattern mining, are used to assign cost to edges of the tree. However, they do not perform well when the scores of candidate semantic types are similar. For example, it is easy to confuse a *date* column with the created date of paintings or artists' birth date. If we simply look at one signal from the cost of the edge, it would be hard to decide which one is correct because they are similar. However, there are also multiple signals from its

relationships with other remaining columns that can assist our decision-making. For instance, if we group the values of a *date* column by painting id and find that one painting appears to have been painted at two different points in time, then it is likely that this *date* column is not the created date.

To address these issues, we present a new approach to semantic modeling that uses a probabilistic graphical model (PGM). A PGM allows us to express and capture the signals within the data. From a set of known semantic models, we auto-generate positive and negative examples representing many possible descriptions of sources. Using this training set, the PGM is trained to distinguish between good and bad models. To predict the most probable semantic description for a table, we use beam search to explore the space of possible descriptions and evaluate the likelihood of the model using the trained PGM as a scoring function. Our contribution is a novel way to explore relationships within tables and semantic descriptions using a PGM, as well as a flexible framework that can easily be extended to incorporate new features. We present experimental results on two museum datasets published in Taheriyan et al. [59]. In the experiments, we control different levels of noise by using four different semantic labeling methods. These experiments show that our approach outperforms state-of-the-art systems by an average of 8.4% F_1 score, even with noisy input data.



ref	title	classification	artist	ArtistBornDate	ArtistDiedDate	name	bornDate	diedDate	image
NY270002	Chester Alan Arthur	Painting	Eastman Johnson	29 Jul 1824	5 Apr 1906	Chester Alan Arthur	5 Oct 1829	18 Nov 1886	http://npgportraits.si.edu/.../ny270002.jpg
NPG.70.36	Apollo 11 Crew	Painting	Ronald B. Anderson	1929	-	Neil Alden Armstrong	5 Aug 1930	25 Aug 2012	http://npgportraits.si.edu/.../7000062a.jpg
						Edwin Eugene Aldrin, Jr.	20 Jan 1930	-	
						Michael Collins	31 Oct 1930	-	
NPG.67.62	Chester A. Arthur	Painting	Ole Peter Hansen Balling	13 Apr 1823	1 May 1906	Chester Alan Arthur	5 Oct 1829	18 Nov 1886	http://npgportraits.si.edu/.../6700114a.jpg

Figure 2.1: NPG data source and its semantic description

2.1 Motivating Example

In this section, we provide an example of a table from the National Portrait Gallery (NPG*) museum to describe the typical process of building a semantic description, the shortcomings of existing approaches, and how signals within the data can be used to address them. Figure 2.1 shows the table data and its semantic description using classes and predicates from the DCTerms, SKOS, ElementsGr2, AAC, and EDM ontologies†.

The first step in building a semantic description is semantic labeling. A user or an automated semantic labeling system annotates each columns with semantic types. For example, the columns *title* and *image* are labeled as $\langle$*aac:CulturalHeritageObject*, *dcterms:title*$\rangle$, $\langle$*edm:WebResource*, *karma:classLink*$\rangle$, respectively. Note that we use a special predicate *karma:classLink* to indicate that the data node *image* contains URIs of *edm:WebResource*.

The second step is specifying relationships between nodes in the model. This is more complicated, as there are several paths to connect two nodes in the domain ontology. For instance, *aac:Person* can be *dcterms:creator* or *aac:sitter* of *aac:CulturalHeritageObject*. Another example is

*http://npg.si.edu/home/national-portrait-gallery
†These ontologies are used in the evaluation datasets

that *bornDate* and *artistBornDate* do not belong to the same person. If we only know the scores of semantic types of columns and how frequent a predicate is used to connect two class nodes, we may not know which path is correct.

When users model a table, they use various information to make decisions. For example, in Figure 2.1, we have three people who are listed under column *name* and are in the "Apollo 11 Crew" painting. As a painting is usually painted by one individual, a person would conclude that these are the names of the sitters rather than the names of the creators. The *artist* and *ArtistBornDate* are two columns next to each other. This is evidence indicating that they contain the name and birth date of the same person. If the *bornDate* is mislabeled as death date of *aac:Person*, we would be skeptical about having Eastman Johnson draw a painting of Chester A. Arthur when he was 5 years old. In the next section, we explain how our method learns to combine various signals to build a correct semantic description.

2.2 Probabilistic Graphical Models for Semantic Modeling

Our approach for building semantic descriptions is similar to the procedure humans use. We model table columns one at a time. For every column, we rank its modeling options based on the likelihood of the overall semantic description. Then, we select the column that has the highest rank. The process ends when there is no column left.

As we have discussed, users can exploit multiple signals to assist their decisions. For example, in Figure 1, given two options for relation $\langle n_1\_CulturalHeritageObject, n_4\_Person\rangle$: *dcterms:creator* and *aac:sitter*, users can use the fact that name of n_4_Person is linked to the attribute *artist* to select predicate *dcterms:creator* over *aac:sitter*. Those signals are often collective

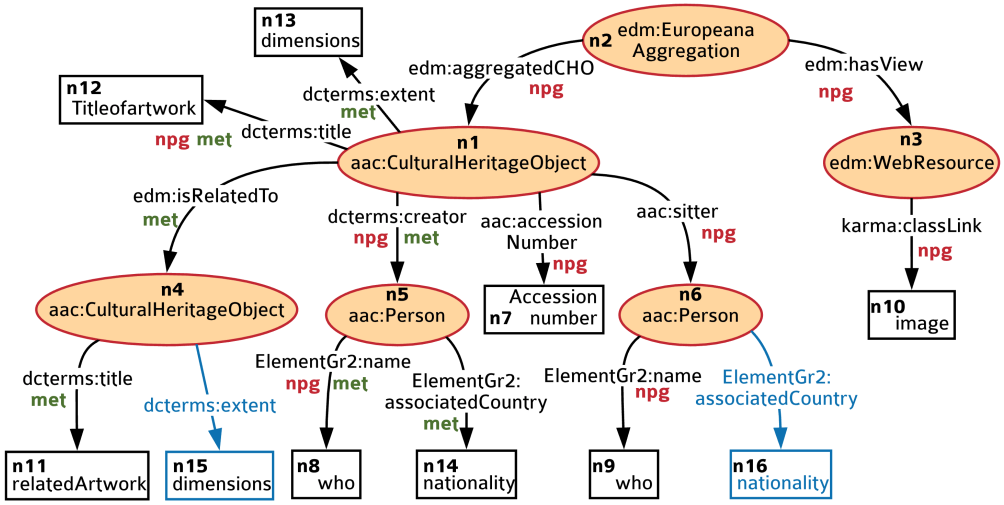


Figure 2.2: A example of integration graph[‡]. This is built from sm(npg) and sm(met). Some nodes and properties are hidden for readability.

and structural. They can be captured and represented naturally using a PGM, which is a very powerful framework that captures dependencies between relations and is robust to noise.

In Section 2.2.1, we describe the procedure to build a semantic description using search. We then explain how we apply and train a PGM as a ranking function in Section 2.2.2 and 2.2.3.

2.2.1 Searching for semantic descriptions

Let $D_0 = sm(\{\})$ be a start state, which is an empty description, and $f : D_0 \rightarrow D_1$ be a transition function to move from one state to another state by adding one source attribute. Then, in general, constructing a semantic description can be solved using search algorithms by repeatedly invoking the transition function f until we reach the goal state, a tree connecting all columns.

The transition function we use is defined in algorithm 1. We start with each unmerged column (line 2 - 3) and generate next states by merging them into current state (line 4). Algorithm 2 is used to combine a column with the current semantic description. The intuition of algorithm 2 is

[‡]This example is based on an example from [59]

Algorithm 1: TransitionFunction

Input: A search state: s_t
A set of columns: $C = \{c_0, ..., c_n\}$
An integration graph: G_{int}
Output: List of next search states

```
1 nextStates ← []
2 C′ ← C \ s_t.leafNodes
3 for c_i ← C′ do
4     trees ← MergeIntoTree (s_t, G_int, c_i)
5     for s_{t+1} ← trees do
6         nextStates.append(s_{t+1})
7 return nextStates
```

that we find all possible connecting paths between a column and leaves or root of the tree. Each path with a current tree forms a new state we want to generate.

To find the paths connecting a column and the existing tree, we use an integration graph from Taheriyan's work [59]. An integration graph is a directed weighted graph built from known semantic descriptions, and represents a space of plausible semantic descriptions we have seen in the training data. In Figure 2.2, we show an integration graph built from two tables *npg* and *met*. Each link is annotated with its table id to specify where it comes from. When the graph is used in prediction, it is expanded by adding semantic types depicted in blue color. For example, in a new table, we have a column called *dimensions* tagged with a semantic type ⟨*aac:CulturalHeritageObject*, *dcterms:extent*⟩, we add one data node n_{15} and one link *dcterms:extent* from n_4 to n_{15}. However, we do not need to update node n_1 because we have already had link $\langle n_1, \textit{dcterms:extent}, n_{13}\rangle$.

Now given an integration graph $G_{int} = \{V_{int}, E_{int}\}$, suppose that we are in the state s_t that we have four nodes $\{n_1, n_5, n_7, n_8\}$ as in Figure 2.3 and we want to merge column c_t = *title*. First, we need to align c_t to nodes in V_{int} that have same semantic type (line 2 in algorithm 2). The two nodes that match are n_{12} and n_{11}. For each node, we find all paths that connect the

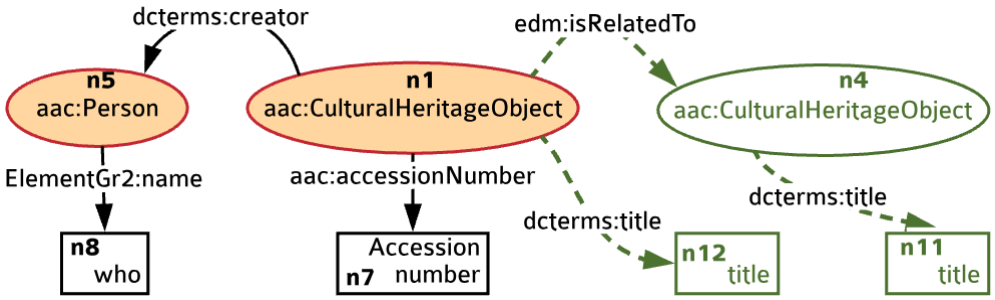


Figure 2.3: Two possible paths to add *title* attribute into current tree

node to s_t, each path generates different tree. For example, there are two paths ($n_1 \rightarrow n_{12}$) and ($n_1 \rightarrow n_4 \rightarrow n_{11}$). Therefore, we have two next states. Each next state will be a tree that combines s_t and a merge path.

Algorithm 2: MERGEINTOTREE

Input: A search state: s_t
An integration graph: G_{int}
An column: c_t
Output: List of trees which contain the given column c_t

```
1 newTrees ← []
2 mntPts ← G_int.findAttribute(c_t.semanticType)
3 for mntPt ← mntPts do
4     connectedPaths ← all paths that connect leaves or root of s_t with mntPt
5     for path ← connectedPaths do
6         newTree ← merge mntPt to s_t using path
7         newTrees.append(newTree)
8 return newTrees
```

In addition to using the transition function in the beam search, we can also use it in interactive semantic modeling systems such as Karma [31] or SAND [64]. In particular, from a current curated semantic description, the transition function is invoked to generate all semantic description candidates. Then, the candidates are ranked using a scoring function to produce the top-k

candidates for users to choose from. A user can also make corrections if there is no suitable option. In our approach, the scoring function directly affects the quality of our predicted semantic descriptions. In the next section, we will describe how we use PGM to learn a scoring function.

2.2.2 Using Graphical Models as the score function

A probabilistic graphical model (PGM) is a well-known and effective framework to represent a probability distribution of random variables [33]. In order to apply PGMs in semantic modeling, we choose two kinds of random variables: input variables $\boldsymbol{x}$ that are links between nodes in a semantic model, and output variables $\boldsymbol{y}$ that are labels of the links. A label of a link has possible values in $\mathcal{V} = \{\textit{true}, \textit{false}\}$. A link with label *true* means it is correct and present in the gold semantic description. Since we are interested in predicting labels of links: $P(\boldsymbol{y}|\boldsymbol{x})$, it is natural to use a conditional random field (CRF). In particular, our standard CRF has following form:

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x}_c)} \prod_{C_p \in \mathcal{C}'} \prod_{\Psi_c \in C_p} \Psi_c(\boldsymbol{y}_c, \boldsymbol{x}_c; \theta_p)$$

$$Z(\boldsymbol{x}_c) = \sum_{\boldsymbol{y}} \prod_{c \in \boldsymbol{C'}} \prod_{\Psi_c \in C_p} \Psi_c(\boldsymbol{y}_c, \boldsymbol{x}_c; \theta_p)$$

where $\Psi_c(\boldsymbol{y}_c, \boldsymbol{x}_c; \theta_p)$ is a factor function that takes two set of variables $\boldsymbol{y}_c$ and $\boldsymbol{x}_c$ with parameters θ_p, and $Z(\boldsymbol{x}_c)$ is a partition function acted as a normalization factor to ensure the distribution $P(\boldsymbol{y}|\boldsymbol{x})$ sum to 1. The factors that have same input structure are grouped into $\mathcal{C}' = \{C_1, C_2, ..., C_n\}$, where each C_i is a set of factors called *clique template* [58]. Factors in a clique

template C_p share same parameters θ_p. A common choice of factor functions is exponential function, and each factor is log-linear with a set of feature functions f as follows:

$$\Psi_c(\boldsymbol{y}_c, \boldsymbol{x}_c, \theta_p) = \exp\left\{\sum_k \theta_{pk} f_{pk}(\boldsymbol{y}_c, \boldsymbol{x}_c)\right\}$$

To simplify the notation, for every equation, x or y refers to one random variable, and $\boldsymbol{x}$ or $\boldsymbol{y}$ (bolded) refers to sets of random variables. Additionally, x and y symbols are used to denote input and output of a same link, respectively. For a complete guide about CRF, readers can refer to Sutton et al. [58].

Our ultimate goal is to construct a semantic description where all links are true. So the likelihood of the model being correct is $P(\forall y \in \boldsymbol{y}; y = \textit{true}|\boldsymbol{x})$, and can be used as a score function for ranking. With the CRF framework above, in the following subsections, we will describe different clique templates and their features, each capture different relations between tables and semantic descriptions.

2.2.2.1 Link between nodes

These are factors over one link representing the likelihood of the link being correct or not. In particular, the factors have this form: $\Psi(y, x)$. The first feature of the factor is the confidence score of a semantic type, which is one of the outputs from the semantic labeling step. The features are defined as:

$$f_{ij}^{SS}(y, x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{st(x)=j\}}\text{st\_score}(x); \forall i \in \mathcal{V}, j \in \mathcal{ST}$$

$$f_{ij}^{\overline{SS}}(y, x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{st(x)=j\}}(1 - \text{st\_score}(x)); \forall i \in \mathcal{V}, j \in \mathcal{ST}$$

where $\mathbf{1}_{\{*\}}$ is an indicator function, $\mathcal{ST}$ is set of all possible semantic types, $st(x)$, $st\_score(x)$ are the semantic type and confidence score of link x, respectively. These are typical features used in graphical models, where each feature has its own weight and is non-zero only when a particular condition is met.

Note that if we only use $f_{ij}^{SS}(y, x)$, then $\Psi(y = \textit{true}, x) - \Psi(y = \textit{false}, x) = \text{st\_score}(x)(\theta_{\textit{true},j} - \theta_{\textit{false},j})$, then $\Psi(y = \textit{true}, x) > \Psi(y = \textit{false}, x)$ is independent of $\text{st\_score}(x)$ but only weights of the features. By introducing $f_{ij}^{\overline{SS}}(y, x)$, we are able to assign higher energy to $\Psi(y = \textit{true}, x)$ when confidence score is high, and higher energy $\Psi(y = \textit{false}, x)$ when confidence score is low (assuming that the higher confidence score, the more chance y is true). The simple and yet efficient technique above is applied to many other signals which have continuous value in this paper.

The confidence score signal is only applied for data links. For class links, we estimate the probability of a link being correct as follows:

$$P(s, x, o) = P(s) * P(o) * P(x|s, o)$$
$$P(x|s, o) = \frac{count(s, x, o)}{count(s, o)}$$

where s, o are source node and target node of a link x in the semantic description, respectively. In the equation, the probability of nodes $P(s)$ and $P(o)$ are estimated using a logistic regression (LR) model, which is trained on the same training set of our graphical model. The intuition of two features used in the LR model is described below:

- Total confidence score of semantic types of all child data nodes of n: a class node is likely to be present in the model if it links to many data nodes.

- The merging cost of node n with another node n' that has the same label in the graph: two class nodes with disjoint sets of properties should be merged together to simplify the model.

Features of class links are defined similar to f_{ij}^{SS} as follow.

$$f_{ij}^{CL}(y, x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{\text{triple}(x)=j\}}P(\text{triple}(x)); \forall i \in \mathcal{V}, j \in \mathcal{T}$$

$$f_{ij}^{\overline{CL}}(y, x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{\text{triple}(x)=j\}}(1 - P(\text{triple}(x))); \forall i \in \mathcal{V}, j \in \mathcal{T}$$

where $\text{triple}(x) = (s, x, o)$, $\mathcal{T}$ is set of all triples.

Beyond the two features above, we also use other features which depend mostly on x. To define these features, we define a series of observation functions $o_t(x)$. The corresponding features have the form:

$$f_{it}^{LO}(y, x) = \mathbf{1}_{\{y=i\}}o_t(x); \forall i \in \mathcal{V}$$

The set of observation functions we used are listed in table 2.1.

Table 2.1: Observation functions used in factor of link of nodes

Features	Explanation
Delta_SS(x=j) $\forall j \in \mathcal{ST}$	Difference between score of current semantic type and the remained best semantic type
SS_rank(x)	Rank of a predicted semantic type (determined by semantic labeling method
P(x|s)	Frequency of a link x given its source node s
NoDataChild	The source node has only child class nodes
NoSiblings	The source node has only one child node
Prior	Prior probability of a link

2.2.2.2 Cardinality relationships

The cardinality constraint between two columns can be used to discover an incorrect description. For example, a painting should have only one primary title, i.e the id and primary title of a painting have a one-to-one relationship. If we observe that the relationship is one-to-many, then one of the properties is linked to the wrong column. We express the heuristic above using factors between two outgoing links of class nodes x_1 and x_2: $\Psi(y_1, y_2, x_1, x_2)$.

Cardinality relationship between column c_2 and column c_1 is computed by averaging the number of unique values of c_2 grouped by c_1. If the average number is greater than a threshold $\alpha = 1 + |\epsilon|, \epsilon \gtrapprox 0$, we consider this to be either a one-to-many or a many-to-many relationship, otherwise, it is a one-to-one relationship. The threshold is chosen to be slightly greater than 1 in order to make this feature robust to noise in the table (e.g., missing values). In particular, α is set to 1.05 based on experimental results. For example, in Figure 2.1, the cardinality of column *name* and column *ref* is $CRD(\text{ref}, \text{name}) = \text{1-to-many}$ because $\frac{1+3+1}{3} = 1.67 > \alpha$. If two columns belong to two separate nested lists, which occurs infrequently, we do not compute its cardinality relationship and set it to NULL. The cardinality features of this clique template are defined as:

$$f^{CRD}_{ijkmn}(y_1, y_2, x_1, x_2) = \mathbf{1}_{\{y_1=i\}}\mathbf{1}_{\{y_2=j\}}\mathbf{1}_{\{st(x_1)=k\}}\mathbf{1}_{\{st(x_2)=m\}} \mathbf{1}_{\{CRD(x_1,x_2)=n\}} ; \forall i, j \in \mathcal{V}, k, m \in \mathcal{ST}, n \in \mathcal{CRD}$$

where $\mathcal{CRD} = \{\text{many-to-many}, \text{1-to-many}, \text{1-to-1}, \text{NULL}\}$.

2.2.2.3 Properties co-occurrence

This clique template gives a boost to properties that are likely to occur together. For example, birth date and death date of an artist have higher co-occurrence than birth date and creation date of a painting. If we have two dates columns, the chance of their being birth date and creation date will be lower than the chance that they are birth date and death date.

A co-occurrence frequency matrix is estimated from known semantic descriptions and are used as features in this clique template:

$$
\begin{aligned}
f_{ijmn}^{OCC}(y_1, y_2, x_1, x_2) =& \mathbf{1}_{\{y_1=i\}}\mathbf{1}_{\{y_2=j\}}\mathbf{1}_{\{st(x_1)=m\}}\mathbf{1}_{\{st(x_2)=n\}} \\
& \text{co\_occurrence}(x_1, x_2) \\
& ; \forall i, j \in \mathcal{V}, m, n \in \mathcal{ST} \\
f_{ijk}^{\overline{OCC}}(y_1, y_2, x_1, x_2) =& \mathbf{1}_{\{y_1=i\}}\mathbf{1}_{\{y_2=j\}}\mathbf{1}_{\{st(x_1)=m\}}\mathbf{1}_{\{st(x_2)=n\}} \\
& (1 - \text{co\_occurrence}(x_1, x_2)) \\
& ; \forall i, j \in \mathcal{V}, m, n \in \mathcal{ST}
\end{aligned}
$$

where x_1 and x_2 are sibling links of a class node.

2.2.2.4 Duplicated properties

An entity usually has many properties, but a few of them are duplicated. An example of a duplicated property is a class node *Person* with two links *worksFor* to two data nodes *organization1* and *organization2*. As semantic labeling systems often output the same semantic types for similar attributes, knowing which properties could be duplicated helps us identify incorrect links.

Factors in this clique template take a set of same label outgoing links of a class node and have the following features:

$$f_{i0}^{DP}(\boldsymbol{y}_t, \boldsymbol{x}_t) = \mathbf{1}_{\{|\{\forall y_i \in \boldsymbol{y}_t | y_i = true\}|=1\}} \mathbf{1}_{\{st(\boldsymbol{x}_t)=i\}}; \forall i \in \mathcal{ST}$$

$$f_{i1}^{DP}(\boldsymbol{y}_t, \boldsymbol{x}_t) = \mathbf{1}_{\{|\{\forall y_i \in \boldsymbol{y}_t | y_i = true\}|>1\}} \mathbf{1}_{\{st(\boldsymbol{x}_t)=i\}}; \forall i \in \mathcal{ST}$$

2.2.2.5 Grouping properties

In hierarchical data sources, properties of an entity are usually grouped under the same nested object. For example, birth date and name of person are properties of the *artist* object and *artist* is in a record object in a data source: {"artist": {"birth_date": "..", "name": ".."}, "title": ".."}.

Let SameScope(x_1, x_2) be a function of two sibling links that output *true* if their target data nodes are under the same nested object and *false* otherwise. Then, the features of this clique template are defined as:

$$f_{ij}^{GP}(y_1, y_2, x_1, x_2) = \mathbf{1}_{\{y_1=i\}} \mathbf{1}_{\{y_2=i\}} \text{SameScope}(x_1, x_2); \forall i, j \in \mathcal{V}$$

$$f_{ij}^{\overline{GP}}(y_1, y_2, x_1, x_2) = \mathbf{1}_{\{y_1=i\}} \mathbf{1}_{\{y_2=i\}} \neg \text{SameScope}(x_1, x_2); \forall i, j \in \mathcal{V}$$

2.2.2.6 Structural similarity

There are many possible ways to link two class nodes. Class nodes involved in the same relationship often connect to a similar set of data nodes. For example, in Figure 2.4, *Concept* nodes n_2, n_5 and n_8 link with *CulturalHeritageObject* nodes via the predicate *hasType* have similar values, which are types of artworks. While *Concept* nodes n_3 and n_6 involved in different relations (*dcterms:subject*) contain values about subjects in artworks. This heuristic can be used to predict the likelihood of links between class nodes. In particular, we define factor functions taking two

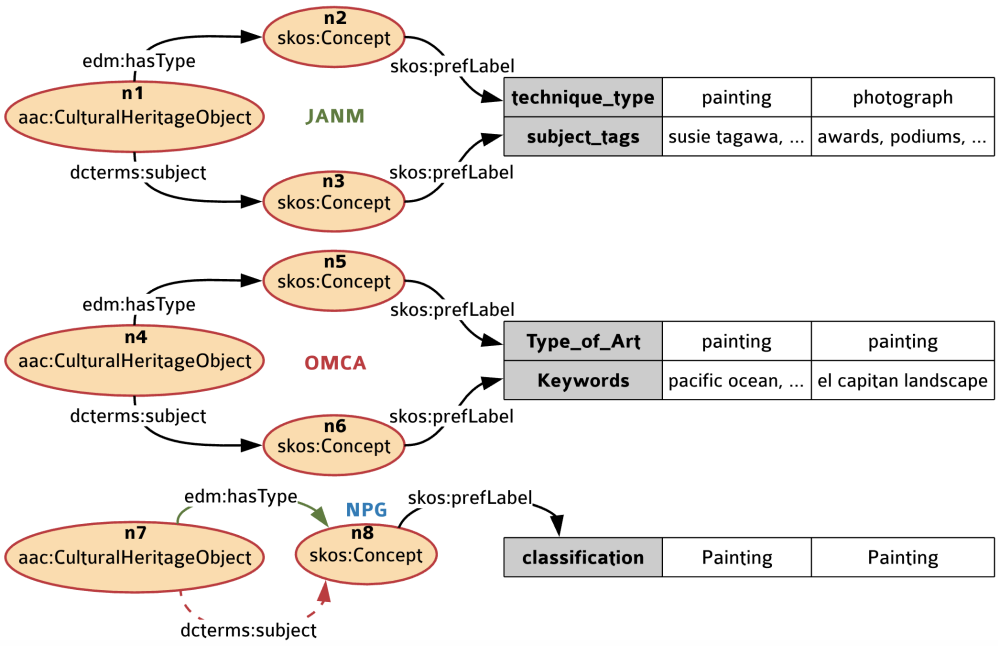


Figure 2.4: Excerpt from semantic descriptions of the JANM, OMCA[§]and NPG data sources

links x_1 and x_2, which are class link and data link, respectively. The input x_1 and x_2 form a substructure. We say two substructures are partially matched if they both have the same link x_2. For example, in Figure 2.4, we have two substructures: $s_1 = \langle$*CulturalHeritageObject*, *hasType*, *Concept*, *prefLabel*, *classification*$\rangle$ and $s_2 = \langle$*CulturalHeritageObject*, *subject*, *Concept*, *prefLabel*, *classification*$\rangle$, in which $s2$ and $s1$ are partially matched.

The likelihood of a substructure is the maximum similarity score between its column with other columns of other identical substructures in the training set. The reward of the substructure, Reward(x_1, x_2), is defined to be the difference between the likelihood of current substructure with the highest likelihood of other partial matched substructures. For example, in Figure 2.4, assuming the similarity score between column *classification* and columns *technique_type*, *subject_tags*, *type_of_art*, *keywords* are 0.7, 0.5, 0.8, 0.2, respectively. Then, the likelihood of substructure

[§]http://www.janm.org/ and http://museumca.org/

$s_1 = max(0.7, 0.8) = 0.8$, and the likelihood of substructure $s_2 = 0.5$. The reward of the substructure $s_1 = 0.8 - 0.8 = 0$ while reward of $s_2 = 0.5 - 0.8 = -0.3$. Hence, our CRF will prefer s_1 to s_2.

Using Reward(x, y) function, the features of this clique template are defined as follow:

$$f_{ij}^{SrS}(y_1, y_2, x_1, x_2) = \mathbf{1}_{\{y_1=i\}}\mathbf{1}_{\{y_2=i\}}\text{Reward}(x_1, x_2); \forall i, j \in \mathcal{V}$$

$$f_{ij}^{\overline{SrS}}(y_1, y_2, x_1, x_2) = \mathbf{1}_{\{y_1=i\}}\mathbf{1}_{\{y_2=i\}} - \text{Reward}(x_1, x_2); \forall i, j \in \mathcal{V}$$

2.2.2.7 Error propagations.

The factors of this clique template are used to remove class nodes in which all of their outgoing links are labeled *false* because they are redundant. In particular, the factors need to output high energy when all links of a class node are incorrect, and low energy when all of the outgoing links have label *false* but the incoming link is labeled *true*. Features of this clique template are defined as follow:

$$f_i^{EP}(y_0, \boldsymbol{y}_c, x_0, \boldsymbol{x}_c) = \mathbf{1}_{\{y_0=i\}}\mathbf{1}_{\{y=\text{false};\forall y \in \boldsymbol{y}_c\}}; \forall i \in \mathcal{V}$$

where x_0 is the incoming link, $\boldsymbol{x}_c$ are the outgoing links.

2.2.3 Training the Graphical Model

To train the graphical model, beside positive examples obtained from set of known semantic descriptions, we also need negative examples. A typical approach to generate more training data is collecting generated semantic descriptions from the search procedure. Then, every link in the generated models is labeled *true* or *false* using an automated labeling procedure.

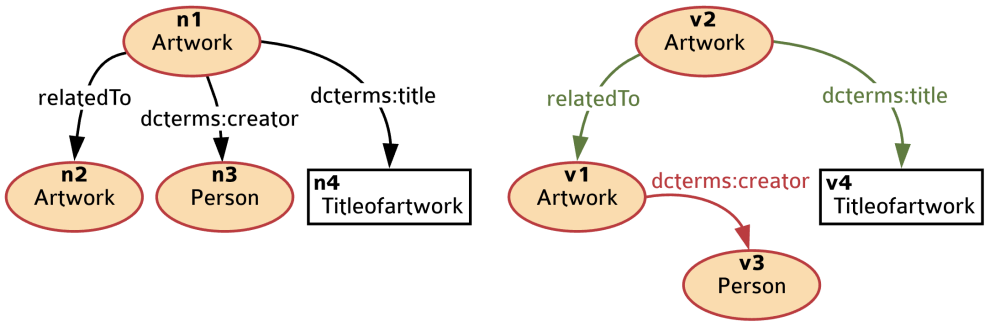


Figure 2.5: Example of a gold description (left) and predicted description (right). The green lines in the predicted description show correct links and red ones are incorrect links.

2.2.3.1 Auto-label examples

Define rel(sm) is a set of triples (u, e, v) of semantic description sm, in which u, v is a source node and target node of a link e, respectively. The labeling procedure works as follows. First, we find the best mapping from a node in the predicted description sm' to a node in known description sm such that it maximizes the number of overlapped triples between rel(sm) and rel(sm'). For example, in Figure 2.5, if the mapping is $\{n_1 \rightarrow v_2, n_2 \rightarrow v_1, n_3 \rightarrow v_3, n4 \rightarrow v_4\}$, we have 1 overlapping triple $\langle$*Artwork*, *dcterms:title*, *Titleofartwork*$\rangle$. If we switch the mapping between two nodes n_1 and n_2, we have 2 overlapping triples and this is the best mapping we can have. After the alignment step, each link e in predicted description sm' is assigned a true label if its triple (u, e, v) is in rel(sm); otherwise, it is assigned *false*.

2.2.3.2 Generate training examples

To make the process of generating training examples efficient, instead of starting from an empty semantic description, we start from a known semantic description, remove some data nodes, then invoke the search algorithm in Section 3.1 to add back the removed columns. Algorithm 3 describes how we remove data nodes. The goal is instead of removing random nodes, we remove the

Algorithm 3: EliminateDataNodes

Input: A gold semantic description: sm
Output: List of new semantic descriptions in which some data nodes have been removed

```
startStates ← []
sm_new ← remove data nodes in sm, which semantic labeling doesn't predict their
  semantic types correctly
for u ← sm_new.dataNodes do
    for stype ← PredictedSemanticTypes (u) do
        V_match ← all the data nodes have semantic type stype
        if |V_match | = 0 then
            state ← remove u out of sm_new
            add state to startStates
        for v ← V_match do
            if u is not v then
                state ← remove u and v out of sm_new
                add state to startStates
return startStates
```

nodes that have same semantic types, so that when we add back a data node, there is always more than one possible way we can merge the data node into the model, hence generate a hard example for our graphical model to learn. For example, in Figure 2.1, suppose that we are removing the column *artist*, and *artist* is tagged with two possible semantic types: $\langle$*CulturalHeritageObject*, *title*$\rangle$ and $\langle$*Person*, *name*$\rangle$. After removing, we have two possible starting states: the first one is the graph missing the links $\langle n_1$_*CulturalHeritageObject*, *title*$\rangle$ and $\langle n_4$_*Person*, *name*$\rangle$, the second state is omitting the links $\langle n_4$_*Person*, *name*$\rangle$ and $\langle n_5$_*Person*, *name*$\rangle$.

Note that as there are many possible examples that can be generated, each example is associated with a score computed using the Taheriyan et al. method [59], then we randomly sample the examples using the score to reduce the number of training examples. In our experiments, the number of examples is set to 300 per gold semantic model.

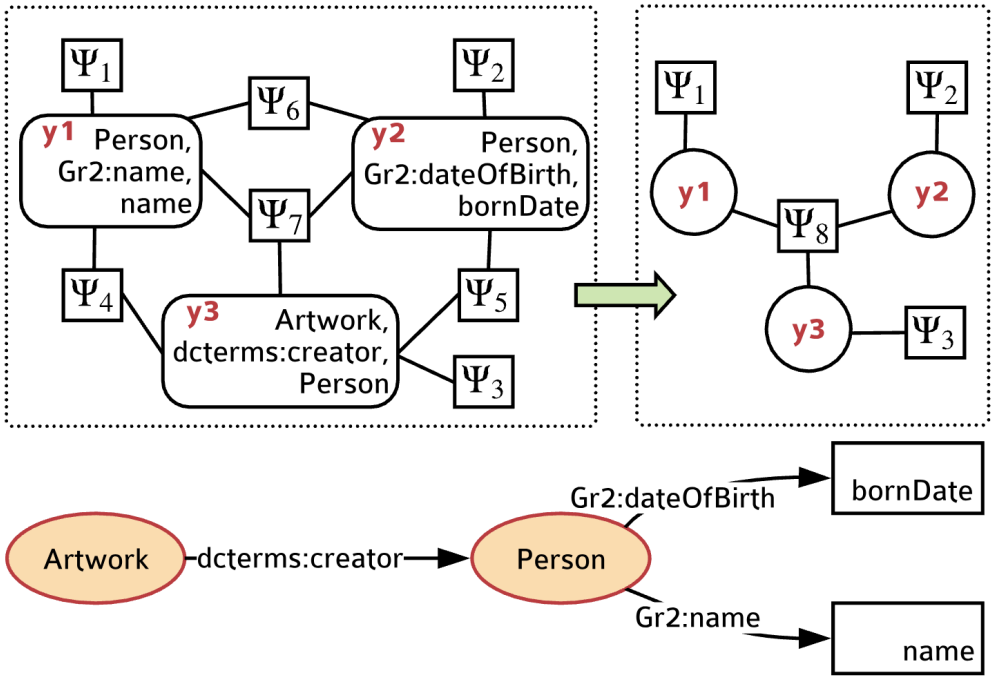


Figure 2.6: Factor graphs before and after eliminating cycles. In the factor graphs, we omit random variables x because their values are fixed.

2.2.3.3 Inference and Parameter Learning

We perform exact inference on CRF using Belief Propagation [58]. Undirected graphical models are often represented as factor graphs in which random variables and factor functions are nodes. An edge is drawn between a random variable and a factor if the variable is an input of the factor function. Since we can only apply Belief Propagation to acyclic factor graphs, we eliminate cycles in our factor graphs by merging cliques which form cycles into single cliques. With the choices of factors in Section 2.2.2, we only need to merge factors whose inputs are incoming links and/or outgoing links of the same class nodes. For example, in the factor graph in Figure 2.6, the graph

becomes acyclic after the four factors Ψ_4 to Ψ_7, which form cycles, are merged and replaced by a new factor:

$$\begin{aligned}\Psi_8(y_1, y_2, y_3, x_1, x_2, x_3) =& \Psi_4(y_1, y_3, x_1, x_3)\Psi_5(y_2, y_3, x_2, x_3) \\ & \Psi_6(y_1, y_2, x_1, x_2)\Psi_7(y_1, y_2, y_3, x_1, x_2, x_3)\end{aligned}$$

The problem with this approach is that the merged factors may have many variables, which results in the number of values of $\boldsymbol{y}$ we need to enumerate increasing exponentially with n: 2^n, where $n = |\boldsymbol{y}|$. Fortunately, it is not common to have a class node containing too many properties. In order to ensure the inference algorithm has reasonable runtime, we could control the number of variables in the factor by keeping top $n' < n$ most frequently used predicates. In the experiments, we set $n = 11$ so there are only maximum 2048 combinations of $\boldsymbol{y}$ values.

Our CRF is trained using the standard maximum log-likelihood and Belief Propagation. For more details about Belief Propagation and training CRF, readers can refer to the tutorial in Sutton et al. [58].

2.3 Experimental Evaluation

2.3.1 Dataset and experimental setup

Our objective is to assess the ability of our method to infer correct semantic descriptions with and without noisy data. We evaluate our approach, which is called PGM-SM, on two different datasets: ds_{edm} and ds_{crm} from Taheriyan et al. [59]. Both datasets contain data sources from different art museums in the US and in different data formats (CSV, XML, JSON, etc). However,

Table 2.2: The evaluation datasets ds_{edm} and ds_{crm}

	ds_{edm}	ds_{crm}
#data sources	28	28
#classes in domain ontologies	119	147
#properties in domain ontologies	351	409
#nodes in the gold-standard models	444	970
#data nodes in the gold-standard models	310	552
#class nodes in the gold-standard models	134	418
#links in the gold-standard models	416	942

domain experts use two different well-known data models in the museum domain: Europeana Data Model (EDM), and CIDOC Conceptual Reference Model (CIDOC-CRM) for knowledge representation. The details of the two datasets are listed in Table 2.2. Note that we updated the semantic descriptions in the ground-truth datasets to fix some mistakes based on the recommendation[¶] from domain experts [32]. The updated datasets, experimental results, and our code are available on Github[‖].

Semantic labeling methods are evaluated using standard mean reciprocal rank (MRR) [9]. We assess the quality of predicted semantic descriptions by comparing them with gold semantic descriptions in terms of precision and recall as in Taheriyan et al. [59]:

$$\text{precision} = \frac{|rel(sm) \cap rel(f^*(sm'))|}{|rel(f^*(sm'))|}$$

$$\text{recall} = \frac{|rel(sm) \cap rel(f^*(sm'))|}{|rel(sm)|}$$

$$f^* = \underset{f}{\text{argmax}}\, |rel(sm) \cap rel(f(sm'))|$$

where $rel(sm)$ is a set of triples (u, e, v) of a semantic description sm, f is a mapping function that maps nodes in sm' to nodes in sm (Section 2.2.3.1).

[¶]The recommendation documentation can be found at: http://review.americanartcollaborative.org/

[‖]https://github.com/binh-vu/semantic-modeling

Table 2.3: MRR scores of three semantic labeling methods

Datasets	DSL	Serene	SemTyper
ds_{edm}	0.886	0.912	0.830
ds_{crm}	0.896	0.910	0.628

In our experiments, we use half of the dataset for training and the other half for testing. We repeat the process three times (3-fold) and average the results. Parameters of our CRF are learned using the ADAM-AMSGRAD optimization method [48] for 60 epochs with batch size 200 and a learning rate of 0.05. Our experiments are run on a single machine with Intel Xeon E5-2620 v4 and 32GB RAM.

2.3.2 Automatic semantic modeling

In this experiment, we evaluate our method on four different semantic labelers: SemTyper [47], DSL [46], Serene [51] and Oracle. Each semantic labeler has different performance and characteristics and thus provides different levels of noise to our system. Of the three methods, SemTyper has the lowest MRR score, indicating that its output includes many incorrect semantic types. DSL has a higher MRR score and provides fewer incorrect semantic types. However, the confidence score between predicted semantic types is similar since DSL predicts semantic labels based on a similarity notion. Serene has the best MRR performance, and the confidence score between correct and incorrect semantic types is very different. The Oracle semantic labeler is a labeler that always outputs the correct semantic type; hence, it provides no noise to the semantic modeling system. The performance of these semantic labelers is reported in Table 2.3.

We compare our method with two state-of-the-art semantic modeling systems: Taheriyan et al. [59] and Uña et al. [62] (Serene). The experiment results are reported in Table 2.4. Generally,

Table 2.4: Performances of semantic modeling systems with respect to different semantic labelers

Datasets	Labelers	Precision			Recall			F1		
		Taheriyan	Serene	PGM-SM	Taheriyan	Serene	PGM-SM	Taheriyan	Serene	PGM-SM
ds_{edm}	SemTyper	0.726	0.689	0.772	0.702	0.701	0.767	0.712	0.693	0.768
	DSL	0.656	0.708	0.812	0.618	0.734	0.820	0.635	0.719	0.815
	Serene	0.808	0.777	0.822	0.800	0.803	0.837	0.803	0.789	0.829
	Oracle	0.883	0.876	0.945	0.892	0.896	0.927	0.887	0.885	0.935
ds_{crm}	SemTyper	0.695	0.710	0.809	0.559	0.624	0.660	0.618	0.663	0.725
	DSL	0.699	0.714	0.851	0.693	0.687	0.839	0.695	0.698	0.844
	Serene	0.779	0.736	0.886	0.770	0.773	0.875	0.774	0.753	0.880
	Oracle	0.869	0.838	0.965	0.847	0.846	0.928	0.857	0.840	0.944

the F_1 score increases with less noise for three systems. Among them, our approach outperforms the two baseline methods with the average increase by 6.53% and 10.92% on ds_{edm} and ds_{crm}, respectively. As both baseline methods predict semantic descriptions by finding the minimum-weighted tree, they would be affected if the weights of edges are similar. Therefore, they are more sensitive to the noise of DSL. Our approach, however, uses the collective signals and achieves significant improvement, which indicates it is more robust to noise.

Table 2.5 shows the average F_1 score of PGM-SM when the number of known semantic descriptions is 25% and 50% of the datasets. As the training size increases, the performance of PGM-SM increases. The average percentage of performance gain is 2.7%, which suggests that the model achieves reasonable performance even when the training size is small (25%). Comparing to the two baseline methods Taheriyan et al. [59] and Serene in the same scenario, PGM-SM improves the F_1 score on average 7.68% and 9.20%, respectively.

Table 2.5: Average F_1 score of PGM-SM with respect to different training size

Datasets	25% of dataset	50% of dataset
ds_{edm}	0.808	0.837
ds_{crm}	0.823	0.848

Table 2.6: Average running time of PGM-SM on two different datasets.

Datasets	Training time	Testing time
ds_{edm}	162.47s	9.54s
ds_{crm}	315.74s	59.37s

In addition, we measure the running time of our approach in terms of training and testing time listed in Table 2.6. The training time starts from generating training examples to finishing training PGM-SM. The testing time is the total time used to predict semantic descriptions of the testing data sources. Since the size of semantic descriptions in ds_{crm} are twice as big as the models in ds_{edm} (Table 2.2), we can see that the training time of ds_{crm} is nearly double ds_{edm}. The testing time does not increase correspondingly because it depends not only on the runtime of PGM but also on the runtime of the search algorithm, which increases linearly with the number of attributes in the data sources.

2.3.3 Feature analysis

To understand the impact of each feature on the result, we perform ablation analysis by removing the factors described in Section 2.2.2. As the results in Table 2.7 have shown, dropping some factors sometimes slightly increases the F_1 score. In particular, STRUCT-SIM and ERROR-PROP are important in dataset ds_{crm}, while they are not in dataset ds_{edm}. The reason for this phenomenon is that the semantic descriptions in ds_{crm} are very complex and structural while semantic descriptions in ds_{edm} are not. For example, in ds_{edm}, the creation date of an artwork is modeled as a literal date time value. While in ds_{crm}, the date is represented as a class *E52_Time-Span* to capture the fact that it could take an artist many years to create an artwork. As those factors have little effect in ds_{edm}, dropping them actually makes the CRF model learn better parameters of the

Table 2.7: Ablation analysis of the factors in PGM-SM. Each cell value is a difference in F_1 score upon dropping one factor.

Factor	Factor name	ds_{edm}				ds_{crm}			
		SemTyper	DSL	Serene	Oracle	SemTyper	DSL	Serene	Oracle
Structural Similarity	STRUCT-SIM	-0.009	0.004	0.020	0.008	-0.006	-0.013	-0.001	-0.015
Error Propagation	ERROR-PROP	0.005	0.005	0.004	0.000	-0.006	-0.016	-0.001	0.003
Co-occurrence	CO-OCCUR	-0.006	-0.014	-0.008	-0.005	0.024	-0.008	0.010	0.004
Grouping properties	GROUPING	0.008	-0.011	-0.003	0.008	-0.007	-0.012	0.004	0.003
Duplicated properties	DUP-PROP	-0.003	-0.009	-0.005	0.000	-0.007	-0.008	-0.003	-0.007
Cardinality relationships	CARDINALITY	-0.006	-0.020	-0.013	-0.023	-0.014	-0.049	-0.022	-0.032

remaining factors. On the other hand, CO-OCCUR and GROUPING factors show the opposite trend. DUP-PROP and CARDINALITY have a positive impact on the F_1 score for all datasets, and CARDINALTIY is the most important feature in our CRF model.

2.4 Summary

In this chapter, we presented a novel approach for semantic modeling that uses a probabilistic graphical model to exploit relationships within tables and semantic descriptions. From a set of tables and their semantic descriptions, we train a conditional random field (CRF) to distinguish between good and bad semantic descriptions. Then, we use the CRF to efficiently search over the combinatorial space of possible semantic descriptions to identify the most probable model for a table. The evaluation shows that by exploring the relationships within data, our approach generates better semantic descriptions and is more robust to noise than previous approaches.

Chapter 3

Creating Semantic Descriptions of Linked Tables

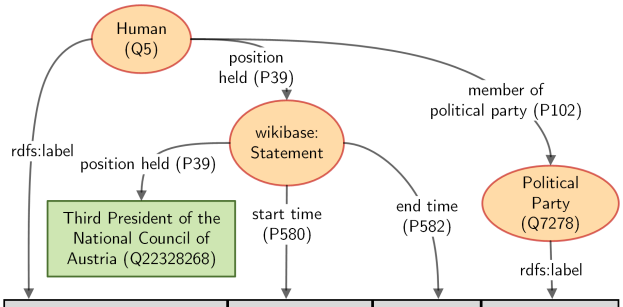
In the previous chapter, we presented a supervised approach for the semantic modeling problem. Although our method can be applied to a given domain ontology, it requires previously modeled tables as training data, which may not be readily available in many domains. On the other hand, there are millions of high-quality tables containing links to entities in knowledge graphs (KGs), such as Wikipedia tables. These links might be useful in creating accurate semantic descriptions of the tables. By doing so, not only can the semantic descriptions be used to add or keep the knowledge in KGs up-to-date, but they also can be used as a large training dataset to address the aforementioned weakness of the supervised approach.

As KGs play a vital role in many applications, it is desired to have an automatic method to integrate data sources to enrich KGs. However, existing work on mapping tables to KGs [10, 43, 50, 54] has two main limitations. First, their methods only consider values inside the tables but not values in the surrounding context. We found that in many tables the implicit contextual values are critical to understanding the semantics of a table. For example, a table about cast members of a movie and their roles typically does not have the movie in the table data since it is mentioned in the context. Second, they do not deal with n-ary relations needed to accurately

and fully represent knowledge in the tables. Examples of n-ary relations are a politician elected to an office position from an electoral district or sales of a company reported in a particular year.

To address these issues, we present a new approach for semantic modeling that uses graphs to represent possible (n-ary) relationships in the tables and collective inference to eliminate spurious relationships on the graphs. Specifically, we construct a candidate graph containing relationships between table columns and the table context values by leveraging possible connections between data in the table and existing knowledge in a KG such as Wikidata. Then, incorrect relationships in the candidate graph are detected and removed using a Probabilistic Soft Logic (PSL) [2] model. Through collective inference, the PSL model favors edges with high confidence, is more informative, and is consistent with constraints in the ontology and existing knowledge in the KG. To assess the effectiveness of our method, we evaluate our method on a real-world dataset of Wikipedia tables and on a synthetic dataset from the SemTab2020 challenge [25]. These experiments show that our method outperforms the state-of-the-art systems on the real-world dataset by 12.6% and 4.8% average F_1 scores on relationships and concept prediction tasks, respectively, while achieving similar F_1 scores on the synthetic dataset (the differences are approximately 0.1%).

Our contribution is a novel graph-based method for semantic modeling that collectively determines correct relationships between two or more columns and implicit contextual values using PSL. Our solution offers two key technologies: (i) an algorithm to construct a graph of plausible semantic descriptions of tables using external knowledge from a KG; and (ii) a probabilistic model that utilizes features from external knowledge and related relationships in the graph for robust relationship prediction.



Name	Entered Office	Left Office	Party
Willi Brauneder	1996	1999	FPÖ
Thomas Prinzhorn	2002	2006	FPÖ
Eva Glawischnig-Piesczek	2006	2008	Grüne

Figure 3.1: A table of third presidents of the National Council of Austria with its semantic description on top. The node `wikibase:Statement` is used to represent an n-ary relationship of a position (Third President) and its start and end time. The position is not in the table but is introduced via an entity (the green node).

3.1 Motivating Example

In this section, we provide a real example to explain how links in the tables can be used for semantic modeling. The example also demonstrates that relying solely on simple data matching can be incorrect and a collective approach is necessary. Figure 3.1 shows a snippet of a table about the third presidents of the National Council of Austria and its semantic description built using Wikidata on top. For readability, we interchangeably refer to Wikidata entities, classes (Qnodes), and properties (Pnodes) either by their labels and ids (e.g., `Human (Q5)`) or just by their ids (e.g., `Q5`).

In the figure, yellow nodes represent ontology classes, green nodes represent entities or literals, and edges are ontology properties. For example, the edge `rdfs:label` between the node `Human (Q5)` and the first column depicts that each cell in the column is a person whose name is specified in the value of that cell. Similarly, the edge `P102` between the class `Human (Q5)` and

Political Party (Q7278) states that each person is a member of the corresponding political party. The property position held (P39) connects the node Q5 to an entity Q22328268 and columns Entered Office and Left Office to describe the time each person holds the third president position. This is an n-ary relationship and is represented by an intermediate wikibase:Statement node. Note that in Wikidata every claim is represented as a statement, so there is a statement node for the relationship P102 of node Q5 and node Q7278. However, since this is a binary relationship, we have omitted the statement node for conciseness.

In the table, some cells are linked to Wikipedia articles such as Eva Glawischnig-Piesczek (third row, first column). By querying Wikidata to obtain a Qnode associated with the Eva Glawischnig-Piesczek article, we know that she was the third president of the National Council of Austria (Q22328268) from 2006 to 2008. As the information appears in the same row of the second and third columns, this suggests that start time (P580) and end time (P582) could be the relationships between those columns and of an n-ary relationship position held (P39) of Q22328268. Following this process, we may discover in the second row that Thomas Prinzhorn was a second president, and he left the office in 2002, while there may be no suggestions from data in the first row as Wilhelm Brauneder does not link to any Qnode.

From this example, we observe that matching table data to KGs can suggest correct semantic descriptions. Yet predictions solely relying on data matching can be imprecise. To go beyond simple data matching, we develop a graph-based approach that uses a probabilistic graphical model to combine evidence from external knowledge and related possible matched relationships to predict the most probable semantic description.

3.2 Approach

The problem of finding semantic descriptions of linked tables makes two assumptions to the semantic modeling task at 1.1: (1) each cell $c_{i,j}$ of row i, column j of an input table may contain links to entities in a KG, and (2) and the target ontology is the ontology of the KG. In this work, we focus on linked tables from Wikipedia and choose Wikidata as the target KG instead of DBpedia. The reason is that Wikidata is a huge crowdsourced KG with millions of classes and thousands of properties. It is designed to capture and represent complex knowledge in many domains.

Our approach consists of two main steps. The first step is to build a candidate graph of relationships between columns and context values. Then we use collective inference to identify correct relationships and correct types of columns containing entities (called entity columns) to create a final semantic description.

Preprocessing Since we use Wikidata as the target KG, the semantic description will be described in terms of the Wikidata ontology. Classes of the ontology are all Qnodes participating in the `subclass of (P279)` relationship, and properties of the ontology are all Wikidata properties and the `rdfs:label` property. Also, cells in the Wikipedia tables are not directly linked to Wikidata entities but have hyperlinks to Wikipedia articles. Thus we apply a preprocessing step to automatically convert the hyperlinks to Wikipedia articles to Wikidata entities using Wikidata sitelinks.

3.2.1 Constructing Candidate Graphs

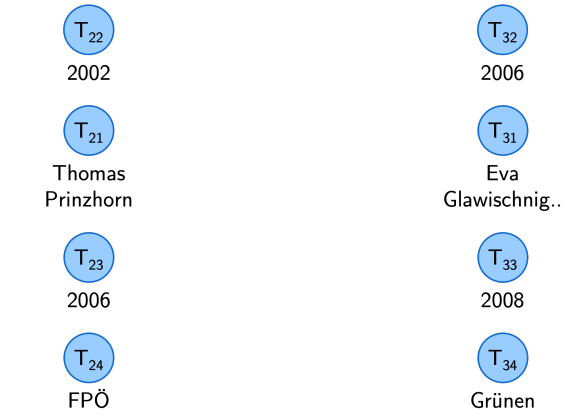
To create a candidate graph, we first create a data graph of all possible relationships between table cells and table context. Then we summarize the data graph to obtain relationships between columns and table context.

Algorithm 4: CONSTRUCT DATA GRAPH

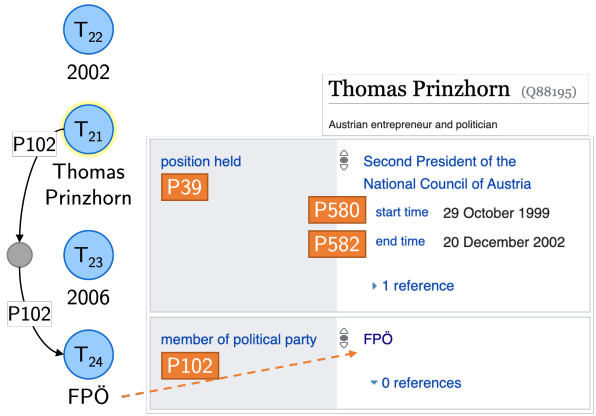
Output: the data graph G_d
Input: Input table T, input context $\mathcal{C} = \{v_1, ..., v_n\}$, max hop maxHop

```
1  G_d ← empty graph
2  add cells in the table (c_{i,j} ∈ T.cells) as nodes to G_d
3  add values in the context (v_i ∈ C) as nodes to G_d
4  for n_i ← G_d.nodes do
5      for n_j ← G_d.nodes do
6          if CanLink (n_i, n_j) then
7              for e_i ← n_i.linkedEntities do
8                  for e_j ← n_j.linkedEntities do
9                      add FindEnt2EntPaths (e_i, e_j, maxHop) to G_d
10                 add FindEnt2LiteralPaths (e_i, n_j) to G_d
11 InferAdditionalLinks (G_d)
12 return G_d
```

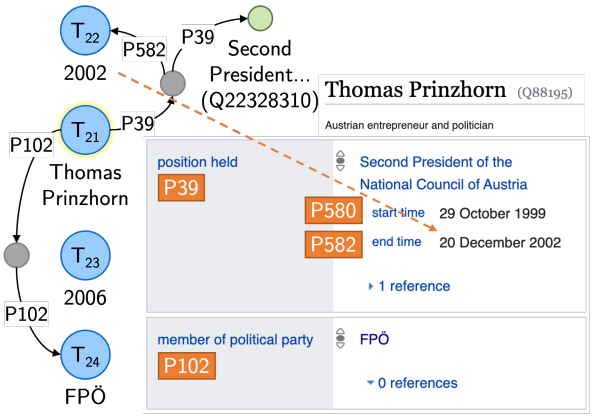
Constructing Data Graphs Algorithm 4 outlines the process of building a data graph, which is also illustrated in Figure 3.2. To begin, we add cells of T and items in $\mathcal{C}$ as nodes to an empty graph G_d (lines 1 - 3, Figure 3.2a). Then we find paths in Wikidata that connect two nodes in G_d using two functions `FindEnt2EntPaths` and `FindEnt2LiteralPaths` (lines 4 - 10). The former function simply returns paths between two entities in Wikidata (Figure 3.2b). The latter function returns paths from an entity to a literal (Figure 3.2c). Since literals in the table are not always matched exactly with the corresponding values in the KG, we "fuzzy" match literals depending on their types. For example, numbers are matched if they are within a 5% range; dates are matched if they are equal or their years are equal (when the literals only have years); strings are matched if



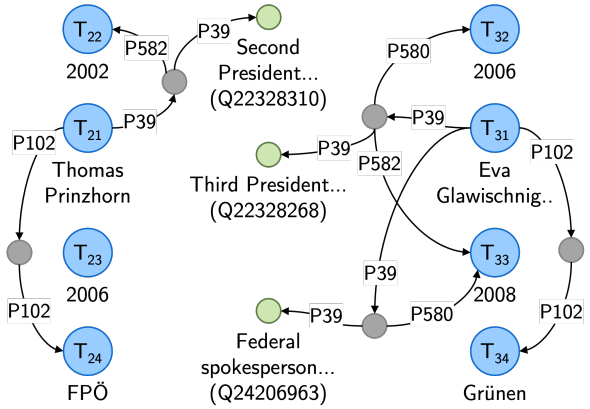
(a) The data graph is first initialized with table cells.



(b) Added the relationship P102 between cells T_{21} and T_{24} as found in Wikidata.



(c) Cell T_{22} is matched to the value of the qualifier P582 of the statement P39. To make the statement valid, the value of the statement, which is the entity `Second President`, is also added to the graph.



(d) An excerpt of the final data graph.

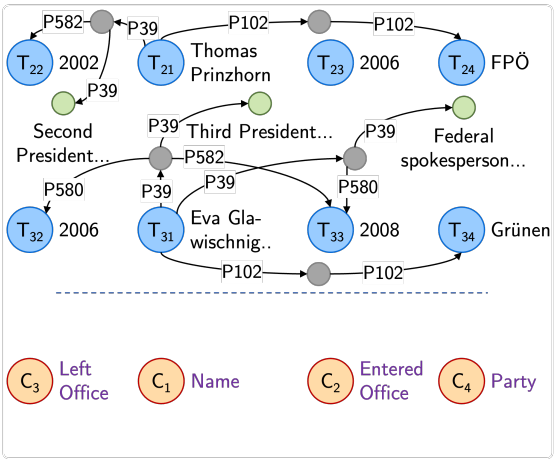
Figure 3.2: Illustrations of steps in building the data graph of the table in Figure 3.1 using Algorithm 4. Edges are displayed without their full labels for readability. Grey, blue, and green nodes are statements, table cells, and entities, respectively.

they are the same. The function `FindEnt2EntPaths` has an extra parameter `maxHop` controlling the length of discovered paths. If the maximum hop is two, a path can reach a target literal or entity via an intermediate entity in Wikidata. If the target entity or literal is found in qualifiers of statements, we also return extra paths from the source entity to the statements' values in order to comply with the Wikidata data model (Figure 3.2c). Note that we only need to find paths between pairs of nodes that can be linked (line 6). Two nodes are linkable when they are cells of the same record (i.e., in the same row), or one node is a cell and the other node is a value in the context.

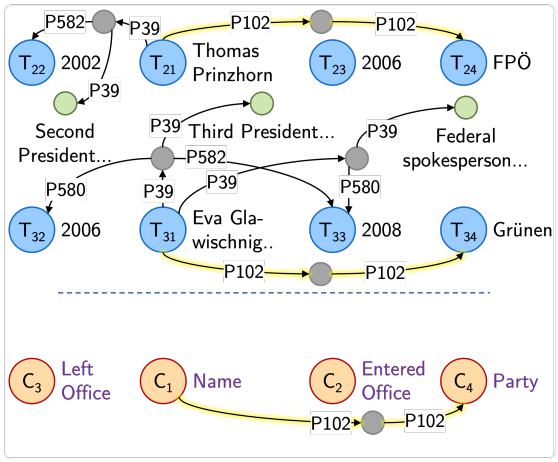
The discovered paths are then added to G_d such that the original identifiers of Wikidata statements and entities are preserved (lines 9 and 10). This allows paths of n-ary relationships to be connected automatically as they share the same Wikidata statements. Figure 3.2d shows an excerpt of the data graph of the table in the motivating example after this step.

Finally, we infer additional edges in G_d based on logical rules (inverse, sub-property, and transitive rules) specified in the Wikidata ontology (line 13). The intuition is that the final graph G_d after inference should be the same as if we run inference on KG, then build the data graph G_d. For example, if we have a link `capital (P36)` from node `France` to node `Paris`, we add a link `capital of (P1376)` from `Paris` to `France` because `P1376` is an inverse property of `P36`. Similarly, we add an edge `location (P276)` between nodes u_d and v_d in the graph G_d if there is an edge `located in admin... (P131)` between them as `P131` is a sub-property of `P276`. Note that for the sub-property rule, we only consider parent properties found in G_d to avoid generating unnecessary top-level properties such as `rdf:Property` and to keep a reasonable size of G_d.

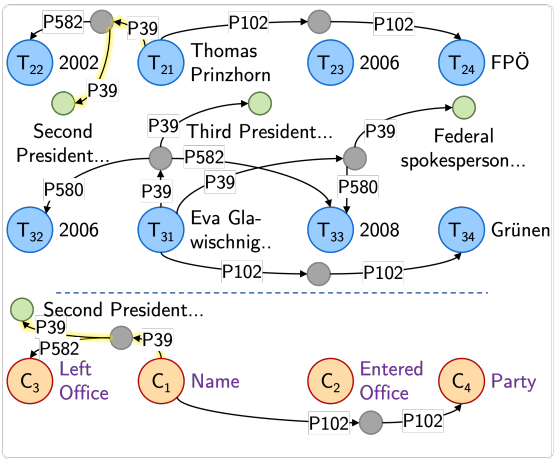
Constructing Candidate Graphs With the data graph G_d built from the previous step, we summarize it to create a super graph of plausible semantic descriptions. The step is similar to a reversion of the process that generates an RDF graph from the semantic description of the table.



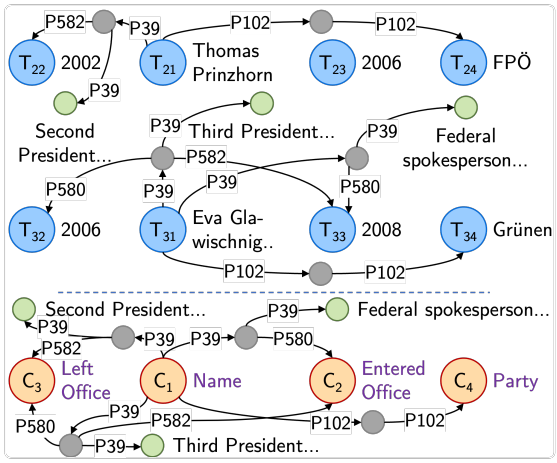
(a) The candidate graph (under the dashed line) is first initialized with table columns.



(b) The relationships P102 between cells of columns 1 and 2 are grouped and added to the candidate graph.



(c) The relationship P39 between cells in column 1 and the entity `Second President` is added to the candidate graph. Then, the qualifier P582 to column 3 is added to the graph.



(d) An excerpt of the final candidate graph (under the dashed line)

Figure 3.3: Illustration of steps in building the candidate graph of the table in Figure 3.1 using Algorithm 5. The graphs on top of the dashed lines are the data graphs; the graphs under them are the candidate graphs. Edges are displayed without their full labels for readability. Grey, blue, orange, and green nodes are statements, table cells, table columns, and entities, respectively.

Algorithm 5: CONSTRUCT CANDIDATE GRAPH

Input: A data graph G_d, Input table T
Output: the candidate graph G_s

1 $G_s \leftarrow$ empty graph
2 add columns in the table T as nodes to G_s
3 add literal or entity nodes in G_d as nodes to G_s, keeping their original id
4 **for** $u_d \in G_d$.nodes **do**
5 **for** $v_d \in G_d$.nodes **do**
6 **if** v_d is value of a property e of u_d **then**
7 $\text{stmt}_d \leftarrow$ statement of property e linking u_d and v_d
8 $u_s, v_s \leftarrow$ corresponding node of u_d, v_d in G_s, respectively
9 $\text{stmt}_s \leftarrow$ statement of property e linking u_s and v_s
10 **if** stmt_s does not exist **then**
11 add stmt_s to G_s and link u_s to v_s: $u_s \xrightarrow{e} \text{stmt}_s \xrightarrow{e} v_s$
12 **for** qualifier q of stmt_d **do**
13 $t_d \leftarrow$ target node of q of stmt_d in G_d
14 $t_s \leftarrow$ corresponding node of t_d in G_s
15 **if** the qualifier $\text{stmt}_s \xrightarrow{q} t_s$ does not exist **then**
16 add qualifier $\text{stmt}_s \xrightarrow{q} t_s$ to G_s
17 **return** G_s

Specifically, relationships of cells of two columns or of cells of a column and a context value are consolidated if they are of the same property. For example, in Figure 3.3b, relationships `P102` between cells of columns 1 and 2 are grouped to be represented as one edge `P102` between these columns in the graph.

This idea is implemented in Algorithm 5 and is illustrated in Figure 3.3. It starts by adding columns in the tables as nodes to G_s (lines 1 and 2). Then we add literal or entities nodes in G_d to G_s keeping their original id (line 3). Next, for each pair of nodes (u_d and v_d) in G_d in which v_d is the value of a property e of u_d specified by statement node $\texttt{stmt}_d$ (lines 4 - 7), we find the corresponding nodes of u_d and v_d in G_s called u_s and v_s, respectively (line 8). If u_d is a cell node, then its corresponding node u_s in G_s will be the column node; otherwise, u_s will be the node of

the same id. Next, we add a new statement node $\texttt{stmt}_s$ of the relationship e between u_s and v_s if it does not exist (lines 9 - 11). After that, we add new qualifiers to $\texttt{stmt}_s$ based on qualifiers of $\texttt{stmt}_d$ with a similar manner (lines 12 - 16).

3.2.2 Predicting Correct Relationships using PSL

The candidate graph obtained from the previous step can contain spurious relationships. To identify correct relationships, we use PSL [2]. PSL is a machine learning framework for developing probabilistic graphical models using first-order logic. A PSL model consists of predicates and rules (logic or arithmetic) constructed from those predicates. An example of a PSL rule is:

$$w : \textsc{CloseFriend}(A, B) \wedge \textsc{CloseFriend}(B, C) \Rightarrow \textsc{Friend}(A, C)$$

where w is weight of the rule, CloseFriend, Friend are predicates, A, B, C are variables. A ground predicate can have a value between 0 and 1 (e.g., $\textsc{CloseFriend}(\text{"James"}, \text{"Mary"}) = 0.8$), and is observed if its value is known. The example rule can be read as "if A and B are close friends and B and C are close friends, then A and C should be friends". If a rule in PSL does not have weight, it will be considered as a hard constraint. Given a set of observations (ground predicates whose values are known), PSL substitutes (grounds) predicates in the rules with the observations and performs convex optimization to infer the values of the unobserved predicates.

3.2.2.1 PSL model

Table 3.1 shows the list of main predicates in our PSL model. $\textsc{CorrectRel}(U, V, S, P)$ and $\textsc{CorrectType}(U, T)$ are the target predicates that we want PSL to infer the values. With these predicates, we design the following PSL rules.

$$\neg\textsc{CorrectRel}(U, V, S, P) \tag{3.1}$$

$$\neg\textsc{CorrectType}(U, T) \tag{3.2}$$

$$\textsc{Rel}(U, V, S, P) \wedge \textsc{PosRelFeat}_i(U, V, S, P) \Rightarrow \textsc{CorrectRel}(U, V, S, P) \tag{3.3}$$

$$\textsc{Rel}(U, V, S, P) \wedge \textsc{NegRelFeat}_i(U, V, S, P) \Rightarrow \neg\textsc{CorrectRel}(U, V, S, P) \tag{3.4}$$

$$\textsc{Type}(U, T) \wedge \textsc{PosTypeFeat}_i(U, T) \Rightarrow \textsc{CorrectType}(U, T) \tag{3.5}$$

$$\textsc{Rel}(U, V, S_1, P_1) \wedge \textsc{Rel}(U, V, S_2, P_2) \wedge (S_1 != S_2) \wedge \textsc{SubProp}(P_1, P_2) \Rightarrow \neg\textsc{CorrectRel}(U, V, S_2, P_2) \tag{3.6}$$

$$\textsc{Type}(U, T) \wedge \textsc{TypeDistance}(U, T) \Rightarrow \neg\textsc{CorrectType}(U, T) \tag{3.7}$$

$$\textsc{CorrectRel}(U, V, S, P) \wedge \textsc{StatementProperty}(S, P) \wedge \neg\textsc{Domain}(P, T) \Rightarrow \neg\textsc{CorrectType}(U, T) \tag{3.8}$$

$$\textsc{CorrectRel}(U, V, S, P) \wedge \neg\textsc{Range}(P, T) \Rightarrow \neg\textsc{CorrectType}(V, T) \tag{3.9}$$

$$\textsc{CorrectRel}(U, V, S, P) \wedge \textsc{DatatypeProperty}(P) \Rightarrow \neg\textsc{CorrectType}(V, T) \tag{3.10}$$

$$\textsc{CorrectRel}(U, V, S, P_1) \wedge \textsc{Rel}(V, T, S_2, P_2) \wedge \textsc{OneToMany}(V, T) \Rightarrow \neg\textsc{CorrectRel}(V, T, S_2, P_2) \tag{3.11}$$

Rules 3.1 and 3.2 are default negative priors indicating that usually there is no relationship between two nodes and a column has no type, respectively. Rules 3.3 and 3.4 state that if there is an indicator supporting or opposing the relationship (U, V, S, P), then the relationship should

Table 3.1: Predicates in the PSL model.

Predicates	Meaning
$\textsc{Rel}(U, V, S, P)$	A candidate relationship P of statement S between nodes U and V
$\textsc{CorrectRel}(U, V, S, P)$	Denoting if a relationship $\textsc{Rel}(U, V, S, P)$ is correct
$\textsc{Type}(U, T)$	A candidate type T of column N
$\textsc{CorrectType}(U, T)$	Denoting if a type $\textsc{Type}(U, T)$ is correct
$\textsc{SubProp}(P_1, P_2)$	Property P_1 is a subproperty of P_2
$\textsc{StatementProperty}(S, P)$	P is the main property of statement S
$\textsc{PosRelFeat}_i(U, V, S, P)$	Value of feature i backing the relationship $\textsc{Rel}(U, V, S, P)$
$\textsc{NegRelFeat}_i(U, V, S, P)$	Value of feature i opposing the relationship $\textsc{Rel}(U, S, V, P)$
$\textsc{PosTypeFeat}_i(U, T)$	Value of feature i backing the column type (U, T)
$\textsc{Domain}(P, T)$	Type T is a domain of property P
$\textsc{Range}(P, T)$	Type T is a range of property P
$\textsc{DatatypeProperty}(P)$	Value of property P is a literal data (e.g., numbers, strings, datetimes)
$\textsc{OneToMany}(U, V)$	A value in column U is associated with multiple values in column V
$\textsc{TypeDistance}(U, T)$	The shortest distance of type T of column U to the most fine-grained candidate types of U divided by the longest distance

be correct or incorrect, respectively. The supporting and opposing features of (U, V, S, P) are computed based on the number of rows in which we discover the relationship (U, V, S, P) (denoted as *match*(U, V, S, P)), and the number of rows in which existing data of the relationship in Wikidata is different from the data in the table (denoted as *difference*(U, V, S, P)). The two numbers are normalized in various ways: divided by the number of rows, the number of rows that have entities, or by $\sum_p$ *match*(U, V, S, P) + *difference*(U, V, S, P) resulting in different features. Similar to rule 3.3, rule 3.5 also uses features to predict if T is a correct type of column U. Currently, it uses two features: the percentages of rows that (1) have entities of type T and (2) have entities of type T or subtype of T.

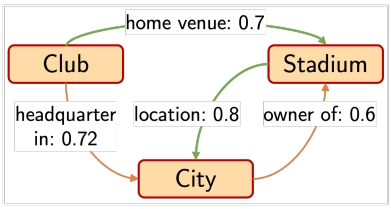
Unlike the previous rules, the remaining rules are applied to more than one relationship or type. They are used to enforce consistency of the descriptions with the Wikidata data model as well as to introduce prior knowledge of the desired semantic descriptions. Specifically, rules 3.6 and 3.7 favor fine-grained properties and types. Rules 3.8 and 3.9 ensure that the predicted type of a column should be one of the domains and ranges of the outgoing and incoming ontology property, respectively. In addition, if an ontology property is a datatype property, the target of the property cannot be instances of a class (rule 3.10). Finally, rule 3.11 prefers that properties' values of non-subject entities should have a one-to-one correspondence to the entities. The non-subject entities are defined as entities with incoming relationships from other entities in the table (i.e., not the main entities that the table is about).

We use the same weight ($w = 2$) for all rules, except that the default negative priors (rules 3.1 and 3.2) should have less weight as instructed in the PSL tutorial ($w = 1$); rules that introduce preferences should have very small weights (rules 3.6 and 3.7 have $w = 0.1$); and rules that act as constraints should have very high weights (rule 3.11 has $w = 100$).

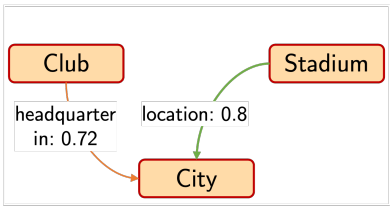
3.2.2.2 Inference and Post-processing

From G_s, we extract values of all predicates in the PSL model except the CorrectRel and CorrectType predicates. Then we run PSL inference to determine the values of the two predicates, which represent the probabilities of edges between nodes in G_s and types of columns, respectively. Values that have probabilities lower than a chosen threshold (0.5) are considered incorrect and are removed.

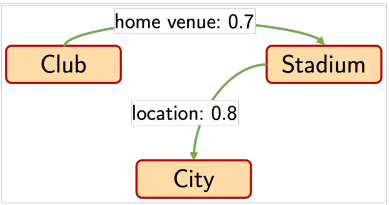
The type with the highest probability for each column is selected. However, using the same approach for selecting relationships is not ideal as it can result in multiple paths between two



(a) Pruned graph with edges having probabilities greater than or equal to a threshold.



(b) Selecting edges with the highest probabilities missed a relationship between Club and Stadium.



(c) Selecting a tree that has the highest average probabilities.

Figure 3.4: An example of relationships with their probabilities predicted by PSL. Green edges are correct while red edges are incorrect.

nodes. In many cases, a single path is sufficient to describe the relationship between two nodes. Including extra paths, which tend to be incorrect, can decrease the precision of our prediction. For example, Figure 3.4a shows four remaining relationships (after pruning ones with low probabilities) in which only two are correct. In addition, it is rare to have two or more independent main subjects, which are nodes that do not have any incoming relationships, in a table. This is demonstrated in the example in Figure 3.4b. In other words, there often exists a path between any pair of nodes in a semantic description. For these reasons, our goal is to select relationships between columns and entity/literal nodes that form a directed rooted tree with the highest average probabilities of edges. The tree may contains other nodes, hence it is a directed Steiner tree.

Algorithm 6 presents the pseudo-code of this step. Inspired by the backward search algorithm [4], the idea is identifying a root node that can reach all column nodes, then generating trees by merging paths from the root node to the column nodes. Lines 8 to 10 finds the top K paths between a pair of nodes in the graph similar to finding K shortest path. Then, for each potential root node, we merge the paths from the root node to each column node in round-robin fashion (lines 14 to 24). If the tree after merging contains statement nodes that do not have their

Algorithm 6: FIND STEINER TREE

Input: - G_{sp}: a candidate graph that its low probability edges were removed
- K_{path}: keeping maximum K paths that reach a node
- K_{tree}: keeping maximum K trees

Output: A predicted semantic description

```
if | G_sp.connectedComponents | > 1 then
    tree ← empty tree
    for component ∈ G_sp.connectedComponents do
        recursive call the algorithm with the component and merge the result into tree
    return tree
Add a pseudo root node to G_sp and edges from the pseudo root node to all nodes in G_sp
pathsBetweenNodes ← {};
for columnNode ∈ G_sp.nodes do
    for node ∈ G_sp.nodes do
        pathsBetweenNodes [columnNode, node ] ← top K_path from node to
         columnNode
roots ← list of nodes that are all visited by all column nodes and not statement nodes
bestTree ← null
for root ∈ roots do
    topKTrees ← []
    for columnNode ∈ G_sp do
        if topKTrees is empty then
            topKTrees ← create a tree for each path of pathsBetweenNodes
             [columnNode, root ]
            continue;
        nextTrees ← set()
        for path, tree ∈ pathsBetweenNodes [columnNode, root ] × topKTrees do
            newTree ← merge path into tree
            fix statement nodes in newTree that do not have their property
            add newTree to nextTrees
        topKTrees ← K_tree best trees in nextTrees
    bestTree ← best tree of topKTrees and bestTree
Remove the pseudo root node from bestTree if it exists
Add context nodes that are correct to bestTree
return bestTree
```

property (i.e., only have qualifiers) and the statement property is in the graph, we add the statement property back (line 21). Finally, to ensure that we always have at least one root node, we add a pseudo node that has edges to all remaining nodes, then remove the pseudo node after the algorithm finds the Steiner tree (lines 10 and 25). The weight of edges of the pseudo node is the total weight of edges in the graph plus 1 so that we do not use those edges unless there is no common root node.

3.3 Evaluation

3.3.1 Datasets for Semantic Modeling

Our objective is to assess the ability of our method to infer correct semantic descriptions of linked tables. There are several standard datasets for benchmarking this problem, such as T2D [50] or Limaye [38]. However, these datasets are not linked to Wikidata; they are relatively simple and do not capture the complexity of the semantic modeling problem in Wikipedia tables. Therefore, we introduce a new dataset of 250 Wikipedia tables, called 250WT[*], with their semantic descriptions built using the Wikidata ontology.

The new dataset's tables are selected from a pool of approximately 2,000,000 relational Wikipedia tables with the following procedure to ensure good coverage over multiple domains and produce high-quality unambiguous annotations. First, we filter to keep tables with at least one relationship between columns and have at least one column with at least 8 links[†]. Each table

[*] https://github.com/binh-vu/sm-datasets

[†] This requirement is to help reduce ambiguity and speed the annotation process.

Table 3.2: Details of the 250WT dataset. New data is the data that is extracted from tables but is not in Wikidata.

Average number of rows	46.34
Average number of columns	5.536
% new relationships	(21235/33336) 63.7%
% new entities	(3717/21007) 17.7%
% missing entities' type	(996/21007) 4.7%
(sampled) % new relationships (after fixing entity linking (FEL))	(1464/2241) 65.3%
(sampled) % new relationships (before FEL)	(1560/2241) 69.6%
(sampled) % incorrect relationships (after FEL)	(3/2241) 0.13%
(sampled) % new or missing type entities (after FEL)	(214/1393) 15.4%
(sampled) % new or missing type entities (before FEL)	(260/1393) 18.7%

is then assigned to a category for stratified sampling to select a maximum of 30 tables per category. The category is the most popular ontology class of the QNode's classes associated with the Wikipedia article of the table. For example, tables in Wikipedia list articles will be assigned to category `Wikimedia list article (Q13406463)`. We initially drew a sample size of 500, then two annotators annotated tables in each category one at a time (ordered by category size) until they agreed on the same semantic descriptions. However, we stopped the manual annotation process when we reached 250 tables as the cost exceeded our budget.

Table 3.2 shows the details of the 250WT dataset. If we extract data from the tables using their semantic descriptions, we obtain 33,336 new relationships and 21,007 new entities or entities' types. By comparing the extracted data with Wikidata's data, we found that 63.7% and 17.7% of relationships and entities are not in Wikidata, respectively. As the comparison is computed automatically, the new data may include data that is already in Wikidata (due to errors in entity linking) or is incorrect. Therefore, we sampled 10% (24/237) of the tables that have new data to manually check and fix the linked entities, then verified the extracted relationships. We found

that there are 46 (3.3%) incorrectly linked or not linked entities and only 3 (0.13%) incorrect relationships in the tables. This result shows that Wikipedia tables contain new knowledge and can be very useful to enhance Wikidata.

Finally, to compare with other systems that match tables to Wikidata, we also use a synthetic dataset from the final round of the SemTab 2020 Challenge [25]. This dataset contains approximately 22,000 tables generated automatically from Wikidata. This dataset also comes with a list of target columns for which we need to predict the types and a list of target columns' pairs for which we need to predict the relationships. However, there are some entity columns or columns' pairs in the tables that should be annotated but are not due to not being in the target lists. Thus, for this dataset, we follow the SemTab2020 evaluation protocol to only evaluate the predictions on the items of the two lists.

3.3.2 Evaluation Tasks

We assess the predicted semantic descriptions' quality in two different tasks: assigning an ontology class to a column (called an entity column) and predicting relationships in the table.

The first task is the Column-Type Annotation (CTA) task in the SemTab 2020 Challenge; the performance on each table is measured by approximations of precision and recall. The difference between the approximate metric and its original version is the use of a scoring function, score$(\cdot)$, to calculate the correctness of an annotation. Specifically, let $d(x)$ be the shortest distance of the predicted class to the ground truth (GT) class where $d(x)$ is 0 if the predicted class is equal to GT, is 1 if the predicted class is the parent or child of GT. Then score$(x) = 0.8^{d(x)}$ if $d(x) \leq 5$ and x

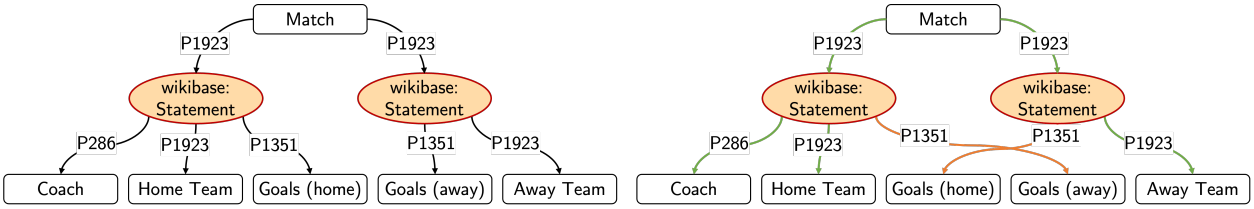


Figure 3.5: Example for CPA metrics (left is ground truth and right is prediction). Green and red edges are correct and incorrect, respectively.

is a correct annotation or an ancestor of GT; score$(x) = 0.7^{d(x)}$ if $d(x) \leq 3$ and x is a descendent of GT; otherwise, score$(x) = 0$.

The second task is slightly different from the Column-Property Annotation (CPA) task in the SemTab 2020 challenge due to n-ary relationships. As shown in Figure 3.5, despite the fact that the relationship (`P1923`, `P1351`) between `match` and `goals (home)` is the same as in the ground truth, it is not the correct relationship because it belongs to a different team. Inspired by the idea in [59], we find the best mapping between statement nodes in a predicted description to statement nodes in the ground truth description that maximizes the number of overlapping edges between them. Then we measure the approximate precision, recall, and F_1 of edges as in the CTA task. For example, in Figure 3.5, the best mapping is $\{n_3 \to n_1, n_4 \to n_2\}$ as it returns 5 overlapping edges. We have two incorrect edges: $\langle n_3, \texttt{P1351}, \texttt{goals (away)} \rangle$ and $\langle n_4, \texttt{P1351}, \texttt{goals (home)} \rangle$. Hence, the approximate precision and recall are $\frac{2}{7}$.

We report the results of the tasks as the average of 10 independent runs. For readability, we omit $\pm 0.1\%$ in our tables when the standard errors of mean (SE) are less than 0.1% unless otherwise specified.

3.3.3 Evaluation Baselines

We compare our method, GRAMS, with three state-of-the-art (SOTA) systems: MantisTable [10], BBW [54], and MTab [43] in mapping tables to Wikidata. MantisTable achieves SOTA results on several gold-standard benchmark datasets on mapping to DBpedia. BBW is among the top three winners of the SemTab 2020 challenge and finished in second place in the final round (within 0.1 - 0.2% average F1 score from the top performer). MTab is the winner of the SemTab 2020 challenge. To ensure a fair assessment, we modify the inputs of the SOTA systems to use linked relational tables (i.e., tables' cells are already linked to entities in Wikidata) instead of unlinked relational tables.

In addition, we also develop another baseline, GRAMS-ST, for comparison on the CPA task in which we replace the PSL inference with the Steiner tree algorithm (Algorithm 6). The idea of using the Steiner tree algorithm is to find a semantic description of a table such that the total weight of relationships is minimized. The weight of a relationship is defined as the inverse of the number of rows in which we discover the relationship using Wikidata's data. Hence it is similar to choosing the most popular relationship.

Our source code, baselines, and evaluated data are available on Github[‡].

3.3.4 Performance Evaluation

Table 3.3 shows that GRAMS outperforms the baseline systems on all tasks on the 250WT dataset and has similar average F_1 scores to the SOTA baseline on the SemTab2020 dataset with the differences are approximately 0.1%. On the 250WT dataset, GRAMS exceeds the SOTA baseline by

[‡] https://github.com/usc-isi-i2/GRAMS/releases/tag/swe22

Table 3.3: Performance comparison with baseline systems on CPA and CTA tasks. AP, AR, and AF_1 are average approximate precision, recall, and F_1, respectively. MantisTable†, BBW†, and MTab† are given correct tables' subject column. *SE $= \pm 0.2\%$

Dataset	Method	CPA			CTA		
		AP	AR	AF_1	AP	AR	AF_1
250WT	MantisTable	53.0%	44.7%	48.5%	**88.4%**	30.4%	45.2%
	MantisTable†	55.0%	57.1%	56.0%	87.7%*	34.9%	49.9%
	BBW	75.6%	11.8%	20.5%	83.1%	23.0%	36.0%
	BBW†	63.6%	58.9%	61.2%	73.9%	76.1%	75.0%
	MTab	83.9%	48.5%	61.5%	77.0%	77.0%	77.0%
	MTab†	84.8%*	54.6%	66.4%	77.5%	77.5%	77.5%
	GRAMS-ST	58.6%	**70.9%**	64.2%	-	-	-
	GRAMS	**88.1%**	63.9%	**74.1%**	81.3%	**82.4%**	**81.8%**
SemTab2020	MantisTable	98.3%	97.5%	97.9%	95.9%	79.0%	86.6%
	BBW	99.2%	**99.2%**	99.2%	96.7%	96.7%	96.7%
	MTab	**99.5%**	99.1%	**99.3%**	**97.1%**	**97.1%**	**97.1%**
	GRAMS-ST	99.2%	99.1%	99.2%	-	-	-
	GRAMS	99.3%	99.1%	99.2%	97.0%	97.0%	97.0%

12.6% and 4.8% F_1 scores on the CPA and CTA tasks, respectively. GRAMS also surpasses our alternative version (GRAMS-ST) by 9.9% F_1 score on the CPA task. This demonstrates that the PSL model, which takes into account both the likelihood of the candidate predictions and contradicting evidence, is more robust than a model based on selecting the most frequent relationship.

The superior performance of GRAMS over the SOTA baselines on the 250WT dataset comes from two main sources. First, MantisTable, BBW, and MTab need to identify a subject column from which we find relationships to other columns. Hence their performance is significantly affected if the results of the subject column detection step are incorrect. If we give MantisTable and BBW the correct subject columns, we observe an increase in their F_1 scores on the CPA task by 7.5% and 40.7%, and on the CTA task by 4.7% and 39.0%, respectively. Second, tables in the 250WT dataset are more challenging. Many tables are denormalized tables, which include

more than one type of entities, require n-ary relationships, or context values to model their data. Thanks to the candidate graph and the PSL model, GRAMS outperforms the SOTA baselines by 7.7% F_1 score (CPA) and 4.3% F_1 score (CTA) even when the baselines receive the correct subject columns of the tables.

However, GRAMS and the baselines do not perform well on tables that have little overlap with Wikidata data. For example, GRAMS cannot predict the correct semantic description of a table in 250WT dataset about athletics participating in a Summer Universiade and their ranking because Wikidata does not have data on the Universiade's participation. This explains the significant gap between the F_1 score on the SemTab2020 dataset and the 250WT dataset.

3.3.5 Table Subject Detection

In this experiment, we evaluate the role of context in GRAMS in helping to recognize the correct subject of the tables. We consider that the subject of a table is what the table is primarily about. It is often expressed in the semantic description via ontology classes (e.g., a table about districts of Uzbekistan uses the class `districts of Uzbekistan` for a column `districts`) or relationships with literal values. For example, the subject of the example table is politicians who are `Third President of National Council... (Q22328268)` (expressed via property `position held (P39)`).

We report the performance of GRAMS on the 250WT dataset because it annotates the table subjects. Overall, 76 tables have subjects and GRAMS can predict the subjects of 18 (23.68%). Among the 18 tables, 17 are predicted correctly and one table is not predicted exactly as the annotated ground truth. The table is about locations of an event that GRAMS uses the property

`destination point (P1444)` instead of the property `location (P276)` used in the gold annotation. We consider that it is still semantically correct because P1444 is a direct subproperty of P276 and is actually used to express the locations of that event in Wikidata. Without using context, GRAMS does not predict the subjects; hence, the recall is 0. We do not include the baselines in this experiment because they do not predict the table subjects.

Analyzing the remaining tables that GRAMS cannot predict the subjects, we find that 48 tables (63.16%) do not have any overlapping data with their subjects in Wikidata and 10 tables (13.16%) have some overlapping data. Among the 48 tables that have no overlapping data, 17 (22.37%) of them have relationships with the subjects that can be derived based on the time range. For example, the table subject is about politicians serving in the 24th Parliament of British Columbia, and some politicians are members of Legislative Assembly of British Columbia from 1952 to 1972. However, such inference can be noisy and may require specific knowledge of the ontology; hence, we leave this for future work. For the 10 tables having some overlapping data but GRAMS cannot predict, the main reason is that besides the overlapping data, Wikidata also has data of the subjects that are different than the data in the tables. For example, in a table about movies that an actor/actress is a cast member of, a few movies list them in their casting list but other movies in the table do not; they only list their lead actors/actresses. Because of that, our PSL model considers the matched relationship coincident and discard the correct subject.

3.3.6 Feature Analysis and Post-processing Evaluation

To understand the impact of the PSL rules on the result, we perform ablation analysis by removing each of three groups of rules: fine-grained properties and classes (rules 3.6, 3.7), relationship-type

Table 3.4: Ablation analysis of the groups of rules in GRAMS. Each cell value is a difference in a score when dropping one group of rules.

	CPA			CTA		
	AP	AR	AF_1	AP	AR	AF_1
GRAMS	**88.1%**	**63.9%**	**74.1%**	**81.3%**	82.4%	81.8%
− Fine-grained Properties & Classes (FG)	86.4%	62.7%	72.7%	80.7%	81.8%	81.3%
− Functional Dependency (FD)	85.6%	63.9%	73.2%	80.9%	82.1%	81.5%
− Relationship-Type Agreement (RT)	88.0%	63.8%	74.0%	81.3%	**82.4%**	**81.9%**
− FG − RT − FD	84.1%	63.0%	72.0%	80.8%	82.1%	81.5%

agreement (rules 3.8, 3.9, 3.10), and functional dependency (rule 3.11). The results are reported in Table 3.4. The most important group is fine-grained properties/classes. Removing this group decreases the performance of both CPA and CTA tasks. This is expected because Wikidata ontology is deep and utilizes specific properties and classes to express different nuances. For example, at the time of writing, an entity such as United States (Q30) is associated with 8 different types. The function dependency group mainly affects the precision of relationship predictions as removing it decreases the CPA precision by 2.5% while the CPA recall is roughly the same. The CTA scores slightly drop due to less accurate relationship predictions. Finally, the relationship-type agreement group has no apparent effect on the average scores on both tasks on 250WT. Because the automatic entity linking step is highly accurate, the domains and ranges of the discovered candidate relationships mostly agree with the corresponding candidate types. Thus this explains the little effect of these rules. We conjecture that this group will be more important when the quality of the entity linking decreases.

Table 3.5 shows the performance of our post-processing algorithm (Steiner tree) in Section 3.2.2.2 versus two baselines: selecting relationships with the highest scores between two nodes

(named Pairwise Selection) and finding the minimum arborescence (named Minimum Arborescence). The first baseline has a slightly higher CPA recall than all methods since it may predict more than one relationship per node. However, because of that, it has the lowest CPA precision. The second baseline has better performance than the first baseline since it enforces relationships to form a tree similar to our Steiner tree method. However, it has smaller scores than the Steiner tree method. The reason is that entity/literal nodes are optional in the Steiner tree method and only used if they are needed to describe the relationships (e.g., connecting two nodes).

Table 3.5: Performance of GRAMS with respected to different post-processing algorithms on the 250WT dataset.

	CPA			CTA		
	AP	AR	AF_1	AP	AR	AF_1
Pairwise Selection	79.8%	**65.5%**	71.9%	80.5%	**82.6%**	81.6%
Minimum Arborescence	86.9%	63.4%	73.3%	81.3%	82.3%	81.8%
Steiner Tree	**88.1%**	63.9%	**74.1%**	**81.3%**	82.4%	**81.8%**

3.3.7 Running Time Evaluation

In this experiment, we evaluate GRAMS's running time against the baseline systems: MantisTable, BBW, and MTab. The experiment is run on a single laptop with M1 Pro and 32GB RAM. We use a local key-value database to store Wikidata to avoid network overhead. The results are reported in Figure 3.6. The baseline systems have much shorter running times on the SemTab2020 dataset than on the 250WT dataset. The reason is that on the SemTab2020 dataset, the targets of CPA (i.e., pairs of columns) and CTA (i.e., columns) tasks are provided, while on the 250WT dataset, the targets are unknown and the baselines have to find them. On the other hand, GRAMS shows similar running times on both datasets because our implementation does not rely

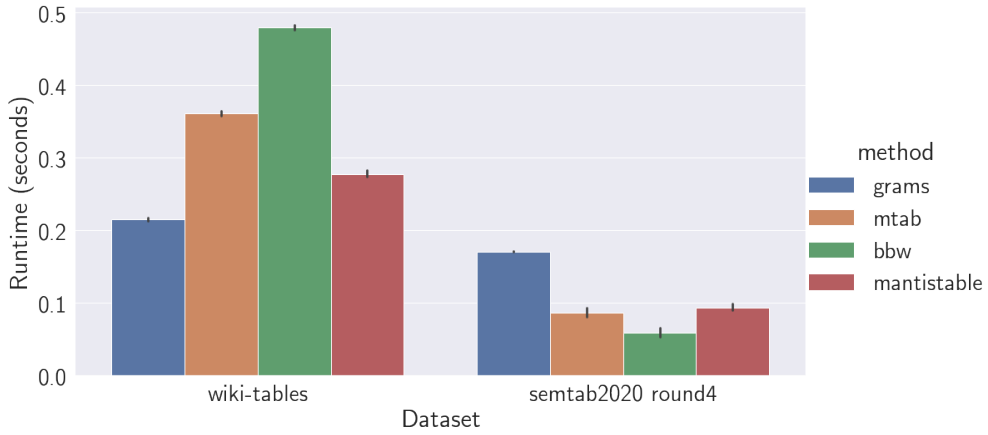


Figure 3.6: Average running time (seconds) per table of GRAMS in comparison with the baseline systems.

on the given CPA and CTA targets. Although our system is more complex than the baselines and is not well optimized, it has a reasonable running time of approximately 0.2 seconds per table, and is faster than MTab and BBW on the 250WT dataset.

3.4 Summary

In this chapter, we present a novel graph-based approach for building semantic descriptions of Wikipedia tables using Wikidata. Our approach constructs a candidate graph of possible relationships between columns in the table and uses collective inference to identify correct relationships and types. The evaluation shows that by using graphs to represent relationships and collective inference, our approach is robust compared to state-of-the-art systems and can handle tables with complex descriptions.

Chapter 4

Creating Semantic Descriptions of Unlinked Tables with Overlapping Data

In the previous chapter, we described our graph-based approach to creating semantic descriptions of linked tables. In this chapter, we extend the problem setting to unlinked tables that have overlapping data with a knowledge graph. Existing approaches on mapping tables to KGs [28, 37, 38, 41, 43, 50, 71] also exploit the overlapping data by linking table cells to entities in KGs, then matching the entities' property values with other table cells to find the semantic description. However, they tend to perform poorly on tables with lots of ambiguity such as having similar candidate entities or few matches. Due to the limited amount of training data, it is challenging for these methods to learn the optimal parameters to combine the entity linking and data matching results to resolve the ambiguity.

To address this limitation, we present a novel approach, named GRAMS+, that leverages the semantic descriptions created automatically from linked tables using GRAMS to train two neural network (NN) models: one for entity linking and one for disambiguating data matching results.

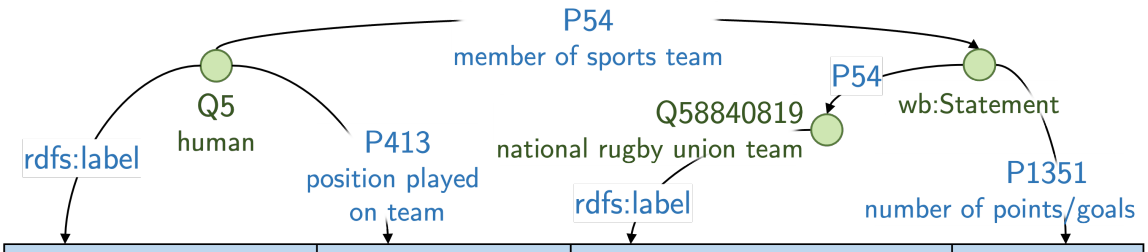
In our empirical evaluation, training the NN models with distant supervision provides approximately 5% improvement in column type prediction and 4.5% improvement in column relationship prediction in F1 scores over the state-of-the-art.

Our contribution is a novel method of using distant supervision for the semantic modeling problem. Our solution provides an algorithm to create the semantic description of a given table that is more robust to noise and ambiguity in the table data than the previous state-of-the-art. We also demonstrate that the automatically labeled dataset, although it may be noisy, can be useful to help train machine learning models to understand the semantics of tables.

4.1 Motivating Example

In this section, we provide a real example where existing systems have difficulties. Then we show how it can be addressed using other information in the table. Figure 4.1 shows an excerpt of a table of players of national rugby teams to Wikidata's ontology with its semantic description on top. Similar to the previous chapter, we interchangeably refer to Wikidata's classes and properties either by their labels and ids (e.g., `Q5 human`) or just by their ids (e.g., `Q5`).

The first step in methods that exploit a KG as background knowledge is to detect linkable cells and then retrieve candidate entities associated with the cells. For example, `Dan Carter` can link to `Dan Carter (politician)` or `Dan Carter (rugby player)`. Similarly, `New Zealand` can link to `New Zealand (country)`, `New Zealand national football team`, or `New Zealand national rugby team`. Then, each property's value of the candidate entities of a cell is matched with other cells in the table to identify potential relationships between the two cells. For example, the value of the property `P27 country of citizenship` of `Dan Carter (rugby player)`



Player	Position	Team	Points
Dan Carter	Fly-half	New Zealand	1598
Ronan O'Gara	Fly-half	Ireland	1083
Percy Montgomery	Fullback	South Africa	893
Chris Paterson	Fullback	Scotland	809

Figure 4.1: Excerpt of a table of players of national rugby teams with its semantic description on top. Green circle nodes are ontology classes; blue cells on the first table row represent columns. The node `wb:Statement`, named statement node, is used to represent a ternary relationship between the players, their teams, and the number of points scored for the teams.

is `New Zealand (country)` suggesting that `P27` can be the relationship between the columns `Player` and `Team`.

The ambiguity arises as the correct entity of the cell `New Zealand` is the rugby team, but the candidate entity `New Zealand (country)` has a label that matches exactly with the cell's value. In addition, property `P27 country of citizenship` is found in more rows than the correct property `P54` making it even more uncertain. Fortunately, the surrounding context such as the column header `Team` tells us that entities in this column should not be countries. Also, we find more members of rugby teams in the table than members of football teams. Thus, rugby teams should have higher likelihood, which are the correct entities. With a large training set, we can learn to combine all these signals to predict the correct semantic description for this table.

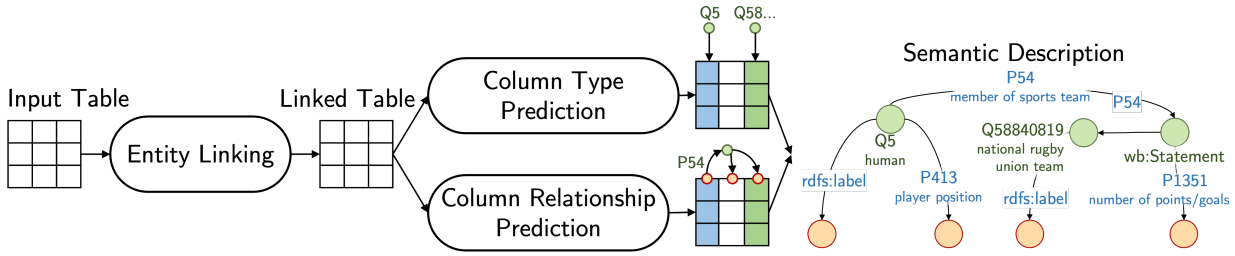


Figure 4.2: Overall approach

4.2 Approach

In this section, we explain our approach for the semantic modeling task for unlinked tables. In this approach, the target ontology to map the tables to is the ontology of the knowledge graph. Figure 4.2 shows the overall process. It starts by finding KG entities that are mentioned in a table (entity linking). Then, we use a neural network (NN) to compute the scores of candidate entities of each table cell. The NN model is trained with distant supervision. Using the discovered candidates and their scores, we perform two tasks: predict column types (CTA) and column relationships (CPA). The results of CTA and CPA are combined to get the final semantic description. In our experiment, finding column relationships is much more difficult than finding the column types. To combine all potential signals for the CPA task, we use another NN model to estimate the likelihood of possible relationships. This model is also trained using distant supervision.

4.2.1 Creating Labeled Dataset from Wikipedia Tables

To automatically annotate Wikipedia tables, we leverage the hyperlinks inside the tables to find corresponding Wikidata entities and predict columns' relationships based on the linked entities. However, it's a non-trivial task to label all Wikipedia tables with high accuracy. There are several challenges such as detecting table layout and orientation, identifying location of headers and

content, or context-inconsistent hyperlinks. An example of context-inconsistent hyperlinks is a column named `city`, but the links in the column are to `airport`.

For simplicity, we focus on relational tables for which it is relatively easy to automatically generate labeled data. We define the following conditions to identify an easy-to-label table: (1) has a minimum of 10 rows, (2) has a maximum of one hyperlink per cell, (3) has a column with at least 70% of its cells containing hyperlinks, and (4) more than 80% of the links have corresponding entities in the KG. To remove context-inconsistent hyperlinks, we first automatically assign a type to each column based on the most common type of its entities. Then, we employ a blocklist to remove all links in a column if the column header is incompatible with the predicted column types. We create the blocklist by manually reviewing normalized column headers* that appear in multiple predicted types and labeling the incompatible types. The number of headers to review is approximately 230 headers.

To generate labeled data for entity linking, we use the existing hyperlinks as positive examples. The negative examples are other candidate entities retrieved using the method described in Section 4.2.2. To generate labeled data for column relationship prediction, we first apply our method described in Section 4.2.4 to create a graph containing candidate relationships, then remove the relationships that occur in less than 50% of the rows or have more than 10% of the rows with different data than in KGs. After that, the remaining relationships are grouped by the source and target columns or literal values; only the relationships with the highest matching frequencies are used as the ground truth for the column relationship prediction. The generated candidate relationships that are not in the ground truth are considered as negative examples. The numbers used in the aforementioned steps are chosen heuristically.

*We normalize a header by masking numbers, removing special characters, etc.

4.2.2 Entity Linking in Tables

The first step in our method is to link table cells to entities in the KG. Following typical entity linking (EL) systems, our EL approach consists of three main steps: (1) detect the entity columns, which are the cells that will be linked; (2) retrieve candidate entities for each cell; and (3) compute the candidates' likelihood.

To detect entity columns, we use the same heuristic as in [43]. Specifically, if the majority of the cells of a column are classified as text (e.g., using Spacy and regex), the column will be linked. This heuristic yields a high recall but low precision (about 0.75 to 0.8 on the evaluation datasets). Next, we generate candidate entities for each cell in the entity columns by combining search results of several approaches: public KG search API (e.g., Wikidata API), keyword search using ElasticSearch, and fuzzy matching of entity names [18]. Finally, we use a neural network that takes a mention (i.e., table cell), the surrounding context (e.g., column header), and a candidate entity, then, predicts the likelihood of the candidate entity.

Our candidate entity scoring model is a two-hidden-layer perceptron with RELU activations. It is trained using the auto-label dataset described in Section 4.2.1 with the following groups of features.

Surface Features consist of four different metrics to measure the similarities between a mention and candidate names: Levenshtein, Jaro-Winkler, Monge Elkan, and Generic Jaccard.

Entity-context Similarity Features capture the coherence between a candidate and the surrounding context of a mention. We have two context similarity features: the weighted dot product of the embeddings of the column header and the candidate's description, and the number of cells matched with the candidate's property divided by a large constant representing the

maximum number of columns in a table (e.g., 20) for rescaling. The embeddings are computed from a Sentence Transformer model [49][†], and the weights of embedding dimensions are learnable parameters.

Entity Prior Features bias the predictions toward popular entities. Currently, we use the normalized log page rank of a candidate as the prior feature. The normalized log page rank of an entity e is calculated as follows:

$$\frac{\log(\textbf{pagerank}(e)) - \min_{e' \in \mathcal{E}} \log(\textbf{pagerank}(e'))}{\max_{e' \in \mathcal{E}} \log(\textbf{pagerank}(e')) - \min_{e' \in \mathcal{E}} \log(\textbf{pagerank}(e'))}$$

where $\mathcal{E}$ is the set of entities in KG, pagerank(e) is the pagerank of an entity e.

As real-world tables sometimes do not have headers, we train another version of our entity linking model on the same auto-label dataset in which the headers have been removed. At the testing time, if a column header is empty, we use the model trained without headers. Otherwise, we use the model trained with headers. We find that this strategy works better than training a single model on the combined datasets.

4.2.3 Column Type Prediction

Algorithm 7 describes our column type prediction method. This greedy algorithm first selects the type with the highest score from the set of types directly found in the candidate entities (lines 1 - 3) of a column. The score of a type is computed by summing the maximum likelihood of the candidate entities of the type for each cell (line 2) and divided by the number of rows (line 3). Next, the algorithm iteratively refines the prediction by replacing it with an ancestor type within

[†]We use the pretrained all-mpnet-base-v2 model.

Algorithm 7: COLUMN TYPE PREDICTION

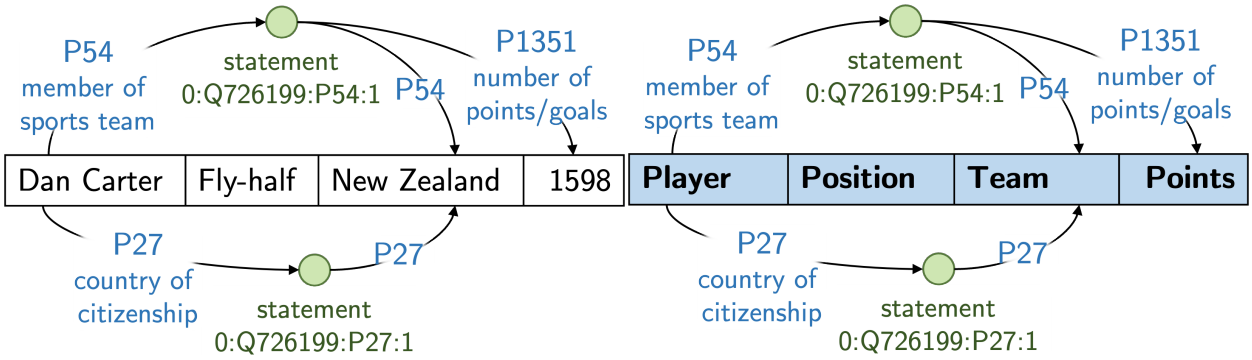
Input: A target column C, candidate entities of each cell in C, a threshold δ, and a maximum searching distance max_distance

Output: predicted column type and its score

```
1  di_types ← direct types of candidate entities in C
2  type2cells ← mapping from a type in di_types to the maximum likelihood of candidate
     entities of the type (no inheritance) for each cell in C
3  best_type, best_avg_likelihood ← the type with the highest average likelihood from
     type2cells
4  for d ← 1..max_distance do
5      extend_types ← di_types and ancestors of di_types within d-hop
6      extend_type2cells ← mapping from a type in extend_types to the maximum
         likelihood of candidate entities of the type (with inheritance) for each cell in C
7      new_type, new_avg_likelihood ← the type with the highest average likelihood from
         extend_type2cells
8      if new_avg_likelihood ≥ best_avg_likelihood + δ then
9          best_type ← new_type
10         best_avg_likelihood ← new_avg_likelihood
11 return best_type, best_avg_likelihood
```

d distance of the directed types if the score difference is larger than a specific threshold δ (lines 4 - 10). d is increased by one after each iteration up to max_distance, which is a parameter of the algorithm. The threshold ($\delta = 0.1$) and maximum distance (max_distance = 2) are chosen empirically.

To illustrate the algorithm, we use an example of a column named `university` containing ten cells. The annotated type of the column is `Q3918 university`. We have eight cells linked to public universities and two cells linked to private universities. First, the algorithm finds two candidate types `Q875538 public university` and `Q902104 private university` from the entities in the column (line 1), which the likelihoods are 0.8 and 0.2, respectively (line 2 - 3). Thus, it assigns the best type to be `Q875538` (line 3). Then, it considers ancestors of the discovered types within 1-hop and discovers `Q3918 university` (line 5). The likelihood of the new type is 1.0 (line 6 - 7). Since it is greater than $0.8 + \delta$, the best type is reassigned to `Q3918`. In the



(a) The data graph after processing the first row of the table

(b) Excerpt of the candidate graph built from the data graph

Figure 4.3: Excerpt of a data graph and a candidate graph containing relationships discovered in the table in Figure 4.1

second iteration, it searches ancestors within 2-hop distance, and finds `Q2385804 educational institution`, of which likelihood is also 1.0. However, the algorithm will not reassign the best type as the score difference is now less than δ. The iteration ends and the algorithm returns `Q3918`, which is the correct type.

4.2.4 Column Relationship Prediction

Our column relationship prediction consists of three steps. First, we generate a candidate graph containing potential relationships between columns. Then, we use a classifier to predict the likelihood of each link in the graph. Lastly, we employ the updated Steiner Tree algorithm presented in Section 3.2.2.2 to select the links that maximize the average likelihood as our final prediction.

The candidate graph is created using a modified version of the algorithm introduced in Section 3.2.1. The general idea of the algorithm is to build a data graph, containing relationships between cells in a table. Then, we group the relationships between the cells from the same pairs of columns to obtain the relationships at the column level. Different from the original version, our modified algorithm treats each row of the table independently. Therefore, when building

the data graph, we can process the table in parallel while not having hubs with many connected edges.

To discover the relationships between cells, for each row, we iterate through each candidate entity of a cell and match the property value of the candidate with other cells in the same row. When we find a match, we add a statement node and link from the cell to the matched cell through the statement node. The statement node is uniquely identified by the row number as well as the original statement id in Wikidata. This enables n-ary relationships to be constructed automatically during the matching process. For example, in the first row of the table in Figure 4.1, assuming that we have the following candidate entity `Q726199` for `Dan Carter`, and $\langle$`Q664`, `Q55801`$\rangle$ for New Zealand. We discover the following relationships: (`Q726199` $\xrightarrow{\texttt{P54 members of sports teams}}$ `statement:0:Q726199:P54:1` $\xrightarrow{\texttt{P54}}$ `Q55801`), (`Q726199` $\xrightarrow{\texttt{P54}}$ `statement:0:Q726199:P54:1` $\xrightarrow{\texttt{P1351}}$ `1598`) , and (`Q726199` $\xrightarrow{\texttt{P27}}$ `statement:0:Q726199:P27:1` $\xrightarrow{\texttt{P27 country of citizenship}}$ `Q664`). The data graph after processing the first row is shown in Figure 4.3a.

From the data graph, the candidate graph is created by grouping links of nodes between the same pair of columns. The grouping is performed in two phases: links representing statement properties are grouped before links representing statement qualifiers. For example, in Figure 4.3a, we will first group links ($\xrightarrow{\texttt{P54}}$ `statement:*:*:P54` $\xrightarrow{\texttt{P54}}$) between cells of columns `Player` and `Team`. Then, we add the qualifiers (`statement:*:*:P54` $\xrightarrow{\texttt{P1351}}$) to the grouped statement node. An excerpt of the candidate graph is shown in Figure 4.3b.

The classifier employed to predict the likelihood of links is also a two-hidden-layer perceptron with RELU activations. It is trained on the auto-label dataset (Section 4.2.1) to take features extracted from a single link and classify whether it is correct. The features include the relative frequency of discovering the link from top K entities, the average link likelihood, the relative

frequency of finding contradicting information between the table data and KG data, and whether there is a many-to-many relationship between the source and target of the link.

We use the trained classifier to predict the probability of outgoing links of statement nodes in the candidate graph. The incoming link of a statement node will have the same score as the outgoing link representing the statement property. Finally, we use the Steiner Tree algorithm to select a subtree that maximizes the average likelihood of the links as our column relationship prediction. For example, assuming the scores of ⟨`Player`, `Team`, `P27`⟩, ⟨`Player`, `Team`, `P54`⟩, and ⟨`Player`, `Point`, `P1351`⟩ are 0.7, 0.85, and 0.6, respectively. Our Steiner Tree will select a subtree containing the last two links (`P54` and `P1351`) as it has the highest average likelihood.

4.3 Evaluation

4.3.1 Experiment Setup

Data We evaluate our approach, GRAMS+, on three datasets for mapping tables to Wikidata: 250WT [66], HardTables (2022 SemTab challenge, round 1) [23], and WikidataTables (2023 SemTab challenge, round 1) [24]. The 250WT dataset created in the previous chapter contains 250 tables sampled from Wikipedia and is manually annotated. We update this dataset to remove all HTML markup (e.g., remove hyperlinks to the correct entities in Wikidata) because this information is generally not available to real-world tables. The HardTables and WikidataTables datasets are synthetic datasets containing approximately 3650 and 9920 tables, respectively. We choose the datasets mapping to Wikidata ontology instead of other datasets because Wikidata has a huge ontology with millions of classes and thousands of properties. Compared to other ontologies, Wikidata ontology is designed to represent n-ary relationships (using statements and qualifiers),

which can be found in real-world tables. Together, this captures the challenges and complexities of the semantic modeling task in the wild.

The most recent SemTab challenge (2023) introduces tFood, a new synthetic dataset generated from Wikidata's entities. However, we cannot use it because the columns' values containing entity names are anonymized using random characters. Since the names are replaced, this dataset focuses on a different aspect of the problem, which is finding the anonymous entities. This is not the focus of this research and we leave it for future work.

Our training dataset is created by randomly sampling five thousand automatically labeled tables from Section 4.2.1. We exclude any Wikipedia articles containing tables from the 250WT dataset from the training set to avoid data contamination. Also, we use Wikidata dumps on 2023-06-19 and Wikipedia dumps on 2023-06-20.

Evaluation Metrics Similar to the previous chapter, we assess the quality of the predictions in two different tasks: column type annotation (CTA) and column relationship annotation (CPA). The two tasks are evaluated using the approximate precision, recall, and F1 scores defined by the SemTab challenge [1]. The difference between the approximate metrics and their exact version is the partial credits they give when a system predicts a sub/parent class/property of the correct one. In particular, let $d(x)$ be the shortest distance of the predicted item (class or property) to the ground truth (GT) item. $d(x)$ is 0, 1 if the predicted item is equal to GT, or parent or child of GT, respectively. score$(x) = 0.8^{d(x)}$ if $d(x) \leq 5$ and x is a correct annotation or an ancestor of GT; score$(x) = 0.7^{d(x)}$ if $d(x) \leq 3$ and x is a descendent of GT; otherwise, score$(x) = 0$. For readability, we sometimes refer to the approximate precision, recall, or F_1 score by just precision, recall, or F_1 score.

Baselines We compare our approach with winners of the SemTab challenges: MTab [43], KGCODE-TAB [37] and DAGOBAH [28], which are the current state-of-the-art (SOTA) systems on this task. We create another baseline from GRAMS [66], the system that has the SOTA result on the 250WT dataset. Because GRAMS assumes that it is given the correct entities, we run GRAMS with the top 1 candidate entities. We cannot evaluate the best systems of the 2023 SemTab Challenge (TorchicTab [13] and SemTex [26]) as their source code is not available. However, SemTex reports its performance on the HardTables dataset, and we directly use the reported numbers for comparison.

To ensure a fair comparison, the baselines are modified to receive the same candidate entities as our method (100 candidate entities per cell). We cannot modify DAGOBAH because it is only available as an API at the time of submission. However, for DAGOBAH's CTA output, we notice that instead of predicting if a column contains people, they attempt to predict the occupation as the column type (e.g., politicians instead of humans). To give DAGOBAH credit for their prediction, if the column type in the ground truth is `Q5 human`, we will map the predicted occupation (subclass of `Q215627 person`) to the correct class.

Modeling Training Our neural network models for entity linking and column relationship prediction are trained using Adam [30] with a learning rate of $1e{-}4$ for 50 epochs.

4.3.2 Overall Performance

Table 4.1 shows the performance of our method versus the baseline on the 250WT dataset. Our method outperforms MTab by 12.32% and 11.78% and DAGOBAH by 4.57% and 4.95% in macro-average approximate F_1 scores in the CPA and CTA tasks, respectively. Our method achieves

Table 4.1: Performance comparison with the baselines on CPA and CTA tasks on the 250WT dataset. AP, AR, and AF_1 are macro-average approximate precision, recall, and F_1, respectively

Method	CPA			CTA		
	AP	AR	AF_1	AP	AR	AF_1
GRAMS+	**82.79**%	62.06%	**66.41**%	**80.54**%	78.26%	**78.94**%
MTab	73.28%	50.72%	54.09%	64.61%	71.9%	67.16%
DAGOBAH	76.42%	60.92%	61.84%	71.47%	**78.95**%	73.99%
GRAMS	75.86%	42.27%	44.71%	66.33%	77.38%	70.45%
KGCode-Tab	28.25%	**70.01**%	36.52%	42.28%	58.03%	47.82%

considerably higher F_1 scores than GRAMS and KGCode-Tab in both tasks. In the CPA task, we generate candidate relationships using the method in [66]. This method does not rely on detecting subject columns of tables and can also detect n-ary relationships. Therefore, it helps us achieve higher recall. In addition, we also have greater precision because the neural network for scoring relationships is better at distinguishing incorrect relationships, thanks to the distant supervised training. The improvement in the CTA task is mainly due to better candidate entity scoring on ambiguous tables. For example, for the column `Team` in the motivating example, our entity linking model correctly ranks entities of national rugby teams among the top 2 (top 1 is national football teams). MTab, however, prefers entities of type country as it does not use any signal from the surrounding context (e.g., header). DAGOBAH also selects countries for this example.

Table 4.2 and 4.3 show the performance of our method versus the baselines on the HardTables and WikidataTables datasets. Most methods achieve high performance. The reason is that this synthetic dataset is much less noisy and less ambiguous than the 250WT dataset. The mentions and entity labels are often identical or differ by one or two characters, whereas in 250WT, the differences can be more significant (e.g., `Ireland` versus `Ireland National Rugby Union Team` or `CB` versus `center back`). DAGOBAH, however, has a much lower CPA recall than the other

Table 4.2: Performance comparison on the HardTables dataset

Method	CPA			CTA		
	AP	AR	AF_1	AP	AR	AF_1
GRAMS+	98.92%	91.20%	91.71%	94.36%	90.58%	90.63%
MTab	98.78%	**94.06**%	**94.10**%	**94.87**%	**91.5**%	**91.53**%
DAGOBAH	**99.2**%	36.40%	37.10%	92.9%	90.0%	90.1%
GRAMS	95.64%	83.72%	84.57%	80.37%	87.64%	80.54%
KGCode-Tab	88.33%	86.79%	81.22%	74.86%	80.42%	73.64%

methods as their API turns off the feature to predict relationships to numeric columns. Also, we cannot evaluate DAGOBAH on the WikidataTables dataset as they discontinue their API. While our method's results exceed DAGOBAH, they are lower than MTab by 2.38% and 0.9% in F_1 scores in CPA and CTA tasks on the HardTables dataset and by 2.99% and 0.9% in F_1 scores in CPA and CTA tasks on the WikidataTables dataset, respectively.

Analyzing the results of our approach, we find that most errors in the CPA task are due to predicting relationships of unannotated pairs of columns. For example, we need to predict the relationship `P361 part of` between columns 0 and 1, but our method predicts the inverse relationship `P527 has part` between columns 1 and 0; hence, it does not get a score. To verify our finding, we run another experiment in which our method and MTab receive the correct target columns that the system should predict. For example, an input table has three columns but the systems are instructed to predict only relationships between columns 0 and 1. In this experiment, on the HardTables dataset, both methods' new CPA average macro F_1 scores are 94.48%, and

Table 4.3: Performance comparison on the WikidataTables dataset

Method	CPA			CTA		
	AP	AR	AF_1	AP	AR	AF_1
GRAMS+	94.38%	91.10%	89.25%	96.08%	**94.00%**	94.06%
MTab	**94.7%**	**95.91%**	**92.24%**	**97.14%**	93.51%	**94.1%**

Table 4.4: Performance comparison on the HardTables dataset when the target columns for CPA and CTA are provided. m-AP, m-AR, and m-AF_1 are micro-average approximate precision, recall, and F_1, respectively

Method	CPA			CTA		
	m-P	m-R	m-F_1	m-AP	m-AR	m-AF_1
GRAMS+	98.03%	92.89%	95.39%	93.98%	91.18%	92.56%
MTab	98.77%	93.16%	95.88%	95.31%	92.53%	93.90%
DAGOBAH [28]	**99%**	**97.8%**	**98.4%**	**97.5%**	**97.5%**	**97.5%**
KGCode-Tab	98.19%	86.85%	92.17%	86.73%	81.40%	83.98%
KGCode-Tab [37]	94.4%	94.0%	94.2%	91.8%	89.43%	90.6%
SemTex [26]	-	-	97.05%	-	-	93.85%

on the WikidataTables, our method has a better CPA F_1 score by 0.51%. Therefore, our neural network for ranking relationships is not the main reason for the lower performance than the baseline on this dataset.

For our CTA results, we see that many failed tables have only 3 or 4 rows. Either the entities' types in the rows are changed in our Wikidata snapshot (they no longer belong to the target classes in the ground truth), or we incorrectly rank some entities in the top 1. Thus, the score differences between the target classes and incorrect classes are very similar. Because our CTA method only relies on the candidate entity scores, it does not use our CPA task's output to help disambiguate them. This is a limitation of our CTA approach and we leave this for future work. Note that this issue tends not to happen for bigger tables. The reason is that our CTA method

Table 4.5: Performance comparison on the WikidataTables dataset when the target columns for CPA and CTA are provided.

Method	CPA			CTA		
	m-P	m-R	m-F_1	m-AP	m-AR	m-AF_1
GRAMS+	98.39%	96.03%	97.20%	95.79%	93.94%	94.86%
MTab	99.35%	96.90%	**98.11%**	97.18%	95.47%	**96.32%**
SemTex [26]	-	-	96.4%	-	-	93.4%

indirectly utilizes the collective signal from other columns since our entity linking method prefers entities with overlapping data with the table.

As we cannot obtain the source code of baselines such as DAGOBAH and SemTex, we cannot ensure the same experiment setup for our method and the baselines (e.g., having the same candidate entities). Therefore, we evaluate our method and MTab in the same setting as the SemTab challenge to compare with their results. Results of this experiment are reported in Table 4.4 and Table 4.5. For DAGOBAH and SemTex, we directly use the numbers from their papers. On the HardTables, DAGOBAH outperforms the other systems in the CPA and CTA tasks by at least 2.52% and 3.6% (F_1 scores), respectively. One possible reason is the changes in the correct entities' types and properties in our Wikidata snapshot, as discussed above. The difference between the CPA performance of MTab and our method in this experiment is not statistically clear[‡]. On the WikidataTables, MTab's performance surpasses all systems by at least 0.91% and 1.46% (F_1 scores) in CPA and CTA tasks, respectively. However, our method also outperforms SemTex, the system that has the highest performance among the SemTab2023 participants by 0.8% and 1.46% (F_1 scores) in CPA and CTA tasks, respectively.

4.3.3 Impact of Entity Linking on the Performance

To understand the impact of the entity linking step, we evaluate our method on two different entity ranking methods: (1) using our entity linking likelihood score and (2) using MTab's candidate retrieval score. In addition, we experiment with different numbers of candidate entities per cell.

[‡]The p-value of the sign test [16, 70] on the accuracies of the two systems is 0.086

Table 4.6: The performance of our method on the 250WT dataset with respect to two different candidate ranking methods

Method	CPA			CTA		
	AP	AR	AF_1	AP	AR	AF_1
Our Entity Linking Score	82.79%	62.06%	66.41%	80.54%	78.26%	78.94%
Retrieval Score	80.74%	61.00%	64.41%	73.60%	71.70%	72.22%

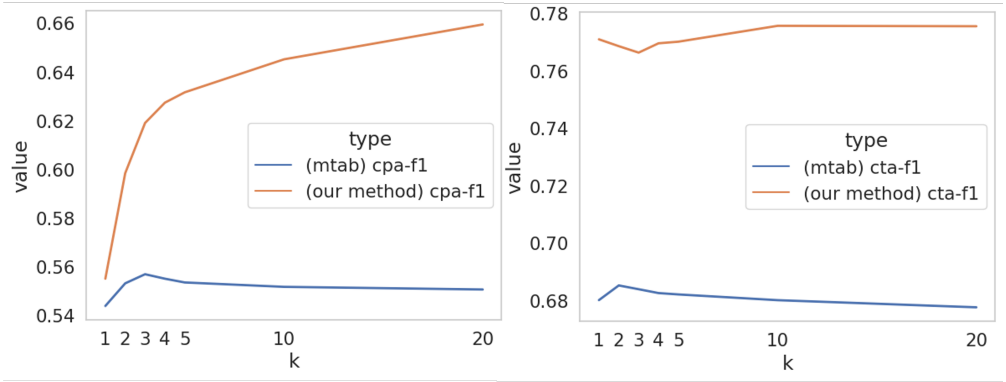


Figure 4.4: Performances of our system and MTab with different numbers of candidate entities (x-axis) on the 250WT dataset. The CPA F_1 scores are shown in the left figure and the CTA F_1 scores are shown in the right

Table 4.6 shows the performance of our system with the two aforementioned ranking approaches. The CTA F1 score drops significantly by 6.72%, while the CPA F1 score slightly decreases by 2%. The results indicate that the candidate entity scores play a critical role in the CTA task. In the CPA task, our model can leverage other collective signals in the table to make more accurate predictions and, thus, is less dependent on the candidate scores.

Figure 4.4 shows our system's and MTab's performance with different numbers of candidate entities (K) per cell. In general, our CPA F_1 score increases as K increases, while MTab's CPA F_1 score goes up to the highest value at K = 3 (55.7%) and then gradually declines. We also see a similar weaker trend in CTA scores. One of the reasons is that the input noise increases as the number of candidate entities increases. This demonstrates that our method is more robust to noise.

Table 4.7: The performance of our method on the 250WT dataset with and without table headers

GRAMS+	CPA			CTA		
	AP	AR	AF_1	AP	AR	AF_1
With metadata	82.79%	62.06%	66.41%	80.54%	78.26%	78.94%
Without metadata	77.64%	61.29%	63.75%	77.67%	77.18%	76.89%

4.3.4 Impact of Table Metadata on the Performance

To understand how well our method can leverage the table metadata (column headers) to overcome ambiguous examples, we perform an ablation study by removing the table metadata from the entity-context similarity features described in Section 4.2.2. Table 4.7 shows that our approach can utilize the table metadata to overcome ambiguous cases; hence, our F_1 scores increase by 2.66% and 2.05% in CPA and CTA tasks, respectively.

4.4 Summary

We presented a novel distant supervision approach for learning semantic descriptions of tables. Using the hyperlinks in Wikipedia tables, we generate labeled examples to train two neural networks to predict the likelihood of candidate entities and column relationships. Then, we use the two models to predict column types and relationships to obtain the semantic descriptions of tables. The empirical evaluation shows that our approach outperforms state-of-the-art methods on a large set of real-world tables. Moreover, it is more robust to noise from the entity linking step and can leverage the table metadata to overcome ambiguous examples.

Chapter 5

Creating Semantic Descriptions of Unlinked Tables without Overlapping Data

In the previous two chapters, we explain our approaches for tables with overlapping data with knowledge graphs (KGs). However, it is well known that KGs are incomplete and there are domains where KGs have very little data. When we do not have overlapping data and previously labeled tables, there are no examples to teach a model for the semantic modeling task. Thus, we need to rely on textual descriptions or instructions in a target ontology and the information in a table to find appropriate ontology classes and properties for each table column. For example, in Figure 5.1, the class *associate football position* is a more suitable type for a column *pos.* than other classes *baseball position* or *location* based on the name, description, and a few examples of instances of the class. However, learning this semantic textual similarity is more difficult and would require lots of training data compared to existing classification or value-matching approaches. However, significant progress has been made in the natural language processing field in recent years; language models are now extremely capable of capturing semantic textual similarity and memorizing knowledge in the real world. Given the large number of automatically

labeled tables we created in Chapter 4, pre-trained language models can be explored and adapted to help solve the semantic modeling problem.

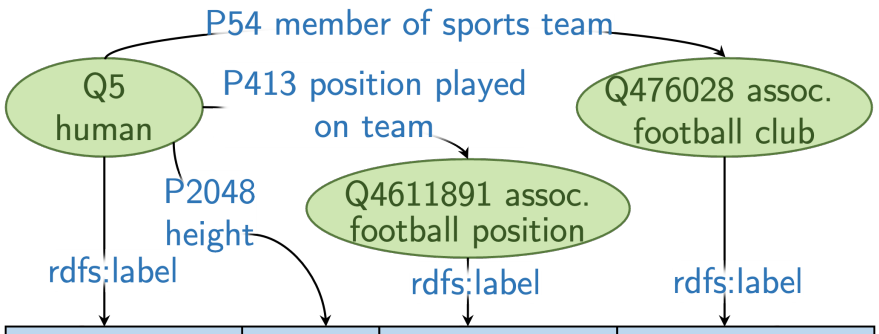
In this chapter, we present GRAMS++, a novel approach for tables without overlapping data with KGs. GRAMS++ is trained with distant supervision to estimate a score reflecting the semantic relatedness between a table column and an ontology class or property. For each column in the table, GRAMS++ predicts top K classes and properties. Then, the candidate classes and properties are combined to create the final semantic description. Our empirical evaluation shows that our approach significantly outperforms strong baselines. Our contribution is a novel method for the semantic modeling problem in domains where we have limited background knowledge or labeled tables, which have not been addressed by the previous methods.

5.1 Motivating Example

In this section, we provide a real-world example to explain how information in the table and ontology can help create the semantic description of a table of soccer players in Figure 5.1.

Scenario 1 (using table data): we assume that each class and property in the ontology comes with a set of example values. For the column *position*, we find that its values such as midfielder and goalkeeper are very similar to examples of the correct class *Q4611891 association football position* and are different from examples of the class *P39 position held*.

Scenario 2 (using table metadata): for the column *player*, we do not find any column values in the set of examples of the correct class *Q5 human*. However, the column's header steers us toward people's names instead of other classes (e.g., places).



player	height	position	team
Tom Aldred	1.88	Defender	Brisbane Roar
Shae Cahill	1.71	Midfielder	Brisbane Roar
Macklin Freke	1.93	Goalkeeper	Brisbane Roar

Figure 5.1: Excerpt of a table of association football players with its semantic description on top. Green circle nodes are ontology classes; blue cells on the first table row represent columns.

5.2 Approach

Our approach consists of two main steps. The first step is to predict candidate classes and properties to describe columns in a table. Next, we construct a candidate graph containing potential relationships in the table, then use a Steiner Tree algorithm to select relationships to create the final semantic description.

5.2.1 Column Concept Prediction

In this step, we find candidate classes and properties of a table column. For conciseness, in this section, we refer to an ontology class or ontology property as a concept. For example, the column `team` in Figure 5.1 has the top 3 concepts followed by their scores: `Q17156793 American football team: 0.21`, `Q476028 association football club: 0.22`, and `P54 member of`

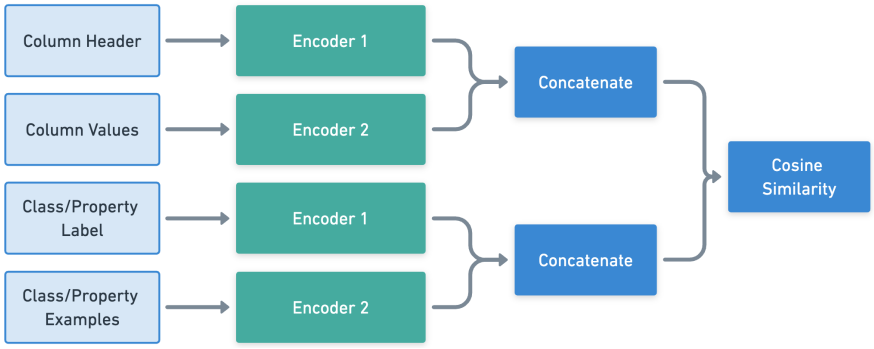


Figure 5.2: Architecture of our neural network model to estimate a similarity score between a column and a concept

`sports team`: `0.19`. If a column is an entity column, the property, if correct, will be the incoming relationship to the class associated with the column as shown in Figure 5.1.

Note that our goal is slightly different from the semantic labeling task mentioned in Chapter 2. The semantic labeling task predicts semantic types of a column, where a semantic type is a pair of a class and a property while our goal is to predict a single class or property. Semantic labeling is more complex and may not be ideal for two reasons. First, we observe that a column often only contains information about the corresponding property. For example, a column `capacity` can store the maximum capacity of stadiums, parks, restaurants, or vehicles. The correct class, `stadium`, for this column is implicit, and we need a holistic view of a table to identify it. This makes learning more challenging. Moreover, the different combinations of the same property with different classes would overwhelm the top K answers. The second reason is that predicting semantic types may leave out significant information that could be used in later processing steps. For example, the correct semantic type of a column `author` in Wikidata is ⟨`Q5 human`, `rdfs:label`⟩, which fails to capture that the column contains authors (represented by the *P50 author* property).

To find the candidate concepts of a column, we learn to estimate the similarity score between the column and a concept based on the textual description and example values of the concept. This approach has two assumptions. First, we assume that we can retrieve some example values or instances of the target ontology classes and properties. Usually, the examples can be retrieved automatically without manual input by querying knowledge graphs. If we cannot retrieve the examples, our method can still be used but the performance may be reduced. Second, since we compute the similarity score for every concept in the ontology, we assume that most of the classes and properties in the target ontology are actually used to describe the data. This assumption does not apply to Wikidata because it has millions of classes and thousands of properties. Hence, when evaluating our method on datasets using Wikidata ontology, we only keep classes and properties that are used in the ground truth.

Figure 5.2 shows the overall architecture of our neural network (NN) model for this task. First, it encodes a column name, column values, the name of a concept, and examples of the concept using encoders. Our encoders are comprised of a pre-trained transformer-based model* and a linear fully-connected layer. For data that are a list, such as column values or semantic label examples, we concatenate the data to create a sentence before feeding it to our encoders.

Our model is trained with multi-task learning [5] using a triplet loss [52]:

$$\mathcal{L}(\mathbf{x}, \mathbf{c}^+, \mathbf{c}^-) = \max(0, \text{dis}(\mathbf{x}, \mathbf{c}^+) - \text{dis}(\mathbf{x}, \mathbf{c}^-) + \epsilon)$$

where dis is the distance function between a column and a concept $\mathbf{c}$, $\mathbf{c}^+$ and $\mathbf{c}^-$ are the correct and incorrect concepts of the column, respectively, and the margin parameter ϵ is configured as

*We use BAAI/bge-m3 [6] in our experiment

the minimum offset between distances of similar vs dissimilar pairs. In our experiment, we use $\text{dis}(a, b) = 1 - \text{cosine\_similarity}(a, b)$. We create three related tasks: predicting column concepts using only table metadata, using only table values, and using both table metadata and values. Solving three tasks simultaneously encourages our model to learn a better representation and, thus, is less prone to overfitting. We train our model on the automatically labeled dataset created in Section 4.2.1.

5.2.2 Semantic Description Prediction

To predict the semantic description of a table, we perform two sub-tasks: column type prediction and column relationship prediction. For column type prediction, we first identify entity columns in the table using the entity column recognition algorithm described in Section 4.2.2. Next, for each entity column, we use the ontology class that has the highest score returned from our previous step as the class for the column.

For column relationship prediction, we create a candidate graph containing possible relationships in a table. Then, we employ the Steiner Tree algorithm to select relationships that maximize the average likelihood as our final prediction.

Constructing Candidate Graphs Algorithm 8 shows steps to create a candidate graph. It uses tuples of meta edges $\langle$ source class, property, qualifier, target class, frequency$\rangle$ as guidelines to discover relationships between classes and columns (lines 3 - 17). If the target ontology is a KG ontology, we can create the meta edges automatically by grouping and counting properties. Otherwise, we can mine Linked Open Data [60] or extract from the domain and range constraints in the ontology definitions. The algorithm has two main steps. In the first step (lines 3- 12), for

Algorithm 8: CONSTRUCT CANDIDATE GRAPH

Input: the input table T,
columns' concepts: $\mathbb{C} = \{c_i : \{\text{class or property} : \text{score}, ...\}\}$,
indexes of entity columns,
list of meta edges: { ⟨ source class, property, qualifier, target class, frequency⟩, ... }
Output: the candidate graph G

```
1  G ← empty graph
2  add columns in the table T as nodes to G
3  class2nodes ← a mapping from each ontology class to associated columns based on ℂ
   // add relationships between classes and columns
4  for column_index, col_classes, col_props ← ℂ do
5      for label, score ← col_props do
6          candidate_edges ← meta edges used label
7          for candidate_edge ← candidate_edges do
8              if domains and ranges of candidate_edge do not satisfy then
9                  continue
10             for node ← class2nodes do
11                 if node.column_index ≠ column_index then
12                     add candidate_edge between node and column_index to G
   // add relationships between classes
13 for source_class, source_nodes ← class2nodes do
14     for target_class, target_nodes ← class2nodes do
15         candidate_edges ← meta edges between source_class and target_class
16         for candidate_edge ← candidate_edges do
17             add candidate_edge between pairs of source_nodes and target_nodes to G
18 return G
```

each predicted property of a column, we use the meta edges to determine its source class and target class (if the property is an object property, otherwise the target class is none). Then, we create new edges to connect each pair of columns containing predictions of the source class and target class. In the second step (lines 12- 17), for each pair of source and target classes, we find the columns containing the corresponding predictions, then create new edges between the columns containing properties that can connect the two classes together.

Finding Steiner Trees To identify correct relationships from the candidate graph, we use our updated Steiner Tree algorithm described in Chapter 3. The updated algorithm can work on graphs with n-ary relationships. Let

- $e = \langle u, v, p \rangle$ be an edge representing the relationship p between nodes u and v,
- f(v, c) be the predicted likelihood of the concept c for column v,
- sou_f(e) and tar_f(e) be the predicted likelihood of source and target class connected by e,
- $g(e)$ be the relative frequency of the property p being used to connect the source and target classes of e. The relative frequency is computed from the input meta edges.

We calculate the likelihood of e for different cases as follows. If e is a data property, then score$(e) = f(v, p) * \text{sou_f}(e)$. If e is an object property, then score$(e) = \text{sou_f}(e) * \text{tar_f}(e) * (f(v, p) * \alpha + g(e) * \beta)$, where α and β are hyper-parameters to balance the predicted likelihood by our model and the property popularity mined from the knowledge graph. Finally, if v is a statement node, then score$(e = \langle u, v, p \rangle) = \max_{t,p'}(\text{score}(e' = \langle v, t, p' \rangle))$.

5.3 Evaluation

Evaluation Data and Metrics We evaluate our approach on the 250WT and T2Dv2 [50] datasets. We use the automatically labeled Wikipedia tables generated in Section 4.2.1 to train our model. The training dataset and 250WT use Wikidata ontology while T2Dv2 uses DBpedia ontology. Table 5.1 shows statistics of these datasets. For the 250WT dataset, approximately 58% of classes and 66% of properties do not appear in the training dataset. We evaluate the performance of our

method in two tasks: column type annotation (CTA) and column relationship annotation (CPA) using metrics presented in the previous chapter.

Table 5.1: Statistics of datasets used in the evaluation

	train dataset	250WT	T2Dv2
#tables	18261	250	224
#classes	326	155	36
#properties	104	112	103

Baselines As existing semantic modeling methods cannot handle tables that have no overlapping with KGs yet, we develop two new baselines to verify the effectiveness of our method. The first baseline, named DSL-SM, uses DSL [46] to generate semantic types of columns of a table, then uses a frequent pattern mining method proposed by Taheriyan et. al. [60] to combine the semantic types and create the semantic description of the table. The second baseline uses large language models that are trained and fine-tuned to follow textual instructions and to answer given questions. Specifically, we give LLMs the table and the ontology classes and properties as context, then ask the LLMs a series of questions to determine columns' types [†] and columns' relationships[‡]. In this evaluation, we use two LLM open-source models and create two versions of the second baseline: Llama2 [61] and OLMo [20].

Model Training Our neural network model for column concept prediction is trained using Adam [30] with a learning rate of $5e-5$ for 30 epochs. We also use a maximum of 150 examples per target ontology class and property.

Table 5.2: Performance comparison on the 250WT dataset. AP, AR, and AF_1 are macro-average approximate precision, recall, and F_1, respectively

Method	CPA			CTA		
	AP	AR	AF_1	AP	AR	AF_1
DSL-SM	19.53%	22.03%	19.71%	27.76%	27.97%	27.37%
Llama2-70B	18.86%	47.84%	26.54%	30.44%	32.11%	31.00%
Llama2-7B	06.22%	17.17%	08.98%	27.61%	28.14%	27.73%
OLMo	26.70%	22.00%	15.79%	20.78%	10.09%	09.82%
GRAMS++	**60.82%**	**57.65%**	**58.50%**	**73.74%**	**74.79%**	**73.85%**

Table 5.3: Performance comparison on the T2Dv2 dataset

Method	CPA			CTA		
	AP	AR	AF_1	AP	AR	AF_1
DSL-SM	51.14%	48.57%	48.69%	80.26%	88.94%	82.95%
Llama2-70B	46.32%	**68.42%**	51.96%	**87.97%**	**93.21%**	**87.43%**
Llama2-7B	33.97%	44.22%	35.29%	67.42%	74.32%	69.58%
OLMo	64.99%	21.73%	21.79%	30.67%	25.64%	18.75%
GRAMS++	**59.04%**	59.1%	**58.66%**	83.41%	91.80%	85.99%

5.3.1 Overall Performance

Table 5.2 shows the performance of our method versus the baselines on the 250WT dataset. GRAMS++ outperforms the best-performed baseline, Llama2-70B, on this dataset by 31.96% and 42.85% in F_1 scores in the CPA and CTA tasks, respectively. The baselines do not perform well on this hard dataset due to many classes and properties. Also, the precisions of LLM baselines are much lower than their recalls because they cannot predict if two columns have a relationship or not. Our gains in performance come from two sources. First, distant supervised training helps GRAMS++ to better distinguish between candidate classes and properties. Second, finding Steiner trees enables us to eliminate incorrect predictions to have good precisions.

[†]Question's template: *What is the best class in the list above to describe the column "COLUMN"*

[‡]Question's template: *What is the best property in the list above to describe the relationship between the column "COLUMN1" and the column "COLUMN2"*

Table 5.3 reports the performance of all methods on the T2Dv2 dataset. In the CPA task, our method outperforms all baselines by at least 6.7% in F_1 score. In the CTA task, Llama2-70B has the highest F_1 score of 87.43% while our method achieves a similar score. However, Llama2-70B's performance on the 250WT dataset is much lower than on the T2Dv2 dataset by 25.42% and 56.43% in F_1 scores for CPA and CTA tasks, respectively. Llama2-7B, ten times smaller than Llama2-70B, shows a similar trait. However, it also achieves much higher scores on the T2Dv2 dataset than the other LLM model, OLMo, with similar sizes and similar performances reported on several benchmarks [20]. Although we are unable to obtain and verify the training and fine-tuning data for Llama, this indicates that Llama may have been fine-tuned on similar tasks or data before. In addition, this demonstrates the effectiveness of GRAMS++ on multiple domains despite being much smaller in size (567 million versus 70 billion parameters). DSL-SM achieves much better performance on the T2Dv2 dataset than on the 250WT dataset. The reason is that tables in the T2Dv2 dataset are mostly annotated with only one ontology class and the target DBpedia ontology is less complicated than the Wikidata ontology used in the 250WT dataset.

5.4 Summary

We presented GRAMS++, a novel method for creating semantic descriptions of tables that have no overlapping data with a knowledge graph. Using distant supervision and pre-trained language models, GRAMS++ learns to find K most similar classes and properties to table columns and create the semantic descriptions using candidate classes and properties. Our evaluation shows that GRAMS++ achieves good performance even on a new domain ontology. GRAMS++ tends to perform better on smaller ontologies and cannot handle a huge ontology of millions of classes such

as Wikidata without discarding unrelated classes and properties. This is a limitation discussed in Section 5.2 of GRAMS++ and we leave it for future work.

Chapter 6

Related Work

Semantic modeling is one of the core tasks in data integration and machine learning and has attracted much research over the years. In this section, we review relevant studies and categorize them into three groups. Figure 6.1 shows the high-level summary of the related work.

The first group is methods that solve part of the semantic modeling problem such as semantic labeling or column relationship prediction. DSL [46] predicts the semantic types of a table column by using a matching function to find and return the types of K most similar columns it has seen before. Sherlock [27] also predicts column types using a deep neural network (NN) model trained on web tables. As the NN model is trained in the classification setting, the number of types is fixed and chosen by the authors based on the table headers. ColNet [7] predicts column types from a KG ontology. Specifically, ColNet trains binary classifiers for each ontology class and combines the classification results with entity lookup voting. Taheriyan et al. [60] assume that the column types are given and predict column relationships by mining frequent patterns from Linked Open Data. Luzuriaga et al. [39] focus on generating triples of relationships between columns for Wikipedia tables. Compared to our work, methods in the first group are limited and cannot infer the full semantic descriptions of tables.

	Method		Does not Require Manually Labeled Sources	Unlinked Table	Not requiring target columns	No overlapping data with KGs	Modeling Capabilities		
							Handle Literal Columns	Handle Qualifiers	Denormalized Tables
Supervised Approach	Taheriyan et al. 2016		N	Y	Y	Y	Y	Y	Y
	Una et al. 2018		N	Y	Y	Y	Y	Y	Y
	Suhara et al. 2022 - DODUO		N (huge)	Y	N	Y	Y	N	Y
	Vu et al. 2019		N	Y	Y	Y	Y	Y	Y
Value-linked Approach	Iterative Method	Ritze et al. 2015	Y	Y	Y	N	Y	N	N
		Zhang et al. 2017	Y	Y	Y	N	Y	N	N
		SemTab systems	Y	Y	Y	N	Y	N	N
	Graphical Models	Limaye et al. 2010	Y	Y	Y	N	N	N	Y
		Mulward et al. 2013	Y	Y	Y	N	N	N	Y
		GRAMS	Y	N	Y	N	Y	Y	Y
		GRAMS+	Y	Y	Y	N	Y	Y	Y
GRAMS++			Y	Y	Y	Y	Y	Y	Y

Figure 6.1: High-level color-coded summary of semantic modeling approaches. Our approaches are highlighted in yellow. Y/N stands for Yes/No. The lighter green is used to denote that the systems do not fully follow the corresponding classification. For example, most SemTab systems do not require target columns to make predictions.

The second group is supervised methods. Taheriyan et al. [59] build an integration graph that represents the space of plausible semantic descriptions for a new data source. Each link in the integration graph is associated with a weight representing the frequency of use of the link in previous descriptions. First, attributes of a new data source are associated with data nodes in the integration graph. Then, the semantic description is built by finding the Steiner Tree that connects the data nodes. Uña et al. [62] also predict semantic descriptions by finding the Steiner Tree. However, to solve the Steiner Tree problem, instead of using an approximation algorithm as in Taheriyan et al. [59], they use constraint programming, an exact algorithm, which allows it to incorporate more features as additional constraints. Our supervised approach, PGM-SM, in Chapter 2 belongs to this group. However, instead of solving the Steiner Tree problem, PGM-SM

uses a probabilistic graphical model (PGM) to exploit collective signals from a data source. The PGM allows us to express complex dependencies between the features of good and bad semantic descriptions. Therefore, we are able to obtain better performance than the two aforementioned methods.

TURL [14], DODUO [56], and TorchicTab-Classification [13] create the semantic description of a table by predicting column types and column relationships independently. TURL and DODUO train two transformer-based classifiers for the two tasks on a set of automatically labeled Wikipedia tables using the Freebase ontology. The labels are generated based on the common types and relationships of entities found in the tables. TorchicTab-Classification extends DODUO with a sub-table sampling strategy to work for large tables and is trained on SOTAB [34], a dataset constructed automatically from microdata embedded in websites. Compared to our supervised approach, the three methods need to be provided with the target columns to predict, while in our problem, this information is not given and the method figures it out. Moreover, they can only be applied on domains they are trained on (Freebase or Schema.org ontologies).

The third group is approaches exploiting existing knowledge in KG. These approaches are typically unsupervised, however, they can also benefit from some labeled annotations to fine-tune their parameters. The common methodology of these approaches is to identify KG entities in a table (Entity Linking - EL) and match the properties of entities with values in the table to find column types (CTA) and relationships between columns (CPA). Limaye et al. [38] and Mulwad et al. [41] are among the first works to solve the three tasks (EL, CTA, and CPA) jointly using probabilistic graphical models. However, they do not handle non-entity columns in the tables. Later work shifts to the iterative paradigm and expands the problem setting to include literal columns.

Ritze et al. [50] first identify a table's subject (entity) column, then find the candidate relationships between the subject column and other columns in the table. Ritze et al. iteratively update the candidate entities and candidate relationships until stability is achieved. Zhang et al. [71] refine entity linking results to be consistent with the annotated column types and the table's domain, estimated using a bag-of-words method, and then predict column relationships. The best performance systems of the SemTab challenges [1, 12, 22, 29], such as MTab [43], DAGOBAH [28], and others KGCode-Tab [37], LinkingPark [8], BBW [54], TorchicTab-Heuristic [13], and SemTex [26] also follow the iterative paradigm. Their systems improve various aspects of the pipeline, such as candidate entity retrieval and scoring functions to rank the matched results.

Our approaches, GRAMS and GRAMS+, belong to the third group. Compared to the aforementioned methods, we broaden the scope of the problem to build semantic descriptions containing n-ary relationships and implicit context values. Moreover, most of them make an assumption about the table structure: a table has only one subject column, and all relationships in the table are between the subject column and the remaining columns. This limits the ability to predict relationships between non-subject columns, which are often found in denormalized tables (e.g., two tables about books and authors are merged into one). Because our approaches do not make this assumption, not only can we detect relationships between non-subject columns but we also avoid the cascaded error from the subject column detection phase. For GRAMS, instead of using an iterative approach to solve the CTA and CPA tasks, using a Probabilistic Soft Logic (PSL) model enables us to express dependencies between columns and their relationships and solve the tasks jointly through convex optimization. Therefore, we were able to outperform previous work significantly. For GRAMS+, it uses distant supervision to automatically generate labeled data

to train machine learning models instead of relying on hand-crafted scoring functions. Thus, GRAMS+ can exceed the performance of existing work on real-world tables.

Finally, our method, GRAMS++, described in Chapter 5 is the first approach for semantic modeling that can be applied to a chosen domain ontology without training data. By training a neural network model that can assign a matching score to an ontology class or property based on the label, description, and examples of the class or property, we only need to train the model once and can reuse the model in other domains. This idea is similar to DSL [46]. However, DSL learns to measure the similarity between columns and can only predict the column types based on similar columns, lacking the capability to generate the full semantic description of a table.

Chapter 7

Discussion

In this dissertation, we presented novel approaches for the semantic modeling problem for tables with and without overlapping data with a knowledge graph. In this chapter, I summarize the contributions of my thesis. Then, I discuss the main limitations of the presented approaches and how we plan to address these limitations in future work.

7.1 Contributions

We presented comprehensive and complementary techniques for the semantic modeling problem under different settings and assumptions. Overall, our research has four main contributions.

In Chapter 2, we described PGM-SM, a novel approach for supervised semantic modeling. PGM-SM trains a probabilistic graphical model (PGM) to combine signals within the data to distinguish between good and bad semantic descriptions. Then, it uses the model to search through the space of possible descriptions to find the most probable description of a table. The evaluation shows that by exploring the relationships within the data, PGM-SM generates better semantic descriptions and is more robust to noise than the previous approaches.

In Chapter 3, we introduced GRAMS, a novel graph-based method for unsupervised semantic modeling of linked tables. GRAMS constructs a graph of plausible semantic descriptions using external knowledge from a knowledge graph (KG) and uses collective inference for relationship prediction. Moreover, GRAMS can handle tables that require contextual values and n-ary relations to accurately and fully describe the data. The experiments show that GRAMS significantly outperforms state-of-the-art methods on real-world tables.

In Chapter 4, we presented GRAMS+, a novel distant supervised approach for unlinked tables. We show how to generate labeled data from Wikipedia tables and use it to train our entity linking and column relationship prediction models. Building upon GRAMS, GRAMS+ can discover necessary contextual values and n-ary relationships to describe the data. Distant supervision helps our method learn to incorporate table metadata into entity disambiguation as well as handle the cascaded errors from the entity linking step. The evaluation shows that our method achieves much better performance than existing methods on real-world tables.

In Chapter 5, we introduced GRAMS++, a novel method for many domains that have little or no overlapping data with KGs or labeled tables. GRAMS++ leverages pre-trained language models and distant supervision to learn to compute a similarity score between a table column and a concept (an ontology class or property) based on the concept name, descriptions, and examples. As a result, GRAMS++ can be trained on one domain and applied to different domains. The evaluation shows that GRAMS++ significantly outperforms strong baselines on hard tables.

7.2 Applications

Our research offers innovative tools to facilitate data modeling and integration by creating semantic descriptions of data sources, particularly in domains with scarce labeled examples. For example, to help geologists perform mineral assessment[*] [53], our work, GRAMS++, is used to automatically create semantic descriptions of tables from resource papers and databases. Then, SAND is used to curate the predicted semantic descriptions and generate D-REPR models. With the D-REPR models, we can automatically convert data from the papers and databases to RDF and publish to a domain knowledge graph created for the mineral assessments.

Our methods can also be used at a large scale to enrich public knowledge graphs. A notable example is mapping Wikipedia tables to Wikidata as demonstrated in Chapter 3. By generating semantic descriptions of Wikipedia tables, we can add new data with provenance to Wikidata automatically and also keep up with the changes and growth in Wikipedia. Thus, it would benefit the Wikidata community.

Besides the applications to data integration, we find our research provides solutions to problems that we had not anticipated, such as exchanging data between different formats. For example, D-REPR is used in the DARPA World Modeler program[†] [19] to specify data formats of inputs and outputs of different systems such as hydrology and agriculture models. Hence enabling communication between systems across various data formats such as GeoTIFF[‡], NetCDF[§], and even custom text files.

[*] https://www.darpa.mil/program/critical-mineral-assessments-with-ai-support
[†] https://www.darpa.mil/program/world-modelers
[‡] https://www.ogc.org/standard/geotiff/
[§] https://www.unidata.ucar.edu/software/netcdf/

Thus, our research has contributed to multiple fields, not just data science. Together with the fact that our code is publicly available as open source for people to use and extend it, we expect our tools will empower communities to better exploit the vast amount of data available today.

7.3 Limitations

Our supervised approach presented in Chapter 2 uses an integration graph from Taheriyan's work [59] as the search space. As the graph is constructed from previously modeled sources and ontology, the amount of labeled sources increases as the size of the ontology increases. This poses a challenge to apply this method especially in domains where it is expensive to obtain labeled data.

Our subsequent work aims to address the above limitation. GRAMS and GRAMS+, described in Chapter 3 and Chapter 4, exploit the background knowledge in a KG. Thus, they can handle large ontologies and do not need manually labeled data. However, they cannot be applied to domains that are not covered by the KG ontology. Even if the domains are covered, the KG may have little information and that can affect the performance of the two methods.

GRAMS++ presented in Chapter 5 removes the overlapping data assumption. However, different from our previous work, GRAMS++ annotates every column in an input table. Yet the table may contain unknown columns that cannot be modeled using the target ontology. Thus, this can negatively impact the accuracy of the prediction.

Finally, our methods cannot build the semantic description of a table in which each row of the table has a different property. For example, a table about awards and nominees of films has

a "*result*" column describing whether a film won the award or not. The property *award received* should be used when the film won; otherwise, we should use the property *nominated for*.

7.4 Future Work

One direction for future work to broaden the coverage and application of our work is to integrate with table format and layout detection systems [57] to ingest tables from different sources such as PDFs or Spreadsheets. As these systems are not error-free, it would be beneficial to expose and incorporate their intermediate representation into a semantic modeling method to create a robust end-to-end approach.

In domains where accurate extracted information is important, having a reliable estimate of the confidence of the predicted semantic descriptions is essential to prioritize human effort in verifying and curating the results. We plan to develop a method to combine the probabilities of predicted column relationships and column types to estimate a confidence score. We can also use the predicted semantic description to extract data. Any inconsistency in the extracted result can be used as extra signals to calibrate the score.

Another direction of future work is to improve the quality of the predicted semantic descriptions and address the limitations of our presented approaches such as GRAMS++. We plan to develop an algorithm to detect and filter out unknown columns in an input table. To handle tables in which each row has a different semantic description, we can model the data at the instance level. Training our methods on a larger dataset could also improve the performance. We can create more labeled tables by relaxing the filtering conditions discussed in Section 4.2.1 or via data augmentation by generating tables from known semantic descriptions.

Bibliography

[1] N. Abdelmageed, J. Chen, V. Cutrona, V. Efthymiou, O. Hassanzadeh, M. Hulsebos, E. Jimenez-Ruiz, J. Sequeda, and K. Srinivas. "Results of SemTab 2022". In: *Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab 2022)*. Vol. 3320. CEUR Workshop Proceedings. © 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0). CEUR, Jan. 2022. URL: https://openaccess.city.ac.uk/id/eprint/29744/.

[2] Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. "Hinge-Loss Markov Random Fields and Probabilistic Soft Logic". In: *Journal of machine learning research: JMLR* 18.109 (2017), pp. 1–67. ISSN: 1532-4435, 1533-7928. URL: https://www.jmlr.org/papers/v18/15-631.html.

[3] Karin Becker, Shiva Jahangiri, and Craig A. Knoblock. "A Quantitative Survey on the Use of the Cube". In: *Proceedings of the 3rd International Workshop on Semantic Statistics*. 2015.

[4] G Bhalotia, A Hulgeri, C Nakhe, S Chakrabarti, and S Sudarshan. "Keyword searching and browsing in databases using BANKS". In: *Proceedings 18th International Conference on Data Engineering*. Feb. 2002, pp. 431–440. DOI: 10.1109/ICDE.2002.994756.

[5] Rich Caruana. "Multitask Learning". In: *Machine learning* 28.1 (Jan. 1997), pp. 41–75. ISSN: 0885-6125, 1573-0565. DOI: 10.1023/A:1007379606734.

[6] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. *BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation*. 2024. arXiv: 2402.03216 [cs.CL].

[7] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. "ColNet: Embedding the Semantics of Web Tables for Column Type Prediction". en. In: *AAAI* 33.01 (July 2019), pp. 29–36.

[8] Shuang Chen, Alperen Karaoglu, Carina Negreanu, Tingting Ma, Jin-Ge Yao, Jack Williams, Andy Gordon, and Chin-Yew Lin. *LinkingPark: An integrated approach for semantic table interpretation.* http://ceur-ws.org/Vol-2775/paper7.pdf?ref=https://githubhelp.com. Accessed: 2023-10-6.

[9] Nick Craswell. "Mean Reciprocal Rank". In: *Encyclopedia of Database Systems*. Boston, MA: Springer US, 2009, pp. 1703–1703. ISBN: 9780387355443, 9780387399409. DOI: 10.1007/978-0-387-39940-9_488.

[10] Marco Cremaschi, Flavio De Paoli, Anisa Rula, and Blerina Spahiu. "A fully automated approach to a complete Semantic Table Interpretation". In: *Future generations computer systems: FGCS* 112 (Nov. 2020), pp. 478–500. ISSN: 0167-739X. DOI: 10.1016/j.future.2020.05.019.

[11] Marco Cremaschi, Anisa Rula, Alessandra Siano, and Flavio De Paoli. "MantisTable: a tool for creating semantic annotations on tabular data". In: *European Semantic Web Conference*. Springer. 2019, pp. 18–23.

[12] V. Cutrona, J. Chen, V. Efthymiou, O. Hassanzadeh, E. Jimenez-Ruiz, J. Sequeda, K. Srinivas, N. Abdelmageed, M. Hulsebos, D. Oliveira, and C. Pesquita. "Results of SemTab 2021". In: *20th International Semantic Web Conference*. Vol. 3103. Copyright © 2021 for the individual papers by the papers' authors. This volume and its papers are published under the Creative Commons License Attribution 4.0 International (CC BY 4.0). CEUR Workshop Proceedings, Mar. 2022, pp. 1–12. URL: https://openaccess.city.ac.uk/id/eprint/28056/.

[13] Ioannis Dasoulas, Duo Yang, Xuemin Duan, and Anastasia Dimou. "TorchicTab: Semantic Table Annotation with Wikidata and Language Models". In: *CEUR Workshop Proceedings*. CEUR Workshop Proceedings, 2023, pp. 21–37.

[14] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. "TURL: Table understanding through representation learning". In: (June 2020). arXiv: 2006.14806 [cs.IR].

[15] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data". In: *7th Workshop on Linked Data on the Web, Proceedings* 1184 (2014).

[16] Wilfrid J Dixon and Alexander M Mood. "The statistical sign test". In: *Journal of the American Statistical Association* 41.236 (1946), pp. 557–566.

[17] Zai-Wen Feng, Jia-Kang Xu, Wolfgang Mayer, Wang-Yu Huang, Ke-Qing He, Markus Stumptner, Georg Grossmann, Hong-Yu Zhang, and Lin Ling. *Automatic Semantic Modeling for Structural Data Source with the Prior Knowledge From Knowledge Graph*. 2021.

[18] Wolf Garbe. *SymSpell*. June 2012. URL: https://github.com/wolfgarbe/SymSpell.

[19] Yolanda Gil, Daniel Garijo, Deborah Khider, Craig A Knoblock, Varun Ratnakar, Maximiliano Osorio, Hernán Vargas, Minh Pham, Jay Pujara, Basel Shbita, et al. "Artificial intelligence for modeling complex systems: taming the complexity of expert models to improve decision making". In: *ACM Transactions on Interactive Intelligent Systems* 11.2 (2021), pp. 1–49.

[20] D Groeneveld, I Beltagy, P Walsh, A Bhagia, et al. "Olmo: Accelerating the science of language models". In: *arXiv preprint arXiv* (2024). URL: https://arxiv.org/abs/2402.00838.

[21] Shubham Gupta, Pedro Szekely, Craig A Knoblock, Aman Goel, Mohsen Taheriyan, and Maria Muslea. "Karma: A system for mapping structured sources into the Semantic Web". In: *Extended Semantic Web Conference*. Springer. 2012, pp. 430–434.

[22] Oktie Hassanzadeh, Nora Abdelmageed, Vasilis Efthymiou, Jiaoyan Chen, Vincenzo Cutrona, Madelon Hulsebos, Ernesto Jiménez-Ruiz, Aamod Khatiwada, Keti Korini, Benno Kruit, et al. "Results of SemTab 2023". In: *CEUR Workshop Proceedings*. Vol. 3557. 2023, pp. 1–14.

[23] Oktie Hassanzadeh, Vasilis Efthymiou, and Jiaoyan Chen. *SemTab 2022: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Data Sets - "Hard Tables" R1 and R2*. Version 2022 (Hard Tables). Zenodo, Dec. 2022. DOI: 10.5281/zenodo.7416036.

[24] Oktie Hassanzadeh, Vasilis Efthymiou, and Jiaoyan Chen. *SemTab 2023: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Data Sets - "Wikidata Tables" R1*. Version 2023 (Wikidata Tables, R1). Zenodo, Sept. 2023. DOI: 10.5281/zenodo.8393535.

[25] Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, Ernesto Jiménez-Ruiz, and Kavitha Srinivas. *SemTab 2020: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Data Sets*. Version 2020. Nov. 2020.

[26] Emil G Henriksen, Alan M Khorsid, Esben Nielsen, Adam M Stück, Andreas S Sørensen, and Olivier Pelgrin. *SemTex: A Hybrid Approach for Semantic Table Interpretation*. 2023.

[27] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. "Sherlock: A Deep Learning Approach to Semantic Data Type Detection". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, July 2019, pp. 1500–1508.

[28] Viet-Phi Huynh, Yoan Chabot, Thomas Labbé, Jixiong Liu, and Raphaël Troncy. "From heuristics to language models: A journey through the universe of semantic table interpretation with DAGOBAH". In: *Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)* (2022).

[29] E. Jimenez-Ruiz, O. Hassanzadeh, V. Efthymiou, J. Chen, K. Srinivas, and V. Cutrona. "Results of SemTab 2020". In: *CEUR Workshop Proceedings* 2775 (Jan. 2020). Copyright © 2020 for this paper by its authors., pp. 1–8. ISSN: 1613-0073. URL: http://ceur-ws.org/Vol-2775/.

[30] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (Dec. 2014). arXiv: 1412.6980 [cs.LG].

[31] Craig A Knoblock, Pedro Szekely, José Luis Ambite, Aman Goel, Shubham Gupta, Kristina Lerman, Maria Muslea, Mohsen Taheriyan, and Parag Mallick. "Semi-automatically Mapping Structured Sources into the Semantic Web". In: *The Semantic Web: Research and Applications*. Springer Berlin Heidelberg, 2012, pp. 375–390. DOI: 10.1007/978-3-642-30284-8_32.

[32] Craig A Knoblock, Pedro Szekely, Eleanor Fink, Duane Degler, David Newbury, Robert Sanderson, Kate Blanch, Sara Snyder, Nilay Chheda, Nimesh Jain, Ravi Raju Krishna, Nikhila Begur Sreekanth, and Yixiang Yao. "Lessons Learned in Building Linked Data for the American Art Collaborative". In: *The Semantic Web – ISWC 2017*. Springer International Publishing, 2017, pp. 263–279. DOI: 10.1007/978-3-319-68204-4_26.

[33] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. en. MIT Press, July 2009. ISBN: 9780262013192. URL: https://play.google.com/store/books/details?id=7dzpHCHzNQ4C.

[34] Keti Korini, Ralph Peeters, and Christian Bizer. "SOTAB: the WDC schema. org table annotation benchmark". In: *CEUR Workshop Proceedings*. Vol. 3320. RWTH Aachen. 2022, pp. 14–19.

[35] Andreas Langegger and Wolfram WoB. "XLWrap — Querying and Integrating Arbitrary Spreadsheets with SPARQL". In: *ISWC '09 Proceedings of the 8th International Semantic Web Conference*. 2009, pp. 359–374.

[36] Maxime Lefrancois, Antoine Zimmermann, and Noorani Bakerally. "A SPARQL extension for generating RDF from heterogeneous formats". In: *European Semantic Web Conference*. Vol. 10249. 2017, pp. 35–50.

[37] Xinhe Li, Shuxin Wang, Wei Zhou, Gongrui Zhang, Chenghuan Jiang, Tianyu Hong, and Peng Wang. *KGCODE-Tab Results for SemTab 2022*. https://ceur-ws.org/Vol-3320/paper5.pdf. Accessed: 2023-10-6.

[38] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. "Annotating and searching web tables using entities, types and relationships". en. In: *Proceedings of the VLDB Endowment*. Vol. 3. Singapore: VLDB Endowment, Sept. 2010, pp. 1338–1347.

[39] Jhomara Luzuriaga, Emir Munoz, Henry Rosales-Mendez, and Aidan Hogan. “Merging web tables for relation extraction with knowledge graphs”. In: *IEEE Trans. Knowl. Data Eng.* (2021), pp. 1–1.

[40] Franck Michel, Loic Djimenou, Catherine Faron Zucker, and Johan Montagnat. “Translation of relational and non-relational databases into RDF with xR2RML”. In: *11th International Confenrence on Web Information Systems and Technologies (WEBIST’15)*. 2015, pp. 443–454.

[41] Varish Mulwad, Tim Finin, and Anupam Joshi. “Semantic Message Passing for Generating Linked Data from Tables”. In: *The Semantic Web – ISWC 2013*. Springer Berlin Heidelberg, 2013, pp. 363–378.

[42] Phuc Nguyen, Ikuya Yamada, Natthawut Kertkeidkachorn, Ryutaro Ichise, and Hideaki Takeda. “Demonstration of MTab: Tabular Data Annotation with Knowledge Graphs”. In: (2021).

[43] Phuc Nguyen, Ikuya Yamada, Natthawut Kertkeidkachorn, Ryutaro Ichise, and Hideaki Takeda. *SemTab 2021: Tabular data annotation with MTab tool*. http://ceur-ws.org/Vol-3103/paper8.pdf. Accessed: 2023-10-6.

[44] Martin J O’Connor, Christian Halaschek-Wiener, and Mark A Musen. “M2: A Language for Mapping Spreadsheets to OWL.” In: *OWLED*. Vol. 614. 2010.

[45] Jeff Z Pan. “Resource Description Framework”. In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 71–90. ISBN: 9783540926733. DOI: 10.1007/978-3-540-92673-3_3.

[46] Minh Pham, Suresh Alse, Craig A Knoblock, and Pedro Szekely. “Semantic Labeling: A Domain-Independent Approach”. In: *The Semantic Web – ISWC 2016*. Springer International Publishing, 2016, pp. 446–462.

[47] S K Ramnandan, Amol Mittal, Craig A Knoblock, and Pedro Szekely. “Assigning Semantic Labels to Data Sources”. In: *The Semantic Web. Latest Advances and New Domains*. Springer International Publishing, 2015, pp. 403–417. DOI: 10.1007/978-3-319-18818-8_25.

[48] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. “On the Convergence of Adam and Beyond”. In: (Apr. 2019). arXiv: 1904.09237 [cs.LG]. URL: http://arxiv.org/abs/1904.09237.

[49] Nils Reimers and Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. In: (Aug. 2019). arXiv: 1908.10084 [cs.CL].

[50] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. “Matching HTML Tables to DBpedia”. In: *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*. WIMS ’15 Article 10. Larnaca, Cyprus: Association for Computing Machinery, July 2015, pp. 1–6.

[51] Natalia Ruemmele, Yuriy Tyshetskiy, and Alex Collins. “Evaluating approaches for supervised semantic labeling”. In: (Jan. 2018). arXiv: 1801.09788 [cs.LG]. URL: http://arxiv.org/abs/1801.09788.

[52] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A unified embedding for face recognition and clustering”. In: *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Dec. 2015), pp. 815–823. ISSN: 1063-6919, 2575-7075. DOI: 10.1109/CVPR.2015.7298682.

[53] Basel Shbita, Namrata Sharma, Binh Vu, Fandel Lin, and Craig A. Knoblock. “Constructing a Knowledge Graph of Historical Mining Data”. In: *6th International Workshop on Geospatial Linked Data (GeoLD) Co-located with the 21st Extended Semantic Web Conference (ESWC)*. CEUR. 2024.

[54] Renat Shigapov, Philipp Zumstein, Jan Kamlah, Lars Oberlander, Jorg Mechnich, and Irene Schumm. *bbw: Matching CSV to Wikidata via Meta-lookup*. https://madoc.bib.uni-mannheim.de/57386/3/paper2.pdf. Accessed: 2023-10-6.

[55] Jason Slepicka, Chengye Yin, Pedro Szekely, and Craig A. Knoblock. “KR2RML: An Alternative Interpretation of R2RML for Heterogenous Sources”. In: *Proceedings of the 6th International Workshop on Consuming Linked Data (COLD 2015)*. 2015.

[56] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. “Annotating columns with pre-trained language models”. In: *Proceedings of the 2022 International Conference on Management of Data*. Philadelphia PA USA: ACM, June 2022.

[57] Kexuan Sun, Harsha Rayudu, and Jay Pujara. “A hybrid probabilistic approach for table understanding”. In: *Proceedings of the ... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence* 35.5 (18 5 2021), pp. 4366–4374. ISSN: 2159-5399, 2374-3468. DOI: 10.1609/aaai.v35i5.16562.

[58] Charles Sutton and Andrew McCallum. “An Introduction to Conditional Random Fields”. In: *Foundations and Trends® in Machine Learning* 4.4 (2012), pp. 267–373. ISSN: 1935-8237. DOI: 10.1561/2200000013.

[59] Mohsen Taheriyan, Craig A Knoblock, Pedro Szekely, and José Luis Ambite. “Learning the semantics of structured data sources”. In: *Journal of Web Semantics* 37-38 (Mar. 2016), pp. 152–169.

[60] Mohsen Taheriyan, Craig A Knoblock, Pedro Szekely, and José Luis Ambite. "Leveraging linked data to discover semantic relations within data sources". In: *Lecture Notes in Computer Science*. Lecture notes in computer science. Cham: Springer International Publishing, 2016, pp. 549–565. ISBN: 9783319465227, 9783319465234. DOI: 10.1007/978-3-319-46523-4_33.

[61] H Touvron, L Martin, K Stone, P Albert, et al. "Llama 2: Open foundation and fine-tuned chat models". In: *arXiv preprint arXiv* (2023). URL: https://arxiv.org/abs/2307.09288.

[62] Diego de Uña, Nataliia Rümmele, G Gange, P Schachte, and Peter James Stuckey. "Machine Learning and Constraint Programming for relational-to-ontology schema mapping". In: *International Joint Conference on Artificial Intelligence* (July 2018), pp. 1277–1283. DOI: 10.24963/ijcai.2018/178.

[63] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. "Recovering semantics of tables on the web". In: *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases* 4.9 (June 2011), pp. 528–538. ISSN: 2150-8097. DOI: 10.14778/2002938.2002939.

[64] B Vu and C A Knoblock. "Sand: A tool for creating semantic descriptions of tabular sources". In: *European Semantic Web Conference* (2022). URL: https://link.springer.com/chapter/10.1007/978-3-031-11609-4_12.

[65] Binh Vu, Craig Knoblock, and Jay Pujara. "Learning Semantic Models of Data Sources Using Probabilistic Graphical Models". In: *The World Wide Web Conference*. WWW '19. San Francisco, CA, USA: Association for Computing Machinery, May 2019, pp. 1944–1953.

[66] Binh Vu, Craig A Knoblock, Pedro Szekely, Minh Pham, and Jay Pujara. "A Graph-Based Approach for Inferring Semantic Descriptions of Wikipedia Tables". In: *The Semantic Web – ISWC 2021*. Springer International Publishing, 2021, pp. 304–320.

[67] Binh Vu, Craig A Knoblock, Pedro Szekely, Minh Pham, and Jay Pujara. "A Graph-Based Approach for Inferring Semantic Descriptions of Wikipedia Tables". In: *International Semantic Web Conference*. Springer. 2021, pp. 304–320.

[68] Binh Vu, Jay Pujara, and Craig A Knoblock. "D-REPR: A Language for Describing and Mapping Diversely-Structured Data Sources to RDF". In: *Proceedings of the 10th International Conference on Knowledge Capture*. K-CAP '19. Marina Del Rey, CA, USA: Association for Computing Machinery, Sept. 2019, pp. 189–196.

[69] Binh Vu, Jay Pujara, and Craig A Knoblock. "D-REPR: a language for describing and mapping diversely-structured data sources to RDF". In: *Proceedings of the 10th International Conference on Knowledge Capture*. 2019, pp. 189–196.

[70] Alexander Yeh. “More accurate tests for the statistical significance of result differences”. In: *COLING 2000 Volume 2: The 18th International Conference on Computational Linguistics*. 2000. URL: https://aclanthology.org/C00-2137.

[71] Ziqi Zhang. “Effective and efficient Semantic Table Interpretation using TableMiner+”. In: *Semant. Web* 8.6 (Aug. 2017), pp. 921–957.

Appendices

A D-REPR: A Language for Describing and Mapping Diversely-Structured Data Sources to RDF

The Web provides an enormous number of valuable datasets, but the variety and complexity of these sources creates a barrier to their widespread use. The recent growth of knowledge graphs and the Linked Open Data movement have sought to overcome these barriers by providing the platforms and common semantic conventions to allow data to be semantically defined, interlinked between sources, and used by all. However, mapping data to RDF to publish into knowledge graphs demands expertise and significant effort because datasets are in different formats (e.g., spreadsheets, JSON, NetCDF) and different layouts (e.g., row-based, matrix, hierarchical layouts). In this appendix, we introduce D-REPR (Dataset Representation), a rich mapping language that addresses these two challenges while extending to many datasets that existing approaches (e.g., RML [15], XLWrap [35]) cannot easily model.

Many languages and tools have been proposed to make mapping data easier and less laborious. R2RML and its extensions [15, 40, 55] can map datasets with heterogeneous formats such as CSV, JSON, and XML. However, the mapped datasets have to be in a row-based layout, compliant with the Nested Relational Model (NRM). For datasets that are not in NRM layout (e.g., Figure 7.1),

format-specific solutions have been proposed. For instance, XLWrap is a powerful tool to map spreadsheets to RDF, but cannot deal with other data formats. No single data mapping tool can handle the diverse set of data format and data layout choices currently in use.

Having one general RDF mapping language for many types of data sources is desirable as it currently requires much time and effort for end-users to learn different tools for different kinds of sources. However, designing such a language is challenging for several reasons. First, data sources with different formats have different protocols to access and refer to data values. Even with the same data format, data sources can be represented in different layouts. For example, a life expectancy dataset in CSV format can organize its data in matrix or in row-based layout (Figures 7.1a and 7.1b), or a museum dataset in JSON format can organize its data in a custom layout or NRM layout (Figures 7.1c and 7.1d).

To address these issues, we present a novel data description language for converting data to RDF. In our approach, users define locations of source attributes in the dataset using JSONPath, then create a semantic description that specifies semantic types of attributes by mapping each attribute to a property of an ontology class, and semantic relations between them. Finally, users define simple rules to join attributes to create tables containing instances of ontology classes. The language also has built-in support for data cleaning using transformation functions.

Our contribution is a novel data description language that makes it easy for mapping format and layout heterogeneous datasets to RDF. Our approach is capable of mapping a wide variety of data sources and goes beyond the set of sources that existing mapping languages support. Furthermore, the language is extensible to new formats and new data layouts.

a. life table (csv format, original layout)

		2016		2015	
		Male	Female	Male	Female
ex - expectation of life at age x	<1 year	57.7	59.6	57.3	59.1
	1-4 years	60.6	62.1	60.3	61.7
	5-9 years	58.7	60.2	58.4	59.9
	10-14 years	54.4	56	54.2	55.6
	15-19 years	49.8	51.3	49.6	51

b. life table (csv format, row-based layout)

Indicator	Age group	Year	Gender	Value
ex - expectation	<1 year	2016	Male	57.7
ex - expectation	<1 year	2016	Female	59.6
ex - expectation	<1 year	2015	Male	57.3
ex - expectation	<1 year	2015	Female	59.1
ex - expectation	<1 year	2014	Male	56.2
ex - expectation	<1 year	2014	Female	58.4

c. museum dataset (json format, original layout)

```
{
  "url": "https://npg.si.edu/object/npg_NPG.70.36",
  "title": "Apollo 11 Crew",
  "sitter": [
    "Neil Alden Armstrong",
    "Michael Collins"
  ],
  "sitter_born_died_date": [
    "5 Aug 1930 - 25 Aug 2012",
    "born 20 Jan 1930"
  ]
}
```

d. museum dataset (json format, NRM layout)

```
{
  "url": "https://npg.si.edu/object/npg_NPG.70.36",
  "title": "Apollo 11 Crew",
  "sitters": [
    {
      "name": "Neil Alden Armstrong",
      "born_died_date": "5 Aug 1930 - 25 Aug 2012"
    },
    {
      "name": "Michael Collins",
      "born_died_date": "born 20 Jan 1930"
    }
  ]
}
```

e. precipitation dataset (netCDF4 format)

```
latitude: float[50]
longitude: float[30]
time: float[100]
precipitation: float[50][30][100]
```

Figure 7.1: Three datasets that have different formats and layouts. Their data is truncated for readability. In the precipitation dataset, only the schema is shown.

A.1 Motivating Example and Problem Requirements

In this section, we provide examples of diversely-structured data sources to illustrate the challenges of mapping them to RDF, the limitations of previous mapping methods, and the requirements of the generic mapping languages.

Figures 7.1a and 7.1b show a life expectancy table for the population of South Sudan[¶] in matrix and row-based layouts, respectively. The original table (Figure 7.1a) contains observations (green matrix) of indicators (dark blue column) such as life expectancy. These observations were collected over several years (light blue row) for different gender (yellow row) and age groups (orange column). This original data can be represented in RDF using the Data Cube vocabulary[‖] but manually generating this representation is cumbersome and it is difficult to generate consistently across multiple sources [3]. Specifically, this representation requires dealing with the complex, nested matrix layout, imputing missing values, such as the implicit years in the light blue row, and transforming values, such as converting age groups into a range of age values.

[¶] https://apps.who.int/gho/data/view.main.LT62270
[‖] https://www.w3.org/TR/vocab-data-cube/

Task-specific tools, such as XLWrap and M2, have been developed to map these types of datasets to RDF, however they cannot be used for all datasets.

Figures 7.1c and 7.1d show a second data source formatted in JSON that contains a list of artworks from the National Portrait Gallery (NPG). Each artwork also includes information about its creator and sitters. As XLWrap and M2 do not support JSON data, users have to rely on extensions of R2RML instead. These extensions are created to support mapping sources with various formats such as JSON, XML, RDBMS, etc. However, these extensions are limited to the layout conventions used in the Nested Relational Model (NRM). In the original dataset (Figure 7.1c), the records do not follow the conventions of the NRM because sitter names and sitter birth dates are stored in two separated arrays, whose values are associated by their orderings. To handle these scenarios, a different system has to be used to transform the structure of the source into the layout in Figure 7.1d (e.g., zip the two arrays to create one array) using transformation functions as in KR2RML [55].

Figure 7.1e shows an extreme example of a rainfall dataset, in which volume of rainfall (*precipitation*[x][y][z]) is recorded at a specific location (*latitude*[x], *longitude*[y]) and at a specific time (*time*[z]). This dataset is one of many scientific datasets that have customized layout for efficiently storing or manipulating the data. Since sharing scientific knowledge can accelerate new findings, it is important that we be able to easily map these datasets to RDF.

As no existing generic mapping language can handle all five examples above, users have to spend significant effort learning to use different tools for different kinds of sources. As we have discussed in the previous section, one reason is the existing generic languages lack the ability to express the layout of data sources. This poses a new requirement that the languages need

to support. Together with the requirements from [15, 36, 40], we create a list of 6 important requirements that the generic mapping language should meet as follows.

R1 Map sources with heterogeneous formats and be extensible to new data formats.

R2 Map sources with heterogeneous layouts and be extensible to custom layouts.

R3 The language is uniform so that extending to new formats and layouts only requires changes in the implementation (e.g., adding plugins), but not in the language itself.

R4 Support interlinking across sources, e.g., generating links between individuals of different classes.

R5 Support cleaning and transforming data.

R6 Support general mapping RDF functionalities, e.g., specifying the data types and used ontologies, generating blank nodes.

Based on the above requirements, in the next section, we will describe D-REPR, a new mapping language, that addresses the above use cases and supports a broad set of capabilities.

A.2 Dataset Representation Language

In this section, we describe our dataset representation language (D-REPR) and its usage in the context of a sample dataset in Figure 7.2. The sample dataset contains a CSV file that is similar to the life table in Figure 7.1a, except that the indicator column contains the indicator code. The dataset has another JSON file that has the indicators' definitions.

Modeling a dataset in D-REPR requires defining four basic components: resources, attributes of data sources, a semantic description that maps the attributes to properties of ontology classes, and rules to join values of these attributes.

a. life table.csv

		2016	
Indicator	Age Group	Male	Female
LIFE_0035	<1 year	57.7	59.6
LIFE_0035	1-4 years	60.6	62.1

b. indicators.json

```
{
  "indicator": "LIFE_0035",
  "url": "http://apps.who/int/ ... /indicator.aspx?iid=35"
},
{
  "indicator": "LIFE_0029",
  "url": "http://apps.who/int/ ... /indicator.aspx?iid=29"
},
```

Figure 7.2: A sample life table dataset.

In addition, many data sources have useful data invariants (e.g., missing values) or benefit from pre-processing functions that D-REPR also supports. Listing 1 shows an overview of the D-REPR syntax for modeling dataset. We use YAML as it is concise and human-friendly to write. However, the D-REPR specification can also be expressed in other languages such as JSON.

A.2.1 Resources and attributes of the dataset

Resources The first step of defining a dataset is describing its resources. A dataset may consist of many, interdependent logical resources such as codebooks or data-definition dictionaries. These logical resources are defined under a section titled `resources`. Each resource is associated with a user-defined name and a format. In the sample dataset, we have two resources: a life table and a list of indicators, whose definitions are in Listing 2.

```
resources:
    <resource_id>:
        type: <resource_type>
        # ..optional arguments depend on the resource type..
    # ..other resources..
[preprocessing]:
    - type: <function type>
      input:
          [resource]: <resource_id>
          path: <json_path>
      [output]: <new_resource_id>
      [code]: <code>
    # ..other transformation functions..
attributes:
    <attribute_id>:
        [resource_id]: <resource_id>
        path: <json_path>
        [unique]: false
        [missing_values]: [<value_0>, <value_1>, ...]
    # ..other attributes..
alignments:
    - type: <join_type>
      source: <attribute_id>
      target: <attribute_id>
      # ..optional arguments depend on the alignment type..
    # ..other alignments..
semantic_model:
    data_nodes:
        <attribute_id>: <class_id>--<predicate>[^^<data_type>]
        # ..other attributes..
    [relations]:
        - <source_class_id>--<predicate>--<target_class_id>
        # ..other relations..
    [subjects]:
        <class_id>: <attribute_id>
        # ..other subjects..
    [prefixes]:
        <prefix>: <iri>
        # ..other prefixes..
```

Listing 1: The D-REPR YAML syntax to describe a dataset. Optional properties are surrounded by brackets (e.g., `[missing_values]` is optional)

```
resources:
  life_tbl:
    type: csv
    delimiter: ","
  indicators:
    type: json
```

Listing 2: Resources' definition of the sample dataset

Different data formats have different ways of referring to data elements. For example, XML and JSON documents have XPath and JSONPath, respectively, or CSV and Spreadsheet can refer to cells by sheet name, row and column number. In addition, query results from relational and non-relational database such as graph databases can also be manipulated using well-known formats such as JSON or row-based table. Therefore, it is safe to view resource's data as a general tree structure similar to JSON tree. For example, a dataset in the NetCDF format can be viewed as a JSON object, where variables in the dataset are properties of the object, values of the variables (multi-dimensional arrays) are nested JSON arrays. Similarly, a CSV table can also be viewed as a nested JSON array.

Viewing resource's data as a tree has several benefits. First, it facilitates downstream tasks such as data cleaning to be independent of the data formats. Second, it enables users to use one single query path language such as JSONPath to refer to a resource's elements. Finally, it reduces the implementation effort as we only need to implement one path language per format. This is different from other generic mapping languages such as RML [15] or xR2RML [40] as they allow users to use various path languages via `rml:referenceFormulation`.

We use JSONPath expressions to select data regions in a resource. The query starts with `$` as the root of a resource. To select a child of the current element, we use `.<key>` or `[<key>]`. To

```
attributes:
  year:
    resource_id: life_tbl
    path: $[0][2:]
  gender:
    resource_id: life_tbl
    path: $[1][2:]
  indicator_column:
    resource_id: life_tbl
    path: $[2:][0]
  age_group:
    resource_id: life_tbl
    path: $[2:][1]
  observation:
    resource_id: life_tbl
    path: $[2:][2:]
  indicator:
    resource_id: indicators
    path: $.*.indicator
    unique: true
  url:
    resource_id: indicators
    path: $.*.url
    unique: true
```

Listing 3: Attributes' definition of the sample dataset

select a range of child nodes, we use the array slice operator `[<start>:<end>:<step>]`. To select all child nodes, we use either `.*` or `[*]`. For example, indicator urls in the JSON resource can be retrieved by `$[*].url` or `$[:].url`, and observations (green cells) in the life table can be selected by `$[2:][2:]`.

Attributes The next element in the D-REPR language is a block of `attributes` that designates specific data attributes found in the data source. Each attribute is first defined by providing a unique, human-readable name. Next, each attribute is associated with a spatial location where

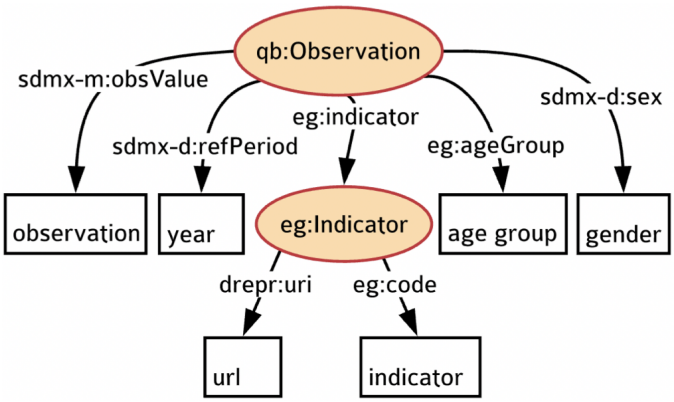
the attribute is physically located. The location includes both `resource_id` and `path`, which is a path expression to its values in the resource using the JSONPath.

In addition to specifying the location of an attribute, the D-REPR specification can optionally include particular values that should be interpreted as missing values (e.g., -999) in the `missing_values` property. It can also specify whether values of an attribute are unique by setting the `unique` property to be true (e.g., the *indicator* attribute contains all unique values). Listing 3 shows the definition of the attributes of the sample dataset.

A.2.2 Semantic description of the dataset

After identifying resources and source attributes, we specify the semantic description for mapping data to RDF. The semantic description is a graph, in which internal nodes are ontology classes, leaf nodes are attributes (also called data nodes) and edges are ontology predicates. The description can be defined under the `semantic_model` block in Listing 1. Figure 7.3 depicts a semantic description and a subset of the generated RDF triples of the sample dataset.

The first step in defining the semantic description is to associate each attribute with a semantic type, which is a pair of an ontology class C and an ontology predicate p, denoted as $\langle C, p \rangle$. This creates predicate-object pairs for each instance of the ontology class C. The subject of each instance is created from the other attribute whose semantic type is of the same class C and uses a particular predicate `drepr:uri`. If there is no such attribute, then the instances will be blank nodes. We sometimes refer to attributes that are mapped to properties of a class as attributes of the class for short. For instance in Figure 7.3, the semantic type of the attribute *observation* is defined as `"qb:Observation:1-sdmx-m:obsValue^^xsd:decimal"` means that it is associated with a pair of ontology class and predicate $\langle$`qb:Observation`, `sdmx-m:obsValue`$\rangle$ and with the




```
_:o00 a qb:Observation;
  sdmx-m:obsValue "57.7"^^xsd:decimal;
  sdmx-d:refPeriod "2016";
  eg:indicator <http:// ... ?iid=35>;
  eg:ageGroup "<1 year";
  sdmx-d:sex "Male" .

<http:// ... ?iid=35> a eg:Indicator;
  eg:code "LIFE_0035" .
```

```
semantic_model:
  data_nodes:
    observation: qb:Observation:1--sdmx-m:obsValue^^xsd:decimal
    year: qb:Observation:1--sdmx-d:refPeriod
    age_group: qb:Observation:1--eg:ageGroup
    gender: qb:Observation:1--sdmx-d:sex
    indicator: eg:Indicator:1--eg:code
    url: eg:Indicator:1--drepr:uri
  relations:
    - qb:Observation:1--eg:indicator--eg:Indicator:1
  prefixes:
    qb: http://purl.org/linked-data/cube#
    sdmx-m: http://purl.org/linked-data/sdmx/2009/measure#
    sdmx-d: http://purl.org/linked-data/sdmx/2009/dimension#
    eg: http://example.org
  subjects:
    "qb:Observation:1": observation
    "eg:Indicator:1": indicator
```

Figure 7.3: A semantic model of the sample dataset

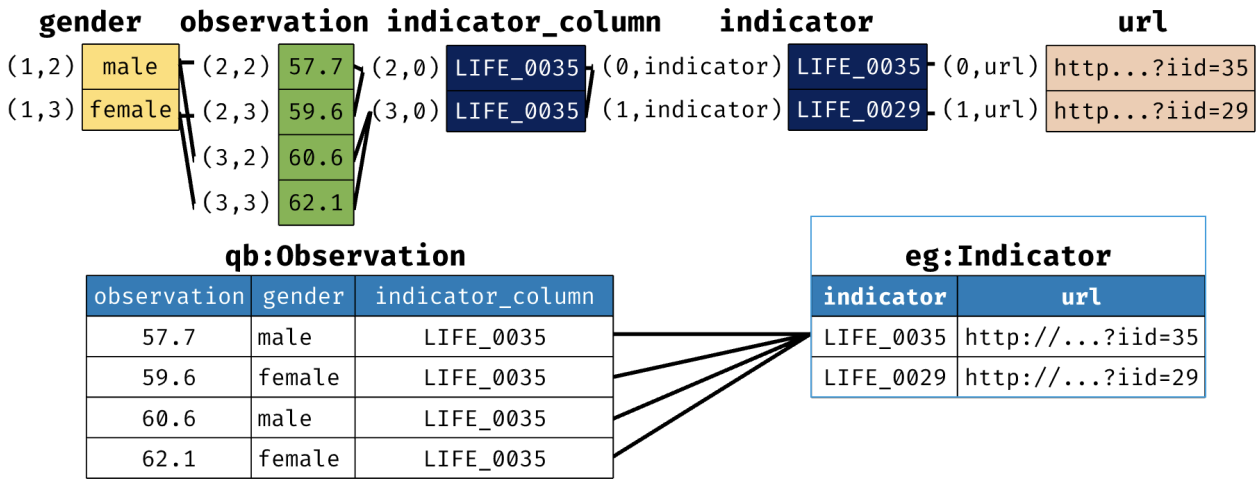
data type `xsd:decimal`. Notice in the definition, we have the ontology classes appended with an extra number (`qb:Observation:1`), which are used as identifiers to uniquely identify the class node. The reason is that there can be multiple class nodes that have the same label in the semantic description. For instance, in the NPG museum dataset, we have two `foaf:Person` classes, one is for creators of the artworks, one is for sitters in the artworks.

In addition to semantic types, semantic relations between instances are determined by edges (labeled with ontology predicates) between their classes. We can specify the semantic relations under the `relations` property. For example, the relationship `eg:indicator` of the `qb:Observation` and `eg:Indicator` is specified as `"qb:Observation:1-eg:indicator-eg:Indicator:1"`. Together with the semantic types defined in the `data_nodes` section, they form a graph of the semantic description as in Figure 7.3.

Moreover, as the IRIs of ontology classes and predicates are quite wordy, D-REPR allows users to use prefixed names similar to the Turtle language. The list of prefix labels and their corresponding IRIs are defined under the `prefixes` property.

Finally, D-REPR and other mapping languages assume that for each ontology class, there is at least one attribute whose values are in one-to-one correspondence with the instances of the class. In other words, we can iterate through the values and generate instance one by one. We say this attribute is the subject attribute of the class. In D-REPR, these subject attributes can usually be inferred automatically as shown in Section A.3.1. This eases the task of updating the semantic description of the dataset, which occurs quite frequently in publishing datasets as 5-star Linked Data [32]. However, users can explicitly specify them in the `subjects` property (Figure 7.3).

A.2.3 Joining the values of attributes of a dataset




```
alignments:
  - type: dimension
    source: observation
    target: gender
    aligned_dims: [{ source: 1, target: 1 }]
  - type: dimension
    source: observation
    target: indicator_column
    aligned_dims: [{ source: 0, target: 0 }]
  - type: value
    source: indicator_column
    target: indicator
  - type: dimension
    source: indicator
    target: url
    aligned_dims: [{ source: 0, target: 0 }]
  # .. more joins
```

Figure 7.4: Joining between the attributes in the sample dataset. Similar joins between ⟨*observation*, *year*⟩ and ⟨*observation*, *age group*⟩ were omitted for readability

After users define the attributes in the dataset, the values of each attribute are retrieved and represented as an array. Because there is no alignment between these arrays, to generate instances of an ontology class, we need to join together values of the attributes that were mapped to the properties of a class. Figure 7.4 illustrates how attributes in the sample dataset should be joined. In particular, *observation* and *gender* are joined by their column position, *observation* and *indicator_column* are joined by their row position, and *indicator_column* and *indicator* are joined by their values.

In D-REPR, a list of joins between the attributes is declared under the section titled `alignments` (Listing 1). Each join is required to define three properties, which are join type (`type`) and the two attributes involved in the function (`source` and `target`). D-REPR currently supports two join functions: joining by values (called value join) and by value position in the resources (called position join). The types for the two functions are `value` and `dimension`, respectively.

Value join is a traditional equity join in SQL. A value join only requires a source and target. Figure 7.4 shows an example of a value join between the *indicator_column* and *indicator* attributes.

Position join combines values if their positions are matched. A position of an attribute's value is a path to the value in the resource, and we refer to each item at index i in the path as dimension i. For example, the age group "1-4 years" is at position $p =$ `[3, 1]`, the value of dimension 0 is $p[0] = 3$ and the value of dimension 1 is $p[1] = 1$. Given a set of pairs of aligned dimensions $S = \{(x_1, y_1), (x_2, y_2), ...\}$, in which x_i, y_i are dimensions of attribute x and y, respectively, two positions p_x and p_y of x and y are considered as matched when $\forall (x_i, y_i) \in S : \frac{p_y[y_i] - y_i^{\text{start}}}{y_i^{\text{step}}} = \frac{p_x[x_i] - x_i^{\text{start}}}{x_i^{\text{step}}}$, where x_i^{start} and x_i^{step} are start and step of a range index of dimension x_i. A position join takes an extra property, which is the set S in `aligned_dims` property. An example of a position join is a join between *indicator* and *url* in Figure 7.4, where the aligned dimensions are

`[{ source: 0, target: 0}]`, which means a value of *indicator* at position `[1, indicator]` only matches with a value of *url* at `[1, url]` not at `[0, url]`.

Given a list of joins, instances of an ontology class are created by first creating a single column table of its subject attribute. Then, we sequentially join every other attribute of the class to the table using the join function between the attribute and the subject attribute. Recall that values of the subject attribute are assumed to have a one-to-one correspondence to instances of the class. Therefore, the size of the joined table does not change after each attribute is added and each row in the table contains one instance of the class. For example, the table of the class `qb:Observation` is created by joining the subject attribute *observation* and *gender*, then joining *indicator_column* to the table (Figure 7.4). This requires users to define $n-1$ joins between subject attributes and other attributes for n attributes. However, as we discussed in the previous section, subject attributes are usually inferred automatically. Therefore, users may inadvertently define join functions of non-subject attributes. For example, in the sample dataset, a user might define join functions between $\langle$*observation*, *gender*$\rangle$ and $\langle$*gender*, *year*$\rangle$. A join function between $\langle$*observation*, *year*$\rangle$ is missing, but D-REPR can complete missing join functions between subject and non-subject attributes, which is described in Section A.3.1. As a result, users can focus on specifying the alignments between $n-1$ pairs of attributes that explain the layout of the dataset. If D-REPR cannot infer missing functions, the system will ask users to provide the correct definitions.

A.2.4 Data cleaning and data transformation

We use transformation functions to facilitate data cleaning and data transformation. D-REPR defines a list of transformation functions under the `preprocessing` keyword as shown in Listing 1. These functions are executed sequentially before the mapping process starts. Each

```
preprocessing:
  - type: pmap
    input:
      resource: life_tbl
      path: $[0][2:]
    code: |
      if value == "":
        return context.get_left_value(index)
      return value
```

Listing 4: A transformation function for the sample dataset, it replaces empty light blue cells (years) with values in the left cells

function is applied to a data region specified in the `input` property using the JSONPath. If the `output` property is also defined, the function runs and produces new data stored in the new resource. Otherwise, the function runs and mutates data in the `input` region. The optional `code` property allows users to write one-time-use ad-hoc data cleaning code. Transformation functions in external libraries can also be used by importing them to the D-REPR processor and specifying their IDs in the `type` property. Hence, users can reuse transformation functions across datasets.

One concrete example of the above transformation functions is the mapping function implemented in the prototype of the D-REPR processor. The function maps each data element in the `input` property to a new data element using python code defined in the `code` property. The python code takes three arguments ⟨`value`, `index`, `context`⟩ and computes and returns a new value. The `value` variable is the value of the current data element, the `index` variable is the element's position, and the `context` variable is a python helper object that has some methods to query resource data. Listing 4 shows an illustration of the mapping function for the sample dataset. In the example, we use a method `context.get_left_value(index)` to access to the element on the left of the current element (e.g., given the `index` is `[0, 3]`, the method returns value at position `[0, 2]`).

A.3 Implementation and Evaluation

In this section, we describe one possible implementation of the D-REPR specification and evaluate the ability of D-REPR to map real-world datasets in different formats and layouts. A prototype of the D-REPR processor[**] is available online for evaluation. We compare the runtime between the prototype of the D-REPR processor and popular processors of R2RML extensions to evaluate the ability of the D-REPR processor to handle large datasets.

A.3.1 Algorithm for RDF generation

In this implementation, the D-REPR processor generates RDF in three steps. First, the system create execution plans that determine the order for generating instances of the ontology classes in the semantic description. In case users do not provide the subject attributes, the system will infer the subject attributes of these classes and determine missing join functions between the subject attributes and the non-subject attributes. The next step is to execute preprocessing functions one by one. Finally, the system iterates through each ontology class in the semantic description and outputs their instances.

To generate instances of a class, we need to join the values of the attributes of the class. A join function f between attribute a_i and a_j ($f : a_i \rightarrow a_j$) has an abstract interface as follows:

$$f(d_i \in V_i) = \{d_j | d_j \in V_j\}$$

in which V_* is a list of pairs of values and value positions of attribute a_*. The function f takes an element d_i of V_i of attribute a_i and return a set of elements of a_j matched with d_i. For example,

[**] https://github.com/usc-isi-i2/d-repr

in the sample dataset, V_{gender} of attribute *gender* is `[("male", [1, 2]), ("female", [1, 3])]`. Given a value d_i = `("male", [1, 2])` $\in V_{\text{gender}}$, the join function f : *gender* $\rightarrow$ *observation* will return a set `{(57.7, [2, 2]), (60.6, [3, 2])}`. This abstract interface enables us to hide the details of the join function (such as value join or position join) and its implementation (such as hash join or sort-merge join), and enables us to incorporate new join functions easily.

Given a subject attribute and a list of join functions between the subject attribute and other attributes of a class, we can generate instances of the class using Algorithm 9. Specifically, the algorithm iterates each value of the subject attribute to create one instance and uses the join functions to get properties of the instance (lines 4 - 6).

Algorithm 9: GENERATING INSTANCES OF A CLASS

Input: A subject attribute of class C: a
A list of values and their positions of a: V_a
A hash map from an attribute a_i of C to a join function between a and a_i: $\mathcal{F} = \{a_i \rightarrow f\}$
Output: List of instances of class C

```
1 instances ← []
2 for d ← V_a do
3     instance ← {}
4     for f ← F do
5         a_i ← target variable of f
6         instance.a_i = f (d)
7     instances.append(instance)
8 return instances
```

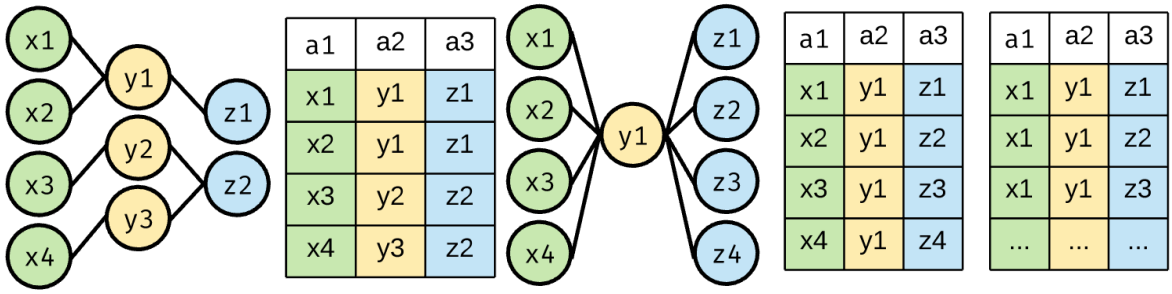
Inferring subject attributes The next step is to infer a subject attribute of a class and missing join functions between attributes. Let $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ be a set of attributes of class C. We have $a \in \mathcal{A}$ is a subject attribute of C, if and only if for every pair of attribute a and $a_i \in \mathcal{A}$, the degree of relationship is one-to-one or many-to-one (i.e., the join function f between a and a_i satisfies: $\forall d_i \in V : |f(d_i)| \leq 1$). Indeed, if we join all $a_i \in \mathcal{A}$ with a, every row in the result table

is unique, and the table contains all possible instances of the class C in the dataset. Therefore, a is the subject attribute of C.

If there is no such attribute in $\mathcal{A}$, we may try to pick another attribute a' in the dataset such that its degree of relationship with every attribute $a_i \in \mathcal{A}$ is one-to-one or many-to-one. This condition is similar to the above rule. However, with a', the joined table now has an extra column a'. To obtain the table of instances of C, we have to remove column a', and potentially filter out duplicated instances in the table. If there are no duplicates, a' can be chosen as the subject attribute of C.

We can determine the cardinality between two attributes based on the join function between them, conservatively. If attributes a_1 and a_2 are joined by value, and values of a_1 or a_2 are unique, then their degree of relationship is either one-to-* or *-to-one. If none of them are unique, their cardinality is many-to-many. For example, in the sample dataset, the cardinality between *indicator_column* and *indicator* is many-to-one. The cardinality of position join is computed based on non-aligned dimensions between two attributes. For example, in the sample dataset, *observation* (location: `$[2:][2:]`) and *year* (location: `$[0][2:]`) have one aligned dimension (dimension 1). Because the dimension 0 of *observation* is specified by the array slice operator (`[2:]`), their cardinality is many-to-one. However, the degree of relationship between *year* and *gender* (location: `$[1][2:]`) is one-to-one as their first dimensions are specified by array index operators (`[0]` and `[1]`).

Chaining join functions between attributes Suppose that users define join functions of $\langle a_1, a_2 \rangle$ and $\langle a_2, a_3 \rangle$. However, they do not specify any rule to join attributes a_1 with a_3. If the cardinality of $\langle a_1, a_2 \rangle$ is either one-to-many or one-to-one, or the cardinality of $\langle a_2, a_3 \rangle$ is either



(a) Only one possible table created from the join between attributes a_1 and a_3

(b) Cannot infer how to join a_1 and a_3 because there is more than one possible result

Figure 7.5: Chaining join functions between attributes $\langle a_1, a_2 \rangle$ and $\langle a_2, a_3 \rangle$

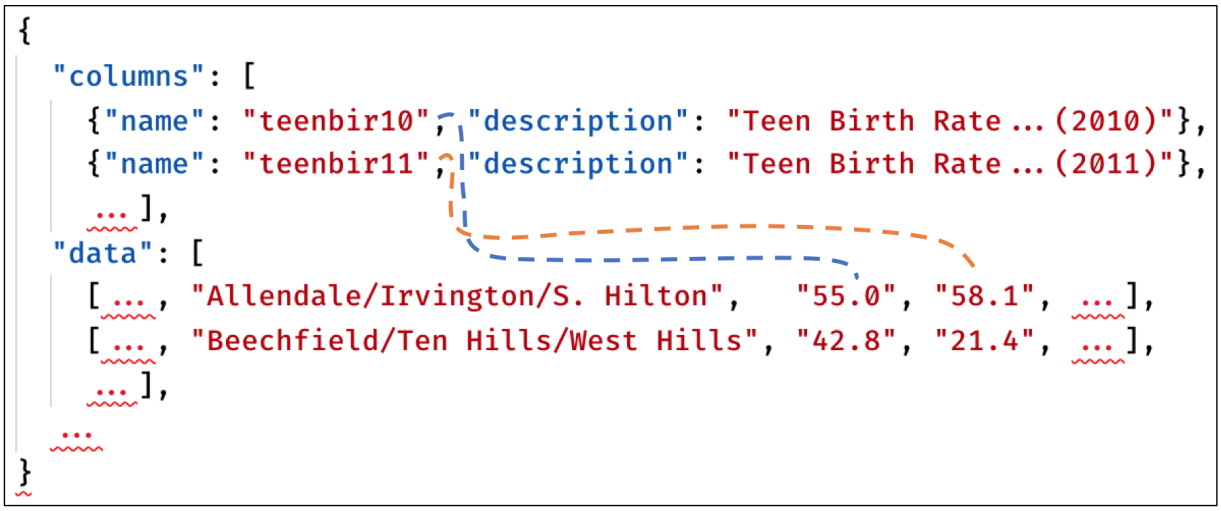
many-to-one or one-to-one, we can compute the join function of a_1 and a_3 as a chained join function that combines of a_1 and a_2 to obtain table $T_1(a_1, a_2)$, before joining a_3 with T_1. The intuition is that it is the only possible alignment between values of a_1 and a_3 which is consistent with the two defined join functions. The chained join is illustrated in Figure 7.5a. An example in Figure 7.5b shows that we cannot infer the join function of a_1 and a_3 when the cardinality of $\langle a_1, a_2 \rangle$ is many-to-one and $\langle a_2, a_3 \rangle$ is one-to-many.

A.3.2 Evaluation

To assess the capabilities of D-REPR to model datasets in different formats and layouts, we crawled 10,000 public datasets on data.gov, then filtered and randomly sampled 700 datasets in CSV, JSON, XML, Spreadsheet and NetCDF formats. Within these 700 datasets, we manually selected distinct datasets of different formats or layouts, as datasets in the same format and layout will have similar D-REPR models. The resulting datasets are diverse[††]. For example, some of them have complex layouts, such as nested JSON arrays with column definitions in another property (Figure 7.6a). Data in some datasets also need cleaning (e.g., date-time reformatting or extracting hierarchical structure from a column (Figure 7.6b)). In order to handle all of the datasets, a

[††]The experiments are available at http://purl.org/tty1/drepr-exp

a. Children and Family Health



b. Sugar production by sugar beet and sugarcane processors

FY 2008	JAN	FEB	MAR	APR	MAY	JUN
From domestic sugar beets	661,586	485,126	423,775	337,473	216,526	82,987
From imported sugar beets	0	37,160	0	0	0	0
Subtotal	661,586	522,287	423,775	337,473	216,526	82,987
Cane production:						
Florida	321,414	253,438	242,560	92,302	47,237	0
...	...					
Subtotal	378,919	283,190	289,237	108,826	68,504	30,903
Total	1,040,505	805,476	713,012	446,298	285,030	113,889

Figure 7.6: Examples of datasets with complex layouts in data.gov. In figure (a), data is in nested JSON array where each column definition is defined in the *columns* property (40 columns). Figure (b) is a dataset that has hierarchical structure (green column).

mapping language is required to satisfy the six requirements in Section A.1. We are able to build D-REPR models for all of them, demonstrating that our language meets the requirements.

To verify that the D-REPR processor can handle large datasets, we compare the runtime of our prototype with two popular processors of R2RML extensions (KR2RML [55] and Morph[‡‡]) on the task of mapping large CSV files (NRM layout). All CSV files contain people information (e.g., name, phone and address). The only difference between these files is the number of records. Our experiments are run on a Macbook Pro, Intel i7-7660U with 16GB RAM. The average runtime is

[‡‡] https://github.com/oeg-upm/morph-rdb

Table 7.1: Average running time (ms) of D-REPR and popular processors of R2RML extensions

	Number of records				
Tools	5,000	10,000	20,000	40,000	80,000
D-REPR	33.44	69.84	132.00	267.50	551.24
KR2RML	1368.00	1776.33	3276.66	4990.33	8305.33
Morph	4812.00	14949.66	65961.33	-	-

reported in Table 7.1. The runtime of the D-REPR processor increases linearly with the number of records. On average, it can generate 1.3 million triples per second, which is 15 times faster than KR2RML. The reasons that D-REPR is fast are its core engine is implemented using the Rust language; RDF triples are generated using a streaming approach; and it leverages properties of attributes to generate an efficient execution plan (e.g., missing value checks are not needed if there are no missing values).

A.4 Related Work

There are many proposed mapping languages or tools to transform data sources into RDF. W3C communities have listed many of them on their website. We can classify them into three different groups.

The first group of tools is specific to certain formats and structures such as bibtex2rdf, or email2rdf. As they are designed for one particular type of source, they may be the most effective tools to generate RDF for that source, but they do not apply to other source formats.

The second group is languages that work with tabular data. It is hard to convert tabular data as they have arbitrary ways of storing and representing data. XLWrap [35] is the language designed for mapping spreadsheets with various layouts to RDF. Users first describe a template graph similar to a semantic description, in which leaf nodes (known as data nodes) are bound

to cells in the table. Then, they define a transformation operation that shifts rows and columns in the spreadsheet such that after every shift, the data nodes in the template graph are bound to new values and produces a new instance. Corcho et al. [44] present the M^2 language, which uses the Manchester Syntax for converting data from spreadsheets into OWL. The language is less verbose than XLWrap and also able to handle many different tabular layouts. The main drawback of these mapping languages is although they can map datasets with heterogeneous layouts, they are limited to only tabular formats such as CSV or spreadsheets.

The third group contains generic mapping languages for integrating heterogeneous data sources. Dimou et al. [15] propose RML, an extension over R2RML, for mapping data sources with different formats and interlinking between sources. In particular, users describe logical sources and their formats, iterators to iterate through records in the sources, and how to map each record's attributes. Michel et al. [40] introduce xR2RML that extends R2RML and RML to support different types of databases from relational databases to non-relational databases such as Cassandra or MongoDB. Slepika et al. [55] present another extension of R2RML, KR2RML. Similar to xR2RML, it supports heterogeneous hierarchical sources. KR2RML also allows users to perform data cleaning or modifying data structures so that users can model noisy hierarchical data sources such as the NPG museum dataset (Figure 1c). Lefrançois et al. [36] develop SPARQL-Generate based on a SPARQL that allows querying a combination of RDF and non-RDF sources. Although the listed languages are capable of handling data sources with heterogeneous formats, the main difference between them and D-REPR is that they only work with the NRM layout.

A.5 Summary

In this appendix, we presented a novel dataset representation language, D-REPR, for mapping data to RDF. To handle datasets with heterogeneous formats, the language enables users to use JSONPath to select values of each dataset attribute. With datasets in different layouts, it allows users to describe join rules to combine the values, which capture the data alignments within each layout. Users can specify ontology classes and predicates that the data should be mapped to using semantic descriptions. The language also supports data cleaning and data transformation using pre-processing functions. The language is designed to be extensible. We can add a new data format by implementing the JSONPath for the format. We can also incorporate a new join rule by implementing the abstract interface of the join function.

B SAND: A Tool for Creating Semantic Descriptions of Tabular Sources

There is a large number of tables available on the Web and Open Data Portals. However, these tables come with various formats and vocabularies describing their content, thus making it difficult to use them. To address this problem, a semantic description of a table is created that encodes column types, relationships between columns, and additional context values needed to interpret the table. Given the semantic description, the table can be automatically converted to linked data or RDF triples providing the ability to quickly combine multiple tables for downstream tasks/applications to consume data using the same representation. Nevertheless, creating semantic descriptions is time-consuming. They are difficult to write manually, and during the creation process, the developers often need to switch back and forth between the ontology and the data to find the appropriate terms. Therefore, a user interface (UI) to assist and guide the developers in creating the descriptions is needed.

There are several UIs developed for creating semantic descriptions. Notable examples include Karma [21] and MantisTable [11]. Karma is a data integration tool that can ingest data from various types of sources (e.g., spreadsheets, databases, etc), map them to an ontology that users choose, and export the normalized data to RDF or store it in a database. MantisTable can import tables from files and create semantic descriptions to map them to KGs such as Wikidata. In both cases, these systems are strongly integrated with their semantic modeling algorithms for suggesting candidate descriptions and hence difficult to extend with state-of-the-art algorithms. In addition, when mapping tables to KGs, we often need to explore the data through filtering and

browsing entities to find rows that do not match the semantic descriptions, yet such features are lacking in the aforementioned systems.

In this appendix, we introduce SAND a novel system that addresses the above limitations. The contributions of SAND are: (1) a UI for creating semantic descriptions with a full pipeline from raw table data to RDF/JSON-LD output; (2) browsing/querying features for exploring the data and how it is modeled in KGs; and (3) an open design enabling easy integration with semantic modeling algorithms for semi-automatically creating semantic descriptions while supporting different knowledge graphs (e.g., Wikidata, DBPedia). SAND is available as an open source tool under the MIT License[§§].

B.1 Creating Semantic Descriptions in SAND

In this section, we will show how a user can build the semantic description of a table of mountains using the Wikidata ontology and then export the data as RDF. First, we discuss the overall process as if the user performs all steps manually. Then, we demonstrate how the user can do it semi-automatically.[¶¶]

Manual process. The process begins by uploading tables to SAND. SAND supports reading tables from various formats such as `.json`, `.csv`, `.xlsx`, and allows the user to refine the parsing options for different formats. The next step is to create the semantic description of the uploaded table. In particular, the user assigns semantic types to columns containing entities (called entity columns) and relationships (ontology predicates) between the entity columns and the remaining columns. Figure 7.7 shows the final outcome of this step. For example, the 2nd column *Name*

[§§] https://github.com/usc-isi-i2/sand

[¶¶] Demo: https://github.com/usc-isi-i2/sand/wiki/Demo

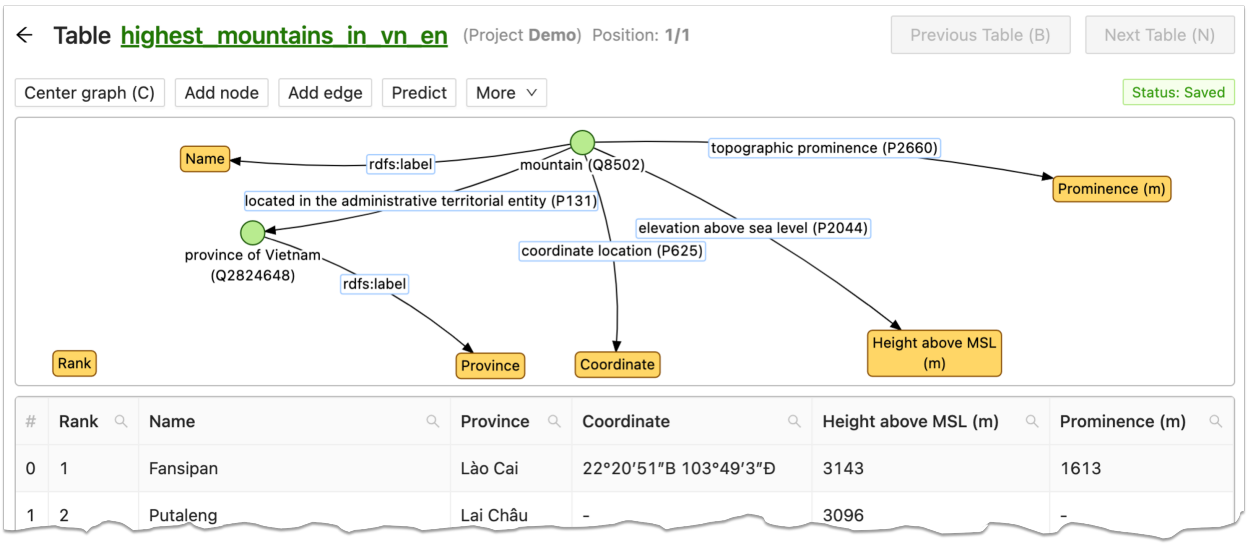


Figure 7.7: The semantic description of the table about mountains shown as a graph on top of the table. Green nodes are ontology classes, yellow nodes are columns. Edges depicts the types (e.g., `rdfs:label`) and relationships of columns (e.g., `located in administrative teritorial entity (P131)`).

is assigned to type `mountain (Q8502)` (via the link `mountain... → rdfs:label`) and the 3rd column *Province* is assigned to type `province of Vietnam (Q2824648)`. The description also tells us that these mountains are located in the provinces by the relationship `located in ... (P131)` between the two nodes: `mountain...` and `province...`. The third step is to link cells with existing entities in KGs (e.g., Wikidata). This step is optional, but we recommend doing it in SAND as during exporting, any cell in the entity columns that does not link to any entity is treated as a new entity. In this table, many entities are already in Wikidata so by doing entity linking, we can avoid creating duplicated data. Finally, once the user is satisfied with the semantic description, they can export and download the RDF of the table. Note that in the RDF data, raw values of the coordinates and heights are converted to the correct types thanks to the value constraints provided by the semantic description. Exporting the data via the UI may not be ideal if the user has a huge table or many tables of the same schema. Therefore, SAND also allows

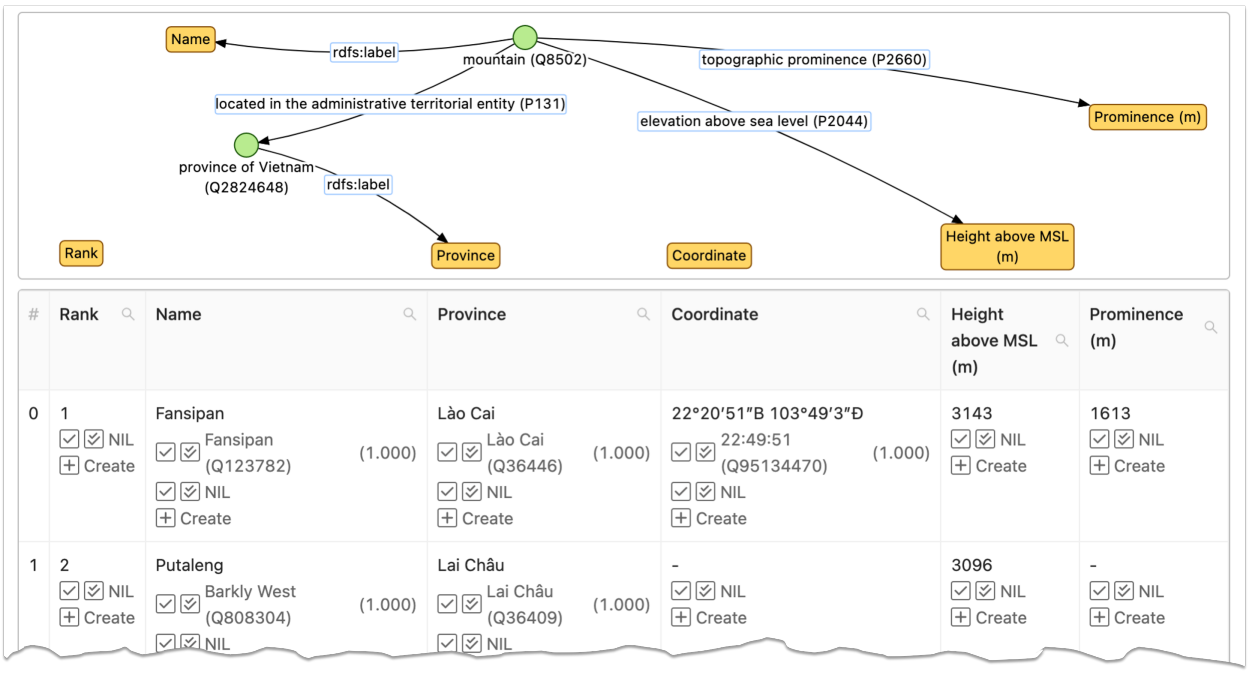
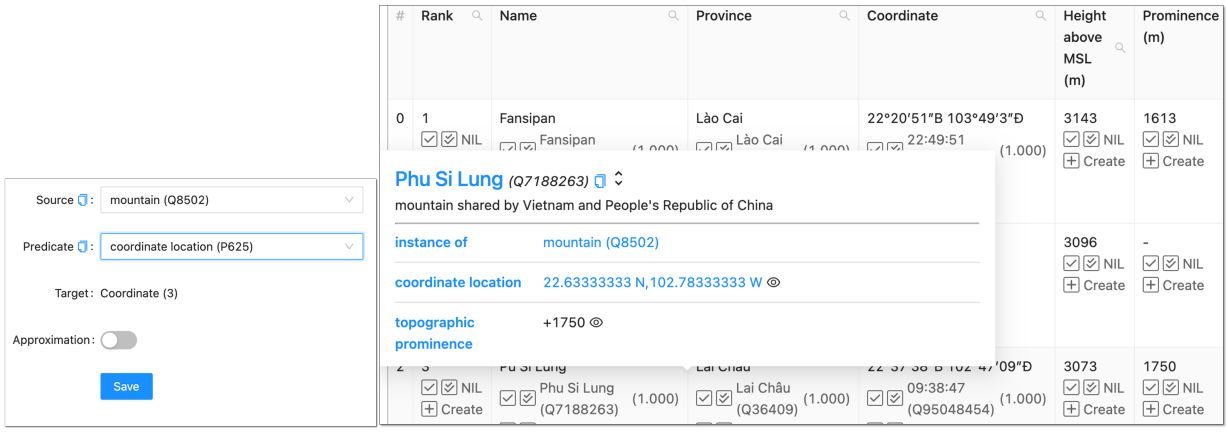


Figure 7.8: The auto-generated semantic description and linked entities of the table. If a cell is linked to an entity, it is shown as a hyperlink and its candidate entities are displayed below it. The user can update the linked entity of a cell by clicking on a single-check or a double-check button. Differing from the single-check button, the double-check button will apply the operation to all cells (same column) that have the same value as the current cell.

exporting an internal D-REPR specification of the table, which is used to transform the table to RDF by the D-REPR engine [69] programmatically.

Semi-automatic process. As the whole process is quite lengthy, hence at the beginning, the user can tell SAND to generate the semantic description and link entities for them by clicking the predict button. Under the hood, SAND will invoke semantic modeling and entity linking algorithms that are integrated to SAND via its plugin systems. Currently, SAND provides plugins for MTab [42] and GRAMS [67]. Figure 7.8 shows the result after prediction. Notice that it cannot predict the correct relationship of the 3rd column and some of the linked entities are incorrect. The user can fix the missing relationship by clicking on the yellow node representing the 3rd column, and selecting the source node and the relationship as shown in Figure 7.9a. Another



(a) Add/Update Relationship Dialog Box

(b) Entity Browsing

way to add the missing relationship is to click on the `add edge` button. To fix the incorrectly linked entities, the user can deselect them or right-click on the column and choose a filter to deselect cells containing entities that do not belong to the current semantic type (`mountain`) all at once.

Exploring the data. Assuming that the last column in the table is ambiguous and the user is unsure if the predicted relationship `topographic prominence (P2660)` is correct, they can choose to filter rows that have entities having the property P2660 and use SAND's entity browsing capability to check if any values of the properties are different from the values in the table (Figure 7.9b).

B.2 System Design and Configurations

To integrate with KGs, we define common schemas for entities, ontology classes, and ontology predicates. Then, we create their data access objects (DAOs), each of which has two functions: (1) querying an object by its ID or URI; and (2) searching the objects by their label. The DAOs also convert objects from their original schema in the KG to the common schema in SAND. This

approach makes it easy to set up SAND since DAOs can be implemented using KG's public APIs. In other words, we do not require to build a local database containing KG's entities and ontology.

To integrate with a semantic modeling algorithm, we declare an abstract class that predicts the semantic description of a given table and links table cells to entities in KGs.

The DAOs and semantic modeling algorithms are specified in SAND's configuration file by their paths (e.g., `plugins.wikidata.get_entity_dao`) and imported dynamically when the server starts.

B.3 Summary

With SAND, users can quickly model and convert their tables to RDF or JSON-LD for integrating with their dataset or knowledge graph. SAND reduces the amount of work that the users need to do via its semi-automatic semantic modeling, entity linking, and data cleaning features. For the researchers or developers, customizing SAND's components is straightforward via its plugin system. They can implement and test their semantic modeling algorithm on SAND or adapt SAND to their use cases without having to develop a new UI from scratch.

For future work, we plan to add more data source types such as databases and APIs. We also plan to support writing custom data cleaning scripts and incorporating data cleaning algorithms into the system.