



UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI INFORMATICA

DOCTORAL PROGRAM IN COMPUTER SCIENCE

**A semantic approach
for constructing knowledge graphs
extracted from tables**

Doctoral dissertation of:
Sara BONFITTO

Advisor:
Prof. Marco MESITI

The Chair of the Doctoral Program:
Prof. Roberto SASSI

A.A. 2021/2022 - Cycle XXXV

Contents

Introduction	1
1 Table understanding approaches	11
1.1 Preliminaries	12
1.1.1 Definition of “table” and issues in automatic processing	12
1.1.2 Classes of layouts	13
1.1.3 Generic tables vs other kinds of tables	15
1.1.4 Main ML classifiers used for table understanding	16
1.1.5 Link prediction methods	18
1.2 Approaches for the table understanding problem	20
1.2.1 The localization and segmentation steps	20
1.2.2 The functional and structural analysis steps	23
1.2.3 The interpretation step	29
1.3 Extracting and transforming tables	35
1.3.1 Basic extraction and transformation tools	35
1.3.2 Transformation tools to the relational/RDF models	37
1.3.3 Programming by example approaches	40
1.4 Knowledge graph construction	42
1.5 Concluding remarks	43
2 Background	45
2.1 Table representation and type system	45
2.1.1 Mixed types	46
2.1.2 Union types	47
2.2 Ontology and knowledge graphs	48
2.3 Semantic description	51

3	Semi-automatic type inference approach	54
3.1	Table identification	55
3.2	Type recognizers	56
3.3	Type inference approach	57
3.3.1	The main model	58
3.3.2	Training of the main model	60
3.4	Visualization and type adjustment	61
3.4.1	Main interfaces and error identification	62
3.4.2	Data type modification	63
3.4.3	Identification of a mixed type	66
3.4.4	Correlation between rows	70
3.5	Experimental results	74
3.5.1	Validation of real documents	74
3.5.2	Evaluation of usability - phase 1	77
3.6	Concluding remarks	80
4	Semantic descriptions of the table content	83
4.1	Overview of the Methodology	84
4.2	Graph embedding	86
4.3	Construction of the complete SD	88
4.3.1	Construction of the initial SD	88
4.3.2	Requirements for the complete SD	89
4.3.3	Generative algorithm of the complete SD	91
4.4	Weighting systems	96
4.4.1	Ontology-based weighting system	96
4.4.2	KG-based weighting systems	97
4.5	Inclusion of properties for unmatched table columns	99
4.6	Generation of the concise SD	101
4.7	Experimental evaluation	102
4.7.1	Validation of the GNN model	104
4.7.2	Validation of the concise SDs	107
4.7.3	Validation of the prediction of unmatched table columns	112
4.8	Concluding remarks	115
5	Visual management of the semantic description and KG construction	117
5.1	Interfaces for the management of SD	118
5.1.1	Graphical representation of SD	118
5.1.2	Overview of the main interface	119
5.1.3	Visual operations on the table columns	120

5.1.4	Visual Operations on the graphical representation of SD	122
5.2	Generation of the knowledge graph	125
5.2.1	Transformation functions	125
5.2.2	Algorithm for the KG construction	129
5.2.3	Interface for completing missing information	130
5.3	Evaluation of usability - phase 2	133
	Conclusions and Future Work	135
	References	137

Abstract

Knowledge graphs (KGs) are networks of real-world entities with their relationships and properties and are more and more used as a means for the integration of heterogeneous sources of information in a common model that facilitates the interoperability of different applications and generates a huge quantity of information that can be exploited for machine learning predictions.

Many approaches were proposed for the construction of KGs starting from tables extracted from spreadsheets, Web tables, or tables contained in digital documents, that entail the location and segmentation of the table in the source document, the extraction of its components, the identification of the function of different areas and the discrimination of relationships between attributes. However, the semantic characterization of the table content in terms of a domain ontology and the generation of the KG are still open research problems because of the heterogeneity of the table contents, the eventual presence of mistakes, and the lack of standardization.

The goal of this thesis is the development of an approach for supporting the user in the construction of a knowledge graph, which is compliant with a domain ontology, starting from tabular data presenting a complex structure and syntactic and semantic mistakes. We believe that a completely automatic approach that exploits sophisticated machine learning (ML) techniques cannot properly be used in this context. A semi-automatic approach can be devised in the process of data cleaning, semantic characterization of the table content, and translation in the KG representation. Users need to be supported by easy-to-use graphical interfaces for correcting mistakes and improving the system's overall performances.

For these reasons, in this thesis, we have devised a three phases approach. The first phase focuses on the semantic characterization of table columns in terms of the basic types and/or properties of a domain ontology. In this phase, a table is extracted from a spreadsheet and different cleaning activities are carried out (like removal of headers and footers, removal of blank and semi-blank rows, and detection of table rows that are correlated using a declarative pattern-based language). Then, through the identification of basic types of table columns, syntactic mistakes are identified and the user can correct them by exploiting different interfaces. This process improves the characterization of the semantic concepts contained in the table. The second phase of the approach focuses on the definition of a semantic description of the table content w.r.t. a domain ontology. The description is created starting from the result of the previous phase and exploits a graph neural network model for the identification of the relations that bind the concepts contained in the table. The third phase focuses on the generation of the triples of the KG starting from the table content and the semantic description. In this activity, we have developed interfaces for the identification of semantic mistakes occurring in the data and for the specification of identifiers of the KG instances. Different experiments have been conducted for validating the three phases of the approach proposed in the thesis and the usability of the entire system.

Introduction

Knowledge graphs (KGs) [93] allow the description of real-world entities, their relationships and properties in terms of nodes and edges of heterogeneous graphs. A KG is usually represented through Resource Description Framework (RDF) triples [102] which indicate that an entity *subject*, usually identified by an IRI (International Resource Identifier), is connected through a relation/property to an *object*. The object can be an entity (i.e. an IRI), or a basic property of the subject.

Knowledge graphs can be exploited in different application contexts (recommendation, querying, classification, knowledge augmentation, and data exchange [178]). They are becoming more and more used because the graph data model is more flexible than the relational model and facilitates the integration of different information and thus the interoperability of the applications working with them. The manual construction of KGs is considered very expensive and many automatic and semi-automatic approaches have been proposed that rely on the use of KGs (e.g. DBpedia, Yago, Freebase) automatically extracted from the Web and covering different domains. The use of an ontology for constraining the relationships that can exist among the classes of a given domain and their properties is fundamental to enabling high-precision knowledge acquisition processes [168], reducing the introduction of noises in the KGs, and facilitating the interoperability of the applications working with the data.

One of the main sources for the generation of the KG content is tabular data. Tabular data can be identified in pdf files, spreadsheets, and Web pages, and may be coded in different formats (like XML, JSON, latex, CSV, TSV,...). These tables, mainly designed for being interpreted by humans, are heterogeneous and do not follow any standard format or notation. Some files present values that are organized as a relational table where column values are homogeneous, while others are highly heterogeneous and the same column contains different types of information at different granularity levels. Moreover, a table can contain errors that make it harder the identification of a column type and introduce inconsistencies.

The automated processing and/or conversion of a table to other formats, principally aimed

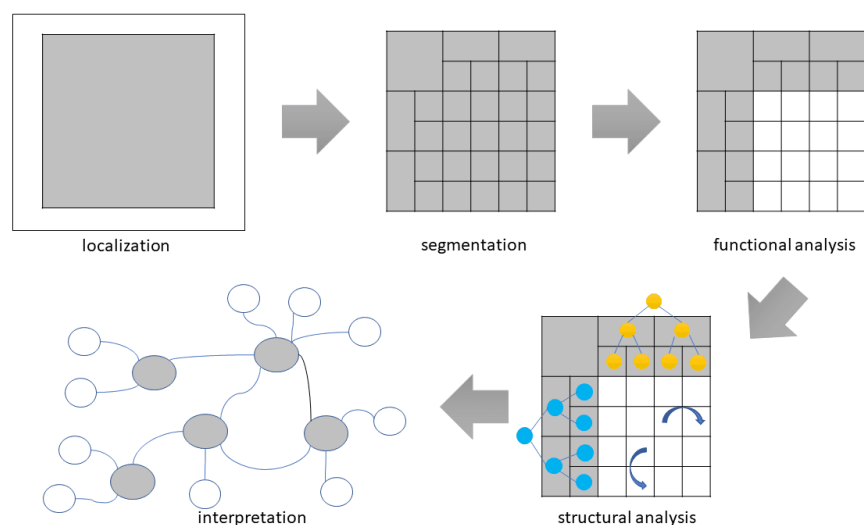


Figure 1: The Table Understanding Approach

at extracting meaningful information, storing and transmitting structured information (e.g. knowledge graphs or relational models), and integrating data from different sources are open problems, that fall under the umbrella of *Table Understanding* problems. According to [92], approaches aimed at obtaining an end-to-end solution generating machine-readable and structured information from tables should encompass the following five logical steps (that are also depicted in Figure 1): *localization*, to identify the location and structure of the table in between all the other elements in the file/page; *segmentation*, to extract all table components, i.e. cells, columns and rows, headers, stubs; *functional analysis*, to identify areas of the table that have a similar function; *structural analysis*, to distinguish attributes relationships; *interpretation*, to extract the semantic meaning of the table content.

The *localization* and *segmentation* steps were, in the nineties, mainly addressed by image processing techniques [71] that were originally used for table detection in documents, since tables were often found in scanned images of documents. However, with the widespread adoption of HTML and other document formats, the table localization step evolved toward the usage of specific tags/commands to declare table contents. These new techniques [165, 47, 48, 153, 66] focused on distinguishing between "layout tables", which were used for formatting purposes, and "real tables", which contained actual data. The emergence of deep learning moved the localization and segmentation techniques towards the usage of convolutional neural networks (CNNs) [114, 42, 110, 81, 69] and conditional random fields (CRFs) [136] that can be particularly effective.

The *functional analysis* step is often preceded by a pre-processing step, aimed at iden-

tifying the table reading order (vertical or horizontal), by generally exploiting layout (style and format) and/or content coherency characteristics. Once headers/data cells have been discriminated and the table reading order has been identified, the *structural analysis* step aims at understanding the relationships between headers and data cells, that is, finding the data cell regions/blocks that each header cell describes and refers to. The structural analysis identifies the table class layout and the occurrence of hierarchies on headers and stubs by aggregating data cells of the same header and eventually identifying parent-child relationships existing on headers and stubs. Some approaches use a set of rules on the table's boundaries, layout, and content features (e.g. [149, 66, 35]), while others use machine learning techniques to infer rules from training sets (e.g. [37, 104, 38, 70]). Some methods view the identification of these relationships as a graph partitioning problem and adopt CRFs [136], and support vector machines [45] approaches.

The *interpretation* step assigns a semantic description to the content of the table, which is a difficult task, even when the previous localization, segmentation, functional and structural analysis steps have been correctly carried out. Initial approaches (e.g. [92]) proposed the relational model for the interpretation of the table content. Newer approaches (e.g. [116, 127, 14, 40]) are considering the use of knowledge graphs expressed in terms of a domain ontology. The KG construction process entails the following tasks: *i) column type inference*, i.e. the identification of column types or ontological classes associated with the table columns; *ii) relation discovery*, i.e. the uncovering of relationships among the identified ontological classes; *iii) triples generation*, i.e. the creation of RDF triples from the table rows. This last operation is not trivial because it depends on the table content. Entity identifiers might not be present in the table as well as mandatory properties used for identifying the entities. In these cases, user intervention (or the use of data/knowledge augmentation techniques) is required to complete the missing data and obtain accurate descriptions of real world-entities.

For the *column type inference* step, many approaches have been proposed for annotating the table columns with simple basic types (e.g. [97, 157, 72, 158, 31]) or classes and properties of an ontology (e.g. [134, 145, 34, 89, 176, 98]). These approaches use neural networks to make predictions on the column types relying on the use of knowledge bases. The *relation discovery* step entails the identification of direct and indirect relations between entities, and it can be seen as a link prediction problem. The most popular link prediction approaches employ structural information, such as node similarity and centrality measures, to yield the prediction [117]; while more recent approaches rely on graph neural networks to improve prediction performances [111]. These last solutions represent a graph in a lower dimensional vector space, whilst maximally preserving properties like graph topology and ancillary data [75]. Only recently, the possibility of identifying relations passing through

other entities has been considered [154, 65, 163] (e.g. two actors reported in a table can be related by means of the film in which they have played). In these approaches, a graph model reporting all possible plausible relations involving the concepts identified in the table is generated. Then, the edges are weighted (according to their frequencies in previously processed tables) and a tree with minimal weight is extracted that better represents the relations among the table columns. Finally, for the last step *triples generation*, different kinds of declarative mapping rules and their associated engines can be exploited for the acquisition, transformation and normalization of the data to be included in the knowledge graph (e.g. R2RML [50] for relational data, RML [54], SPARQL-Generate [115], and YARRRML [82] for dealing with other formats and the heterogeneity of the data sources). However, these approaches are mainly devised for the extraction of a single entity from a table and require a high level of homogeneity of the table content (which is not guaranteed in our context). Moreover, the compliance of the generated triples to the adopted domain ontology is not assured. Once the triples are generated, techniques for knowledge fusion and knowledge refinement [41] should be applied for improving the quality of the knowledge representation and reduce the noise that can occur in the KGs. Another issue related to KG management is the possibility of inferring new relationships on the represented entities and different surveys face this problem [168, 93, 91].

Problem formulation and resolution approach

The purpose of this thesis is the development of an approach for supporting the user in the incremental construction of a consolidated KG compliant with a domain ontology starting from multiple tabular data. These tabular data are transformed into small KGs that can be used for feeding the consolidated KG.

Tabular data are extracted from spreadsheets that do not follow a standard representation format or notation. Moreover, a single column can contain data of different types (e.g. the name of a company or the name of a person), or a cell can contain different pieces of information (e.g. a string containing the different components of an address). In addition, syntactic errors (e.g. a date written without the separators) or semantic errors (e.g. a zip code associated with the wrong town) can occur in the table. Furthermore, a table can contain headers, footers, blank rows and rows containing only a few data (e.g. totals) that are not useful for the generation of a knowledge graph, and that must be identified and removed. Lastly, information related to a single real-world entity can be organized into consecutive rows that need to be identified as correlated rows and properly handled.

An ontology is a formal description of the elements involved in a given domain and the relationships among them. By providing a common set of terms used for describing elements

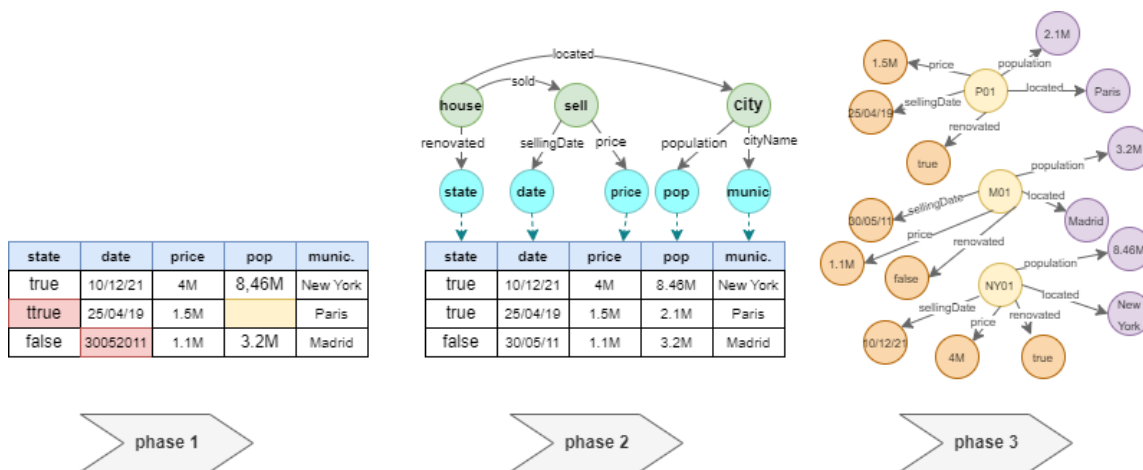


Figure 2: Pipeline of the three phases of the approach

and relationships, an ontology guarantees the interoperability of applications working with data that follow its rules and constraints. In our context, an ontology consists of concepts organized according to inheritance relationships. Moreover, relations can be devised among concepts and basic properties can be associated with concepts. A set of identifying properties can be specified for each concept that can be exploited to specify the identifier of the concept instances. The use of an ontology in the creation of a knowledge graph from data occurring in different data sources allows the generation of applications for the analysis of data that can safely work with homogeneous data and provide meaningful prediction results. Instances of the ontology are represented by means of a consolidated knowledge graph containing all the data previously processed and annotated according to the considered domain ontology.

The table understanding approaches proposed in the literature are not suitable for our problem because they do not consider the variability of data types that can occur within the same cell or column, they do not check for the existence of correlations, and they rarely consider the presence of syntactic and semantic errors. Moreover, it is quite difficult to infer the relationships existing among the concepts identified in a table. Finally, the generation of a knowledge graph requires the existence of an identifier, usually determined by a set of identifying properties, for each new concept that can be found. However, sometimes the identifying properties can be missing from the table. Therefore, we proposed a semi-automatic approach to support the user during the process of data acquisition, cleaning, transformation and semantic characterization. Users are supported by easy-to-use graphical interfaces for correcting mistakes and improving the overall performance of the system.

As outlined in Figure 2, our approach is structured in three phases that combine the use of machine learning techniques with graphical user interfaces in a complex web application. Each phase is dedicated to the management of a particular problem and relies on the output of the previous phase. This approach is semi-automatic, which means that user interactions are often required for solving errors and correcting wrong predictions.

The main purpose of our first phase is table identification and cleaning. Table identification requires pointing out the table boundaries in a spreadsheet file, and removing headers, footers, blank rows and columns. Starting from the identified table, a set of data types for each cell and column is determined through a multi-label classification approach that uses a decision tree and a set of functions for type recognition. The decision tree is trained by means of synthetic data generated for a specific domain that compensates for the lack of real data. The detection of data types is also useful for the identification of syntax errors (i.e. data whose type is not compliant with the type established for the column) that can be fixed by the user through a set of graphical user interfaces that allow the application of a single correction to many rows at the same time. And finally, in this phase, for identifying correlations among table rows, we adopt a declarative pattern-based language for specifying when a correlation exists and we introduce specific user interfaces for correcting and changing the correlation among table rows.

The second phase of the approach is related to the semantic characterization of table content. In this phase, a semantic description of the spreadsheet tables is provided by means of annotations w.r.t. the considered domain ontology. The semantic description is created starting from the concepts identified in the first phase and allows the prediction of the relationships existing among them by taking into account a full heterogeneous GNN model [164]. The model uses two convolutional layers with a graph attention mechanism [161] for computing the embeddings. Each relation name $r \in R$ has its attention mechanism and the node embeddings are obtained as the sum of the contributions of each convolution defined on the relations in which it is involved. The model is constructed on the consolidated KG. One of the main advantages of our approach is the possibility of identifying also indirect relationships among concepts, that is relationships that pass through other concepts that are admitted by the domain ontology. The model is also used for inferring properties on unmatched columns, that is columns without a semantic characterization that must be associated with a concept and property of the considered ontology and inserted in the final semantic description obtained for the table. The semantic description is also coupled with a graphical representation that makes it easier for the user to check the automatically generated model and correct mistakes when needed.

Once the semantic description is complete, it can be used for the automatic translation of the spreadsheet table in an RDF representation according to the considered ontology. In this

phase, we provide facilities for the definition of identifiers for each concept of the semantic description. Specifically, we provide the possibility of associating transformation functions with ontology concepts that can be applied to the identifying properties for the generation of an identifier. Moreover, we propose the use of a graphical language for the specification of the transformation functions that would simplify the user activity. Then, an algorithm that transforms the data contained in the table into an RDF graph is applied for creating a knowledge graph in RDF that represents the table content according to the concepts, relationships and properties identified in the semantic description.

Several experiments have been conducted for assessing the quality of the developed approach. The type recognition algorithm of the first phase has been trained and tested using one million synthetic tables presenting different kinds of syntactic errors. The dataset has been used also for comparing the decision tree with other methods such as Random Forests and MLPs. The average AUROC and AUPRC scores of each method have been computed and they did not show any statistically significant difference, therefore, since the decision tree is the fastest to train and the most interpretable, we have chosen to use it. The usability of the interfaces developed for the first phase have being tested through 20 volunteer students with a set of assigned tasks. The test was successful, with the majority of users (95%) satisfied with the developed interfaces and reporting that they did not encounter any problems during the error correction process. The prediction algorithm developed in the second phase has been compared with two baseline methods: the multimodal approach MRGCN [169] that extends the basic R-GCN model [146], and SeMi [65], which exploit an R-GCN model [146] for the generation of concise *SD*. The obtained AUROC scores show that our method outperforms the other methods for all the used datasets. Moreover, we have evaluated the quality of the produced results by considering manually generated ground truth mappings and we obtained an average accuracy of 80%. Finally, we tested the usability of the interfaces developed for the second phase and we obtained that 85% of the users defined the interface as easy-to-use and intuitive.

Research contributions

The work proposed in this thesis has been initially conceived in the context of a research project with a debt collection agency that needed to collect invoices represented as CSV/XLSX files from different local authorities (e.g. Municipalities, Provinces, and Regions) and integrate them into a KG for proceeding in the rescue of the credit and correctly classifying the debtors as good or bad payers. The work started by studying the literature in the context of table understanding and schema integration and concluded with the publication of a survey on table understanding problems and approaches for extracting tabular data from spreadsheets and assigning an interpretation useful for their integration into a KG [20].

Then, the work was focused on the realization of the first phase of our approach. Specifically, we have worked on the adoption of decision trees on an ensemble of type recognizers for the identification of cell types and column types by taking into account the presence of mistakes in the cell values. Even if the performances of the developed model were particularly good, the need to develop a system in a commercial environment pushed us towards the development of different web interfaces for supporting the user in fixing the predicted types and also in fixing and completing the type assignment. In this activity, we have also considered the use of a domain ontology for a better characterization of the table content. Moreover, we have developed a graphical environment by means of which the user is able to operate on the graphical representation of the semantic description and improve its quality; then, we conducted usability tests for assessing the facility of its use. The key results of this research phase are the development of a multi-label classification approach for dealing with the heterogeneous content of the tables, and the coupling of the predictive approach with user interfaces for supporting the user in the verification and update of the prediction. This is an important contribution to managing the heterogeneity of the table contents and identifying possible errors to be fixed by the user. The obtained results appeared in [18, 19] and were discussed at the DAFSAA Ph.D. consortium [17].

For the realization of the second part of the approach, we have studied deep learning models and the possibility of their adoption in the context of table understanding for the identification of plausible relationships existing among the concepts extracted in the first phase. The problem required a lot of work for the identification of the machine learning model that is more suitable for this purpose and for tuning the parameters for improving the performances. The key result of this research phase is the integration of a machine learning method in the prediction of the kind of relationships that bind together concepts presented in a table with the formal characterization of the requirements that a semantic description should have. The use of embedding techniques for the presentation of the consolidated knowledge graph allows taking into consideration different characteristics of the nodes among which the relationship should be identified. Therefore, there is an improvement in the state-of-the-art approaches that mainly rely on the relations existing among the specific nodes (so they adopt the node identifiers). Moreover, the extensive experimental analysis proved the quality of the obtained results with respect to other approaches developed in the same context. The results of this phase have been submitted for publication [22].

The last part of our work was devoted to the realization of the approach for translating the table content in terms of the knowledge graph. In this context, we have proposed the use of a graphical language for the representation of semantic description that can be visualized, checked and updated by the user in different ways. Moreover, we have proposed a graphical language also for the specification of the instance identifier relying on the identifying prop-

erties occurring in the domain ontology [24]. We are also currently working on the adoption of the proposed methodology for the construction of biomedical knowledge graphs in the context of the National Center for Gene Therapy and Drugs based on RNA Technology, financed under a grant from the Italian PNRR [21].

Last, but not least result of this PhD thesis is a single web platform that integrates the different approaches proposed in the thesis. We are currently completing the implementation of the last features described.

Structure of the thesis

The thesis is organized into 5 chapters.

Chapter 1 provides an overview of the research efforts for converting spreadsheet data into useful information, that come under the umbrella of table understanding approaches. The chapter focuses also on techniques for annotating the information extracted from tabular data with semantics and constructing semantic models of spreadsheets. Finally, the most recent methodologies are discussed and future research directions are outlined.

Chapter 2 reports the background information about table extraction, type system, ontology and knowledge graphs, and semantic description that are used in the thesis. The formal definitions of these concepts are introduced along with examples for a better understanding of their meanings.

Chapter 3 presents our approach for cleaning and removing errors from a table and identifying a data type for each column. The used ML technique is presented along with the performed tests. We also describe the set of graphical user interfaces developed for supporting the users in checking and fixing the automatically generated annotations. Furthermore, experiments for the validation of the ML techniques and the obtained results are presented. At the end of the chapter, usability tests are presented to evaluate the user experience of the various graphical user interfaces developed in this phase of our work.

Chapter 4 introduces a new approach for generating a Semantic Description (SD) from the content of a table. First, the concept of complete SD is introduced along with the algorithm for its creation. Then, different weighting systems are introduced for properly weight the complete SD . Finally, by exploiting the Steiner tree a specific semantic is extracted in the form of a concise SD . Finally, a discussion of the performed experiments and comparisons with other works are presented.

Finally, Chapter 5 presents the graphical representation of the semantic description used for facilitating the user in checking its correctness. The interfaces used for the verification and

modification of the semantic description are then presented. Moreover, the issues and the approach for the generation of a knowledge graph are discussed along with the algorithm used for the transformation. Furthermore, transformation functions for the generation of identifiers for the KG instances are described. Finally, we report the results of the usability test performed on the presented interfaces.

Chapter 1

Table understanding approaches

The purpose of this chapter is to provide a comprehensive analysis of the research efforts so far devoted to the development of an automatized system for the automatic transformation of spreadsheet data into meaningful information, also known as the table understanding problem. The different methods will be classified according to Hurst's table-understanding steps and the exploited approaches and methods (ML or heuristics). Furthermore, to allow a more exhaustive comparison, for each method, the treated table types will be evidenced in terms of input format, table layout, and table hierarchy, together with the (private or public) availability of datasets used for testing and development. Special emphasis will be given to the information extracted from tabular data, and to the techniques adopted for the semantic annotation of spreadsheets and the construction of semantic models. Since the most interesting table recognition literature works published up to 2006 have been already described in [77, 119, 120, 92, 174, 59, 48], in our survey we aim at describing the main works of the last fifteen years. The last part of the chapter is dedicated to the construction of knowledge graphs. Even if many of the techniques discussed in this chapter can be profitably applied for their generation, in this part we wish to empathize the issues that arise in data integration and the approaches for facing them.

The chapter has been realized starting from [20] and is organized as follows. Section 1.1 introduces some preliminary notions and notations that are used in the chapter. Section 1.2 discusses the approaches so far proposed according to the five logical steps of table understanding. Section 1.3 discusses approaches for extracting and transforming tables, whereas Section 1.4 deals with the concept of knowledge graphs and the approaches for their construction. A concluding discussion on the presented methodologies is reported along with some future research directions.

1.1 Preliminaries

Different kinds of documents (pdfs, web pages, spreadsheets, etc.) can contain tables with different characteristics (in terms of, e.g., the way they are coded in the document, their layout, their content, and so on). In this work, we overview both approaches that can be applied to *generic tables*, that is tables contained in any kind of document, and approaches developed for handling *specific table types* (e.g. relational tables, web tables, spreadsheet tables). Section 1.1.1 presents the definition of table and the issues that need to be faced when processing generic tables. Then, Section 1.1.2 outlines the classes that can be identified in table layouts and Section 1.1.3 provides a comparison among generic tables and tables of the relational model, web tables and spreadsheet tables. Section 1.1.4 introduces the main ML classifiers used for the approaches discussed in the chapter, whereas Section 1.1.5 discusses the issue of link prediction in graphs.

1.1.1 Definition of “table” and issues in automatic processing

Identifying a definition for the concept of table is quite hard because of the variety of organizations, access methods, and meanings that can be associated with the information delimited in a grid-based structure. According to [133] “tables have a regular repetitive structure along one axis so that the data type is determined either by the horizontal or vertical indices”. This definition considers a grid-based organization, useful for expressing two kinds of information present in a table: the information that the authors wish to report (*data content*) and the indexes used for interpreting and accessing the information (*access key/index*). Conversely, in [92] a table is presented as “a device used to present information to the reader by organizing some set of meaningful elements on the page so that the relationships between those elements, and the manner in which combinations of the elements interact, is demonstrated to the reader”. This definition highlights the presence of relationships between the table elements that need to be considered to correctly interpret the meaning of its content.

The aforementioned definitions, however, do not explicit the great amount of ambiguity hidden in the tabular organization of information, which should be considered for the correct automatic processing of tables. Moreover, these definitions do not give enough relevance to the context in which tables are embedded and to the assumption made by the tables’ authors during their creation, which would allow associating the correct semantics to the table content. These aspects are really relevant in providing a semantic characterization of the information contained in tables and in posing the bases for the integration of their contents with other kinds of information, as well as for posing and answering queries.

For grasping the problem, consider the simple table in Figure 1.1(a). Even if the structure and the content suggest that the table contains the information of two individuals, the labels

name	city	date
Jack	Milan	1/1/2000
Alice	Rome	5/2/2010

(a)

9	age	children
12	age	
7	address	Terry

(b)

children	age	9
	age	12
address	7	Terry Dr.

(c)

Figure 1.1: Examples of generic tables

city and date associated with the second and third columns do not permit interpreting the meaning of their contents. Does city represent the place where these people live or where they were born, or what else? The same considerations hold for the date column. Contextual information is needed for assigning a precise meaning to their content.

Even if this problem can be considered trivial, the situation can be further complicated as illustrated by the table in Figure 1.1(b). Indeed, this table complies with both Peterman’s and Hurst’s definitions, but the lack of constraints on the location of the data and access cells makes it hard to: *i*) identify the table meaning and visually comprehend both the content and the logical organization of the real world objects that the table represents; *ii*) bind the table cells to the real world objects to which the cells themselves are associated. The re-organization of the table in Figure 1.1(b) as reported in Figure 1.1(c) would simplify the aforementioned binding. Anyhow, it is clear that a full understanding of each of the aforementioned tables needs the specification of a *context*, in which the table should be interpreted, and the *external knowledge* to associate a precise meaning to the table contents.

Figure 1.2 proposes the general structure of a table as a grid of *cells*. Cells can have different sizes and can be used for representing access indexes to the table data content. The *stub* is located in the left-hand part of a table and used for indexing the content area (especially in matrix tables). The *head* is located in the uppermost part of the grid and used, like the stub, to index the content area. Both the head and the stub can be organized hierarchically and thus provide the basis for aggregating the content data at different layers. Therefore, row/column headers can be nested at different depth. Blocks of cells containing related information can be identified. The content of some cells may depend on the value of other cells, as for values in column total. The dependency can be explicitly expressed by the presence of formulas in spreadsheets or can be implicitly specified and needs to be extracted by considering the values occurring in the table.

1.1.2 Classes of layouts

Starting from this general organization of the table and following the notation presented in [137], three major layout classes of tables can be usually identified: 1-Dimensional (1D), 2-Dimensional (2D), and Complex (C) tables. In 1D tables either the header or the stub is

Medal table								
First 5 countries with the most Olympic medals								
countries		summer games			winter games			total
		gold	silver	bronze	gold	silver	bronze	
United States	females	509	400	374	53	70	44	1450
	males	513	395	332	52	42	44	1378
Russia	females	200	190	150	30	30	29	629
	males	195	129	146	48	27	30	575
Great Britain	females	163	125	147	6	1	5	447
	males	200	120	146	5	3	12	486
Germany	females	91	95	115	40	44	29	414
	males	100	99	115	47	44	31	436

Figure 1.2: General organization of a table

present. Specifically, Vertical 1D tables (1D-V) are made of a box head generally placed at the top of the table, miss the left/right stub, and contain at least one-body line with data content. The box-head hierarchy is usually flat; however, if it is composed of nested headers, their labels are related to each other. The rows in the data content area are instances of the header cells above (the table in Figure 1.1a is an example of the 1D-V table). Horizontal 1D tables (1D-H) are the rotated version of vertical 1D tables; in this case, the box-head is missing, and the (eventually hierarchical) stub contains the headers to which all the data cells in the column refer. In 2D tables, both stub and head are present and are usually organized in hierarchies (see Figure 1.2 for an example). The table understanding problem in this kind of tables is harder because it is quite difficult to discriminate among the cells that contain stubs and heads from those with the content. The class of C tables encompasses all the other kinds of tables that are based on a head/stub organization, but whose shape is not so easily separable. Among them, we can mention multiple tables that are embedded in a single table, tables whose cells are aggregations of other cells because of the presence of formulas or because their value is, for example, the sum of the values of other cells, tables whose entries expand over multiple cells (in different rows/columns). These complex tables are challenging for automatic approaches and no completely automatic approach has been proposed yet. Figure 1.3 shows an example of a complex table. As pointed out by [137], discovering and handling this class of tables is hard as it is difficult for a system without any prior knowledge to discriminate between headers/stubs and data contents.

Group	N	%			Comparisons		
		positive	neutral	negative	χ^2	df	p
Q1 (a)	Programming Labs are useful for learning the subject						
CS group	400	73	7	20	0.9	4	ns
Math group	400	70	8	22			
Q1 (b)	Working with data makes predictions more reliable						
CS group	400	52	12	36	13.7	4	< 0.05
Math group	400	44	15	41			
Q1 (c)	Lessons in class are more active than web conferences						
CS group	400	46	11	43	14.8	4	< 0.01
Math group	400	31	14	55			

Figure 1.3: A complex table with the results of a student satisfaction surveys on lesson organization. The table reports questions, answers by different groups (with the number of students per group) and the p-value (p) resulting from a χ^2 test of significance with df degrees of freedom.

1.1.3 Generic tables vs other kinds of tables

When treating tables, it is important to clarify correspondences and differences between generic tables, relational tables, web tables, and spreadsheet tables. Relational tables follow the constraints for the representation of structured data [44]. Given n basic domains $D_i, 1 \leq i \leq n$, a relation R is a subset of their Cartesian product ($R \subseteq D_1 \times \dots \times D_n$). The schema S of a relation R is a set of labels $\{A_1, \dots, A_n\}$ (denoted attribute names) associated with their corresponding domains ($S(R) = \{(A_1, D_1), \dots, (A_n, D_n)\}$) that along with the name of the relation is used for identifying the context in which the data should be interpreted. The term *relational tables* is used to point out their usual tabular representation.

According to this definition, the relational model imposes a fixed, pre-defined schema for relational tables, where all values specified for a column should belong to the same basic domain. This restriction is not imposed in a generic table because its content is generally provided for summarizing a given concept. Moreover, cells can contain heterogeneous contents and may have a heterogeneous structure. Content heterogeneity means that a cell may contain different basic information units (e.g. a cell can contain the income per month of an employee as well as the percentage of the income in the entire year and the increment with respect to the previous year) and structure heterogeneity means that columns may contain more than one information; as an example, information units can be embedded in patterns that make easier their interpretation for readers (e.g. 1000 euro (9%-+2%)). Finally, a single tuple of a relational table represents a single real word entity, therefore the

attributes are aspects of a real word entity that are bound together for its description in a given domain. This is not the case in generic tables where the interpretation of one row depends on its layout, and attributes of the same real-world entity can occur in different table blocks. According to [92], a generic table corresponds to a collection of views applied simultaneously on a set of relational tables. Therefore, different kinds of information may be contained in the table and their specific organization is based on the will of their authors to clarify a given concept and make its interpretation easier for readers.

The term *webtable* is introduced in [30] for representing quasi-relational tables. These tables contain structured data describing a set of entities, and are thus useful for data search, table augmentation, KB construction, and various NLP (*Natural Language Processing*) tasks. However, they do not need to follow the strict constraints of the relational tables (no need to have a key, no need to have attribute names associated with columns, and so on). According to our notation, web tables can be considered as generic 1D tables in which the structure weakly follows the one prescribed by relational tables.

Spreadsheets have been specifically developed for easily creating and managing tables with any layout. Their organization in different sheets allows the creation of several tables both in the same and across several sheets. Moreover, formulas can be included for computing values relying on the values of other cells belonging to the same table/sheet or other tables/sheets. Finally, several formatting artefacts, textual metadata, and floating objects (e.g., pictures and charts) can be included in a spreadsheet, which makes their automatic processing particularly complicated.

1.1.4 Main ML classifiers used for table understanding

Many proposed approaches exploit “supervised” classifiers that rely on different algorithms for inferring the best classification rules from labelled (training) examples, by minimizing the loss between the (manually signed) “ground truth labels” of the training samples and their predicted labels. Among the various classifiers, those mainly used for table recognition are Support Vector Machines – SVMs [45], Decision Trees – DTs [27, 118], Random Forests – RFs [26], C4.5 [138], probabilistic Naïve Bayes (NB) classifiers [80] and Conditional Random Fields – CRFs [112]. More recent works exploit different DL models [114, 123] given their documented capability of reaching superhuman performance in many areas.

Classification DTs or C4.5 classifiers are built by recursively identifying the most accurate rule that splits the training set into subsets by minimizing the classification error [148]. The difference between DTs and C4.5 classifiers relies on the way the splitting rule is chosen; DTs use the Gini index [68], which minimizes the node impurity, while C4.5 classifiers choose the rule maximizing the information Gain [138], which is calculated by comparing the entropy

of the dataset before and after the split. RFs [26] are ensembles of DTs or C4.5 classifiers, which are formed by training several DTs, generally on bootstrapped samples; the final prediction is given by composing the labels predicted by each tree through majority voting. SVMs [45] are binary classifiers that find the support vectors (i.e. sample in the training set) that delimit the decision boundary (hyper-space in case of multi-dimensional points) that best separates the negative and the positive samples, after their projection in a linear, polynomial, or gaussian hyperspace [83].

NB classifiers ground their prediction on the Bayes theorem and, under the assumption of a strong (naïve) independence between features, approximate it by: $p(C_k|x_1, x_2, \dots, x_n) = \frac{1}{Z}p(C_k) \prod_{i=1}^n p(x_i|C_k)$ where $Z = p(\mathbf{x})$ is a scaling constant factor. If the number of different classes is K , NB classifiers exploit the training set to estimate the parameters θ_k of the distribution $f(x_i, \theta_k)$ (e.g. gaussian, multinomial, Bernoulli) that best approximate the class-conditional marginal densities, $p(x_i|C_k)$. By using $f(x_i, \theta_k)$, the predicted class C_k^{pred} of an unseen sample $\mathbf{x} = x_1, \dots, x_n$ is the *maximum a posteriori* (MAP) estimate: $C_k^{pred} = \underset{k=1, \dots, K}{\operatorname{argmax}} p(C_k) \prod_{i=1}^n f_k(x_i, \theta_k)$.

CRFs extend NB classifiers to predict the class (tag), $C_k \in \mathbf{C} = C_1, \dots, C_K$, of the elements in a data sequence, where the dependence between consecutive samples may be exploited to increase the amount of information available for prediction. In more detail, to use contextual information from previous samples and capture nonlinear relationships between each sample and its neighbours in the sequence, the element at the t^{th} position in the sequence is coded through a vector \mathbf{x}_t obtained by concatenating the sample feature-vector to its *conjunction* with feature-vectors from neighbours (conjunction is generally realized by multiplying the vectors). Then, a set of J feature functions $f_j(\mathbf{x}_t, C_k, C_{k-1}, t)$ are defined on the training set, to describe the coded sample, its position t in the data sequence, its relationship with each tag $C_k \in \mathbf{C}$ and the relationship between consecutive tags C_k and C_{k-1} . Under this construction, CRFs find the class C_k maximizing the posterior conditional probability $P(C_1, \dots, C_K|\mathbf{x}_t)$, expressed as: $P(\mathbf{C}|\mathbf{x}_t) = \frac{1}{Z} \exp\left(\sum_{k=1}^K \sum_{i=1}^J \lambda_j f_j(\mathbf{x}_t, C_k, C_{k-1}, t)\right)$ where Z is a normalization factor over all sequences. While the feature functions are predefined on the training set, the training phase of CRFs estimates the best values of parameters λ_j ($j = 1, \dots, J$) through gradient descent.

Given that the table cells may be viewed as pixels in an image, and that the relationship between table entities should be considered when mapping/matching the table cells into an ontology or KB, in the context of table understanding the mostly used DL models are Convolutional Neural Networks (CNN), which are composed by many layers of neurons (hence the term "deep"), organized into consecutive 2D grids (the layers) and connected by weighted edges. Such connections are local, meaning that each neuron in layer l is

generally connected to a subset of neurons (neighbourhood) in the preceding $l-1$ layer. The network works by propagating the input signal (e.g. a table/column coded through a vector) through convolutional layers (feed-forward propagation), which, on their side, process the incoming values through weighted, nonlinear convolutions and then send the computed values to the next layers. By interlacing convolutional layers with dropout layers, which randomly drop some neurons during training, (batch) normalization layers, average/max-pooling layers, which reduce the signal size by taking only the average/maximum values in each neighbourhood, and flattening layers, which transform the signal from 2D to 1D, the signal complexity is increased, combined, and integrated as it proceeds through the network so that its “intelligent” reduction in size and dimension ultimately brings to a prediction in the output layer. Since the network weights are the actors of the information process, and their values are randomly initialized when the network is created, the learning process practically uses a back-propagation of the errors on the training set to gradually fix the network weights in order to minimize a loss function. The advantage of using DL models relies on their ability to learn highly non-linear relationships from examples; moreover, the capability of learning from examples by ensuring generalization further avoids overfitting the examples and allows producing optimal results on unknown samples.

All the approaches that we discuss in the chapter evaluate their performance by exploiting the well-known $precision = TP/(TP+FP)$ and/or $recall = TP/(TP+FN)$ and/or and $F-score = 2TP/(2TP+FP+FN)$ (being TP= true positives, FP= false positives, TN= true negatives, FN= false negatives), as also highlighted by [48], the fact that different criteria are used to define the TP, TN, FP, FN, depending on the research question specifically answered, makes different methods practically incomparable.

1.1.5 Link prediction methods

In graph theory, link prediction is the task of determining if a relation exists between a pair of nodes and presents many applications in social networks (for identifying the friendship links among users) [79], in citation networks (for identifying co-authorship of co-citation), or in biological networks (for predicting interactions between genes and proteins) [135]. In the context of table understanding it assumes a leading role in identifying the relationships existing among the table columns or among the concepts that the table columns represent. It can be easily treated as a binary classification task in which a positive label is assigned if the relation exists, otherwise, a negative label. ML-based solutions for link prediction have been the subject of many research efforts. Most of these approaches [117],[75],[93] consist of mapping nodes and edges of the graph in a lower dimensional vector space to feed traditional (i.e. very well-known in the literature and successfully applied to euclidean data) ML models.

In Liu *et al.* [117], several structural features like in-degree, Pagerank, clustering coefficient, or average neighbour degree are computed to obtain a vectorial representation of nodes to allow to SVM and Logistic Regression models to learn the data. This approach requires a manual feature engineering step based on the structural information of the graph. For removing these limitations, several embedding techniques that perform automatic feature learning on graphs have been proposed in the last few years. These methods are based on the encoder-decoder model [75] and they differ in the way they define the encoder, similarity, and loss functions. Popular methods are those based on matrix factorization, random walks, and translation models [93]. However, these solutions *i)* can work only on the structure of the graph without considering node attributes; *ii)* present an encoder that is just a lookup on an embedding matrix, so they cannot be used on unseen nodes.

Recently, graph neural networks (GNNs), a branch of DL on non euclidean data, emerged as very powerful solutions to solve ML tasks on graphs [170]. GNNs are deep neural network models that work naturally on graph-structured data and perform automatic feature learning on graphs using the structural information and the node attributes if any. Their encoder is a complex function that depends on the graph structure so it can be used on unseen nodes. The main idea behind their computation is the generalization of the operation of convolution from grid data to graph data. They generate a node v 's representation by aggregating neighbours' features x_u , where $u \in N(v)$, and combining them with its features x_v . Different kinds of "aggregation" and "combinations" steps can be considered to build different convolutional GNNs layers like GCN [101] or GAT [161].

The majority of current GNNs work with homogeneous graphs. Only a few works try to extend the convolution on heterogeneous ones, which characterizes many real-world applications with different types of nodes and edges, like KGs. For instance, R-GCN [146] models relational data keeping a distinct linear projection weight for each edge type; while HetGNN [175] adopts different RNNs for different node types to integrate multi-modal features. Though these methods have shown to be empirically better than homogeneous ones, they typically not fully utilized the heterogeneous graphs' properties, using either node type or edge type alone to determine GNN weight matrices.

When it comes down to multi-relational link prediction in KGs, methods that take node basic properties (i.e. literals) into account have received little attention thus far. Only recently, a few approaches [109, 171] highlighted the importance of node properties for the link prediction task. The majority of methods using node properties can only handle literals of a certain datatype; however, literals can be multi-modal (e.g. textual, numerical), requiring models that can handle multiple types of data. MRGCN [169] processes multi-modal literals as node features of an R-GCN but it does not take into account heterogeneous nodes so all the nodes must exhibit the same property datatypes.

1.2 Approaches for the table understanding problem

Different steps have been introduced in [92] for facing the table understanding problem and generating structured information from spreadsheets that is machine readable. The steps identified by Hurst consist in: *localization*, *segmentation*, *functional* and *structural analysis*, and *interpretation*. Even if in the literature several approaches do not clearly make a distinction among them, the analysis of the most interesting and used techniques developed for pursuing each of them can be the starting point for developing new solutions.

1.2.1 The localization and segmentation steps

The *localization* step consists in the identification of the grid structure composing a “genuine” table within a document, while the following *segmentation* step regards the processing of the localized table area, to split it into its atomic elements, the *cells*, which may be characterized by different sizes or content types.

In the nineties, tables were localized in the imaged documents, that is postscript (ps), tiff, gif, or raster image formats. Therefore, classical image processing techniques (Hough transform, edge detectors, corner detectors – [71]) were applied to identify grids. From the beginning of the new digital era (2000), new structured formats (like pdf, HTML, LaTeX, and XML), which substituted image formats, made the table localization step apparently easier, due to the usage of specific tags/commands to declare table contents. However, since tables have been also used to organize the layout of pages (especially for web pages), in the past 20 years the main issue of table localization has shifted to the discrimination between a real (genuine) table from a layout (fake) table. Among the works for genuine table localization [165, 47, 48, 153, 66], those presented by [47, 48] rely on the observation that, in many randomly sample documents, the lines belonging to genuine tables (Tlines) show a symmetric white space distribution, while text lines or fake table lines (Flines) are characterized by skewed, not symmetric distributions. Starting from this observation, lines were coded in terms of their white-space distribution and then were exploited in DTs for performing a discrimination of Tlines. Tline, sometimes simply extracted by taking lines in between HTML tags <TABLE> [165], are then used to compose candidate table regions, which are coded through spatial and content-based coherency features [165, 48, 66], and are then used to train DTs, SVMs [165], kNNs [48], or NBs [66] for recognizing genuine tables.

In spreadsheets, multiple tables may occur in a single sheet. Therefore, [103, 107] regard table localization and segmentation as the problem of identifying multiple tables in the same sheet, and propose finding the “optimal” partition of a graph, where nodes represent cells and edges define the cells “similarity”. Since the functional and structural analysis of each segmented table is also faced, an overview of the approach is described in Section 1.2.2.

Of note, in the past five years the emerging DL field has produced promising table localization and segmentation results [147, 85, 56], through convolutional [85, 56] layers and recurrent neural networks [147] trained either on the imaged document or on other formats. As an example, based on the analogy between table cells and pixels, [56] codes spreadsheet cells through a vector of 20 features describing their content and layout, and then localizes all the tables using a specific CNN model [114, 42, 110, 81, 69] for object detection and segmentation. The model obtained is characterized by a modular architecture where the first layers extract region proposals, while the topmost layers validate and refine the incoming proposals. Among the different table localization and segmentation methods, the one presented by [136] is to date the mostly used and/or improved [36, 37, 167, 99, 4], given its effective results.

By using CRFs (Section 1.1.4), which are particularly suited for classifying structured data, authors simultaneously perform the first three steps of table understanding (localization, segmentation, and functional analysis). They firstly defined proper line classes for recognizing lines from different parts of hierarchical tables (e.g. TITLE, SUPERHEADER, TABLEHEADER, SECTIONHEADER, and DATAROW, SECTIONDATAROW, ...), lines from text (NONTABLE), or separating lines (BLANKLINE), and then coded each line through a vector of Boolean features describing the content of the line itself and its *conjunction* with neighbouring lines (see Section 1.1.4). After training the CRFs with such line representation, when a new document must be processed, all its lines are coded as described above and they are classified by the trained CRFs.

Example 1 *Figure 1.4 shows the data coding procedure applied to train CRFs and the training/test phase. Precisely, the lines of a document in the training sets are first annotated with a label representing the characteristics of the row (e.g. title, header, row, etc.), and a binary vector representing the results of Boolean (layout or content-based) expressions (e.g. “line contains > 70% bold characters”, “> 70% of characters are digits”). Each Boolean line vector is concatenated with the one associated with the previous line and with the next line. The data coding procedure so far described is used to code all the training corpus. After labelling all the coded training lines (see Figure 1.4-bottom, left to right) the CRFs training algorithm is run, which considers the coded training line, its label and also the labels of the preceding line and produces a trained CRF model that will be used to classify the lines of novel (unlabelled) documents. An unlabelled document is firstly encoded by exploiting the same coding procedure we have just described. Then, the CRF model applied to it produces the predicted annotations.* □

At the end of these steps, a *physical representation* of the table is devised. In the past, classical image processing techniques identified the lines splitting the cells and encoded the table in terms of the relative position of the cells. These techniques did not take into

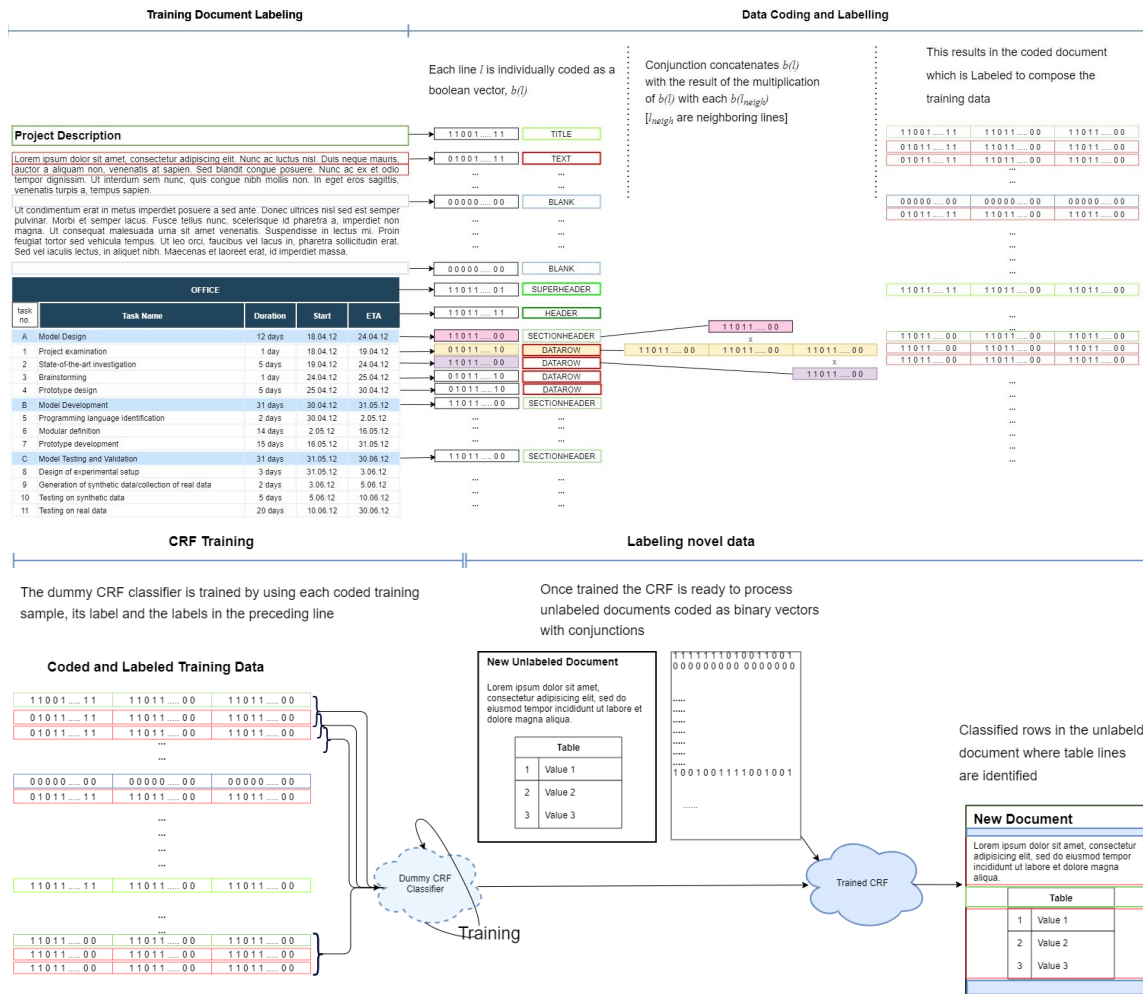


Figure 1.4: The CRF document classification framework

account the existence of malformed tags and table hierarchies [159], while the ones that process recent document formats use specific tags to delimit the cells. After the creation of this hierarchical structure, the the physical representation is encoded in JSON.

Table 1.5 sums up the most interesting state-of-the-art techniques focusing on table localization (L) and segmentation (S) discussed so far and the functional (F) and structural analysis (ST) that will be discussed in the next section. For each approach: column “reference” reports the corresponding paper; column “input formats”, the processed input format (ASCII, image, HTML, XML, pdf, or Excel); column “approach”, the usage of either (i) a set of empirical rules (H), or (ii) supervised machine learners (ML), or (iii) a genetic pro-

paper	input format	step	approach	method	table layout	tables/hierarchy	dataset
[67]	HTML	S	H	positional, content	1D, 2D	single/N.A.	private
[60]	HTML	S+F+ST	H	positional	1D, 2D, C	many/yes	private
[124, 125]	XML	all	H	position, layout, style	1D, 2D	many/yes	private
[103]	Excel	S+ ST	GenProg	spatial, content	1D, 2D	many/yes	private
[47, 48]	HTML	S+F+ST	ML	DT	1D, 2D, C	single/yes	N/A
[167]	HTML	L+S	ML	CRF	1D	many/N.A.	private
[153]	HTML	L	ML	SVM, Parse tree kernel	1D, 2D	many/N.A.	private
[46]	ASCII	L	ML	DT	1D, 2D	many/N.A.	private
[4]	HTML,Excel	L+S	ML	CRF	1D, 2D	many/N.A.	both
[36, 37]	Excel	L+S+F	ML	CRF+SVM	1D, 2D, C	many/yes	private
[66]	HTML	L+S	ML	NB, SVM, kNN, DT	1D, 2D	many/yes	private
[128]	HTML	S	ML	NB	1D, 2D	-	private
[147]	image	L+S	ML	Recurrent Net	1D, 2D,C	many/N.A.	public
[85]	pdf	L+S	ML	CNN	1D, 2D,C	many/N.A.	private
[56]	Excel	L	ML	CNN	1D, 2D, C	many/N.A.	public
[137]	HTML	F+ST	H	coherence rules	1D, 2D,C	N.A./yes	private
[99]	HTML	F+ST	H	coherence rules	2D	N.A./yes	private
[2, 3, 49]	Excel	F+ST	H	spatial, content	2D	N.A./yes	private
[149]	Excel	F+ST	H	spatial, content	1D, 2D,C	N.A./yes	public
[106]	Excel	ST	H	spatial, content	1D, 2D	N.A./yes	private
[35]	HTML	F+ST	H	rectangle order, aggregation	1D-V	N.A./yes	private
[136]	HTML	F	ML	CRF	1D-V	N.A./no	private
[53, 122]	Excel	F+ST	ML	SLR	1D, 2D, C	N.A./yes	public
[38]	HTML	F+ST	ML	logistic regression, DT, SVM	1D, 2D, C	N.A./yes	private
[104, 105]	Excel	F	ML	CART, RFs, C4.5, SVM	1D, 2D	N.A./yes	public
[70]	Excel	ST	ML	DTs, RFs, NB , SVMs	1D, 2D	N.A./yes	public
[126]	HTML	F+ST	KB	probabilistic inference	1D, 2D, C	N.A./yes	private
[61]	HTML	F+ST	KB	maximizes probability	1D-H	N.A./no	public

Figure 1.5: Approaches for location, segmentation and functional and structural analysis.

gramming (genProg) techniques, or (iv) the semantics of cell contents and/or the pairwise relationships between concepts as retrieved by KBs; column “method” reports the specific methods used by the approach, which regard either (i) the rationale behind the Hrules, or (ii) the exploited ML classifiers, recalled in Section 1.1.4, or (iii) the algorithms used by KB approaches for merging the pairwise semantic relationships into relationships among group of cells; column “layout” recalls the processed layouts, as described in Section 1.1.2; column “table/hierarchy” reports, in case of L+S methods, if only a single table or many tables can be identified in a given document, whereas, in the case of F+ST methods, if the technique is able to identify hierarchies on headers/stubs; finally, column “dataset” specifies whether the used test datasets are publicly available or private.

1.2.2 The functional and structural analysis steps

The *functional analysis* step is devoted to the identification of the role of cells as data cells, and header cells, which provide a “description” of the data contained in data cells. This step is often preceded by a preprocessing step, aimed at identifying the table reading or-

der (vertical or horizontal), by generally exploiting layout (style and format) and/or content coherency characteristics. Once headers/data cells have been discriminated and the table reading order has been identified, the *structural analysis* step aims at understanding the relationships between headers and data cells, that is, finding the data cell regions/blocks that each header cell describes and refers. Structural analysis identifies the table class layout and the occurrence of hierarchies on headers and stubs by aggregating data cells of the same header, and by eventually identifying parent-child relationships existing on them.

Even if the functional and structural analysis have been described by [92] as two separate steps, most works do not make a clear distinction between them. For this reason, the discrimination between headers and data cells, which is the aim of functional analysis, is often realized by discovering and exploiting the table reading order and the relationships between cells. Therefore, the steps aimed at functional and structural analysis are often intertwined, so it is difficult to categorize them.

Observing the methods summarized in Table 1.5 it can be noted that the surveyed approaches mainly exploit ML techniques, different types of coherence-based heuristics, and/or some form of semantic descriptions. In particular, methods such as those described in [124, 125, 149] exploit specific heuristics for functional and structural analysis, which are based on knowledge about the domain in which the table is used and also on the terms that are usually employed for representing the access indexes.

In [149] a method based on rules is proposed. This rules may be expressed and applied through rule engines with the purpose of taking into account the boundary features of the tables (presence of horizontal/vertical lines that separate cells), layout features (the text formatting style and background colours of the cells, the alignment of texts), content features (the presence of areas of the tables that contain values of the same type), and so on.

More general methods use the aforementioned features for computing measures of visual and content coherency and/or similarity [121, 48, 137, 99, 66, 35] along rows and columns. Such methods may be iteratively applied by following a top-down hierarchical process, as the one described in [99], where coherence is initially used to find the table reading order (vertical or horizontal) that maximizes the layout coherence (i.e. cells presenting the same tags for the visualization of their content).

Next, depending on the previously detected table reading order, maximization of coherency measures based on formatting style and/or content and/or semantics are used to distinguish between the column/row headers and data cells. The iterative application of the coherence maximization procedure allows identifying all the components of hierarchical tables. The opposite rule - and coherence - based approach is exploited in [137], where authors firstly use the cell content and style to classify them into A(tribute)-cells, which describe the con-

ceptual nature of the instances in a table, and I(nstance)-cells, that represent the actual data cells or the instances of the concepts represented by a certain A-cell. Next, the maximisation of row/column coherence, which is computed by accounting for the cell similarities in each row/column, allows defining the table orientation (vertical/horizontal). The retrieved orientation is then used to split the table into minimal regions, each consisting of cells with the same functional type and belonging to the same logical unit. Finally, an iterative bottom-up process starts, which aggregates all "coherent" minimal regions, therefore expanding them until all the logical units are fully reconstructed and, therefore, the logical structure of the whole table is finally uncovered. A similar hierarchical aggregation approach is defined by [35], which starts with the identification of minimal rectangles and then aggregates them by heuristics that provide a rectangle ordering scheme.

Though useful, methods based on coherency and/or other types of heuristics often become tortuous as an increasing number of rules are added to correctly perform the functional analysis of 2D hierarchical and/or complex tables. Instead, methods based on ML techniques, such as DTs, RFs, NB, SVMs [136, 48, 36, 37, 104, 104, 38, 70] infer rules by analyzing the training sets. Based on this consideration, [48] exploits hybrid functional and structural table analysis methods, where functional analysis is performed by coding the relationships of the cells through coherency measures, which are used as input to classifiers (SVM, DTs, NB); moreover, heuristics on spatial and content-coherence allow defining the parent-child relationships, therefore characterizing the structural relationships.

SVM classifiers, together with CART trees, Random Forests (RFs), and C4.5 classifiers are also used for functional analysis by [105, 104], where cells are described by the classical layout (style, font, spatial) and content features and by Excel reference features. Of note, these works apply a pre-processing feature selection phase¹ before training, which is uncommon in other literature works. After classification, spatial and content-based heuristics (described by [106]) are first used to simultaneously repair classification errors and form "cell clusters" (that is rectangular areas composed of cells with the same functional types); then, other spatial and content-heuristics merge the cell clusters and infer the structural relationships between them. To avoid tortuous heuristics, [107, 103] extend the structural analysis proposed in [106] by viewing the identification of the structural relationships between cells clusters as a graph partitioning problem, where the optimal partition is found by maximizing a "fitness" function through genetic programming (the fitness of a partition expresses the spatial and content coherency between cells cluster related by structural relationships such as header/content, or content/content, or content/header). Probably the

¹Feature selection techniques extract an informative feature set by discarding features with low discriminative power. To assess the features' discriminative power, several techniques may be used (e.g. statistical tests, information gain, Pearson correlation with the cell label, or feature importance).

most successful example of ML method for functional analysis is the CRF based method presented by [136] and extended by [36, 37, 4], with the aim of retrieving a table hierarchy where either relationship between couples of cells are expressed by (parent, child) pairs [36, 37] or by similarities in the visual attributes of table cells [4]. After using CRFs to detect tables and segment their cells, [36, 37] further clean classification errors by coding all the possible (parent, child) pairs through layout, positional, and content similarity features, and then filter out unfeasible pairs by applying the SVM classifier, trained to recognize true pairs. Once the pairs are recognized, loops are heuristically removed and the tree hierarchy and relational structure representing the functional and structural role of each cell is constructed. A human-in-the-loop [86], incremental, user-interactive labelling and training ML strategy has been proposed by [38]. Starting from the consideration that, in many complex Excel tables, it is difficult for any user to define all the required functional and structural cell classes in advance, the authors propose to adopt an incremental labelling and learning system. More precisely, logistic regression classifiers, DTs, and SVMs are initially trained on an initial user-defined set of labelled cells.

After training, the classifiers are used to classify the functional and structural properties of the cells in novel tables, and the user is then asked to check and correct the obtained result, by eventually introducing novel functional and structural cell classes, which are again used to improve the trained classifiers' "knowledge". Such iterative training, testing and user-check procedure continues until the user is satisfied.

Example 2 *Figure 1.6 shows an example of a human-in-the-loop incremental training process. It starts by coding and labelling the cells in tables from a training corpus; the feature vectors, labelled according to their functional/structural role, are then used to train any classifier; the trained classifier is used to label the "unknown" cells in a novel corpus; the user checks and corrects the classifier's labels, so that novel correctly labelled examples may be added to the training set, and the cycle continues until users' satisfaction. Figure 1.7, by contrast, shows the functional and structural analysis system presented in [105, 104, 106, 103, 107] that exploits a classifier (SV, DT, or C4.5) to initially predict the cell functional role. Next, heuristics are used to check and correct the predicted cell labels, and the table is then represented as a graph where nodes are arranged and linked according to the predicted labels, spatial, layout and content similarities. A genetic algorithm is applied to the so obtained graph in order to identify the optimal graph partition, that is the partition maximizing the intra-subgraph coherency while maximizing the inter-graph coherency-based distance. □*

The advantage of using ML methods relies on their ability to learn the structural relationships among different cells (header cells, data cells, cells in stubs, and so on...), which characterize all tables in a corpus. Based on this consideration, an ML approach exploiting

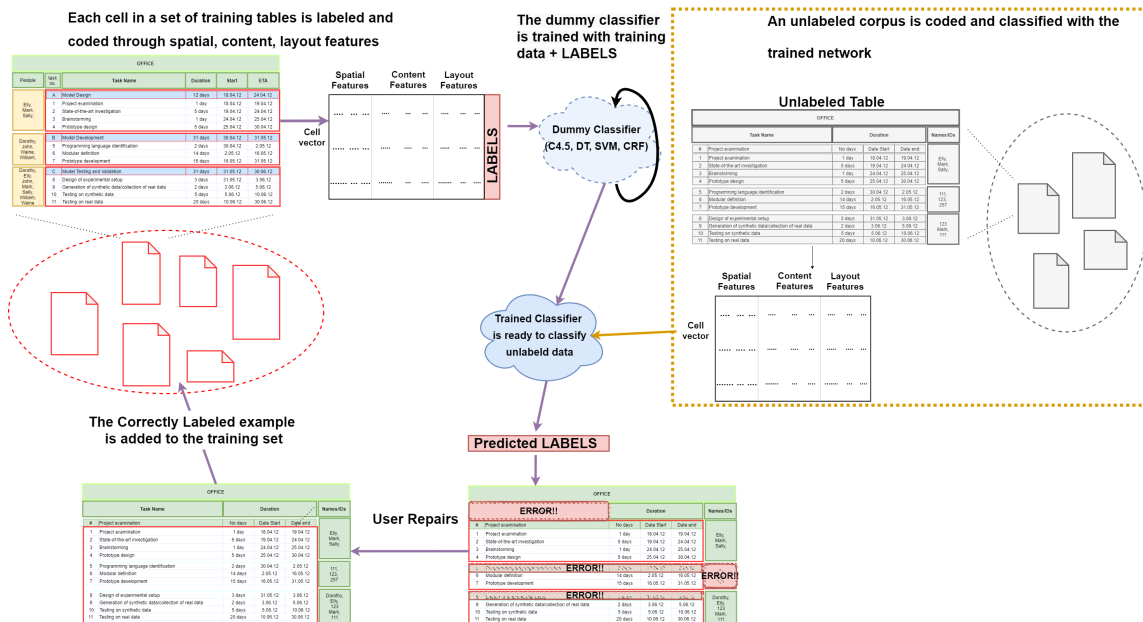


Figure 1.6: The incremental human-in-the-loop procedure proposed by [38].

a Statistical Relational Learning (SRL) approach has been presented in [53, 122], which uses a probabilistic query-based classifier, using first-order logic as a representation language. The algorithm first constructs features by mining frequent queries, then selects the most representative features by using a stochastic local search procedure [87]. After coding each cell with the selected feature set, a maximum “a posteriori” estimation allows classifying each cell as a header or data cell and identifying the parent-child structural relationships between header cells and their data cell.

In [126] tables are modeled as a graph, where table cells are nodes of the graphs and the edges linking them are initially weighted by the strength of the semantic relationships between cells as retrieved from KBs or Linked Open Data [15]. This model can be exploited both for the functional/structural analysis and for the interpretation of the table content in terms of a KB. Using probabilistic inference approaches similar to the message-passing schema of deep belief networks [108] the strengths (semantic relationships) are then “propagated” between the graph nodes until convergence, that is, until the functional role of each cell, the structural relationships between couples of cells, and the semantic interpretation of such relationships are fully defined.

Semantic analysis may be also used to simultaneously enrich the functional analysis with the table’s structural relationships and the interpretation of its content. As an example, TAIPAN

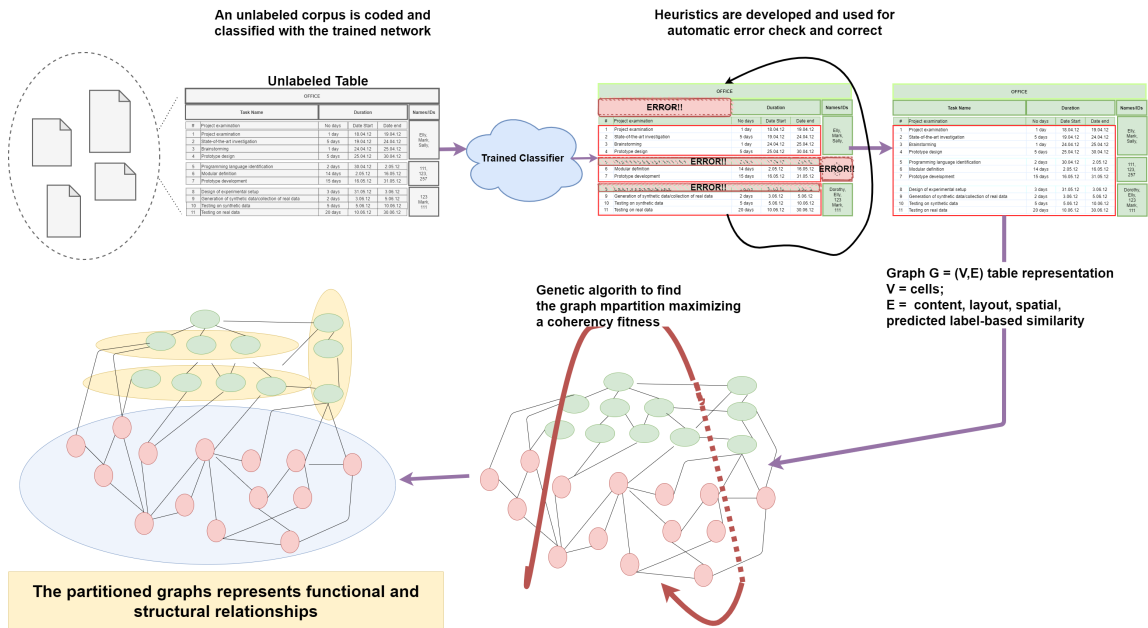


Figure 1.7: The system presented in [105, 104, 106, 103, 107]

[61], which is unfortunately restricted to vertical 1D tables, exploits KBs to first understand when a semantic relationship exists between any pair of cells in the table. Next, the column that contains the highest number of cells included in a pairwise relationship is selected as the “subject column” (that essentially is the column containing the headers). By exploiting the identified relationships, TAIPAN further expresses the table as a relational model, therefore providing also an interpretation of the table (interpretation step, Section 1.2.3).

The final point that should be highlighted is that functional and structural analysis may also be applied for dynamic error checks in spreadsheet files. In this context, [62, 2, 3, 49] interestingly view each Excel spreadsheet as composed of “units” (i.e. consecutive parts of rows and cells associated with a label). Each value in a spreadsheet (except blanks) potentially defines a unit, and the unit of any cell is intuitively determined by its headers [62]. Based on this definition, authors analyze Excel tables by exploiting spatial and content heuristics to both discriminate between headers and data cells and to uncover their relationships. The results of the functional and structural analysis allow applying a unit-based grammar for checking the consistency of each cell and its unit. Note that, since the exploited heuristics also contain rules based on the string content of cells (e.g. cells containing the string “total”, and “sum” are signed as headers when they are located in the topmost row or leftmost column), the method also provides a first table interpretation.

1.2.3 The interpretation step

The *interpretation* step assigns a semantic description to the content of the table in terms of a well-known data model like the relational model or an ontology. In this way the semantics of the table is well-described and its content can be uniquely interpreted in a given domain and facilitate the interoperability of the applications working with the data. This task is however difficult to realize, because of the following aspects, which are mainly due to the heterogeneity of both table contents and table structures. First, different notations can be exploited for representing the same kind of information, e.g. the negative number -10 can be represented as (10) in statistical tables, and the gender could be simply identified by “M/F” or by a prefix “Mr./Miss.”. Second, when numeric tables are treated, the units of measure are sometimes omitted, e.g. the number 10 is used to express distance in terms of kilometres or millions of users/products depending on the context, or as shortcuts for expressing millions of dollars, or billions of euros. The problem is further exacerbated by the presence of different conventions for the representation of the same column names (e.g. address, location, residence) or values (e.g. Barack Obama, B. Obama, Obama B.), and by the fact that some columns often contain different values, e.g. a column containing name, surname, zip code and address of an individual. The aforementioned examples highlight that an automatic table interpretation must consider a lot of parameters, which are often unspecified in the table content itself. In practice, the information contained in other parts of the document where the table is located, together with the knowledge retrievable from external KB, should be taken into account.

Since the problem of identifying a semantic description from heterogeneous data sources is at the base of data integration [55] and exchange [8], many approaches for reconciling the syntactic and semantic heterogeneity of data sources have been proposed, which mainly exploit *schema matching* or *schema mapping* methodologies [11]. More precisely, considering the presence of correspondences between the schema in the source representation and the target schema, *schema matching* methods find the matches between the source properties and the target properties, while schema mapping techniques find the mapping rules for transforming the information in the source according to the target schema. Among *schema matching* techniques, *ontology-matching* methods [52, 154] have been extensively investigated, where the target schema is an ontology containing different classes that can be also hierarchical organized. Conversely, *KB-mapping* approaches (e.g. [127, 14, 40, 179]) map cell/tuple values of the table to KB instances, and then exploit both probabilistic graphical models and iterative algorithms to explore the correlation between different matching tasks for disambiguation. More recent *ML-based* techniques (e.g. [57, 155, 34]) do not exploit any schema matching and interpret the table by using black-box DL models that are trained on annotated corpora. In the remainder, we discuss these three kinds of approaches.

Ontology-matching approaches. One of the first *schema matching* techniques developed for discovering *complex matching*, that is matches among a combination of attributes in the source schema to a combination of properties in the target schema, has been proposed in [52]. These correspondences are detected through a set of special-purpose searchers, ranging from data overlap to equation discovery and ML techniques.

The possibility of identifying relationships passing through other entities has been considered in [154, 65] (e.g. given two actors, they can be related by means of the film in which they have played). In [154], the authors propose an approach for annotating sources of information with a *semantic model* that represents the implicit meaning of data by specifying the concepts and relationships existing among the source data relying on a domain ontology O . The semantic model is a graph that contains both nodes representing the table columns and the concepts of the domain ontology identified in the source. Moreover, edges represent the relationships existing among the identified concepts in the given source. Each time a new incoming table T is considered, they set up an approach for determining the semantic model of T relying on the semantic models of the already considered data sources. Their approach consists of four steps: *i*) using the sample data extracted from T , learn the semantic types of its columns (by means of the approach proposed in [140] for learning from examples how to assign ontology classes to the source attributes); *ii*) construct a graph from the known semantic models, augmented with nodes and linked induced directly or indirectly by the structure of the domain ontology; *iii*) identify the candidate mappings from the table columns to the nodes of the graph; *iv*) build the candidate semantic models and rank them.

Moreover, by means of their graphical environment, the user can interact with the system for modifying the model and these modifications will improve future schema matching.

Example 3 *The top part of Figure 1.8 reports the semantic models that have been already associated with two sources (source1 and source2) in the approach proposed by [154]. The semantic models exploit the classes (the ovals in the figure) and properties (labels associated with the edges) already included in a domain ontology for capturing the intended meaning of the sources. The authors leverage attribute relationships existing in the developed semantic models to hypothesize attribute relationships between the attribute in a novel source (source3 in the figure) that needs to be included. The proposed approach is organized in 4 steps: 1) the semantic types of the attributes in source3 are learned; 2) a graph is constructed by integrating the semantic models (i.e. in our case source1 and source2) augmented with nodes and paths connecting nodes of the graph; 3) the candidate mappings from the source attributes to the node of the graph are computed; 4) the candidate semantic models are generated for the candidate mappings and ranked. The bottom right corner of Figure 1.8 reports the semantic model generated through these steps. □*

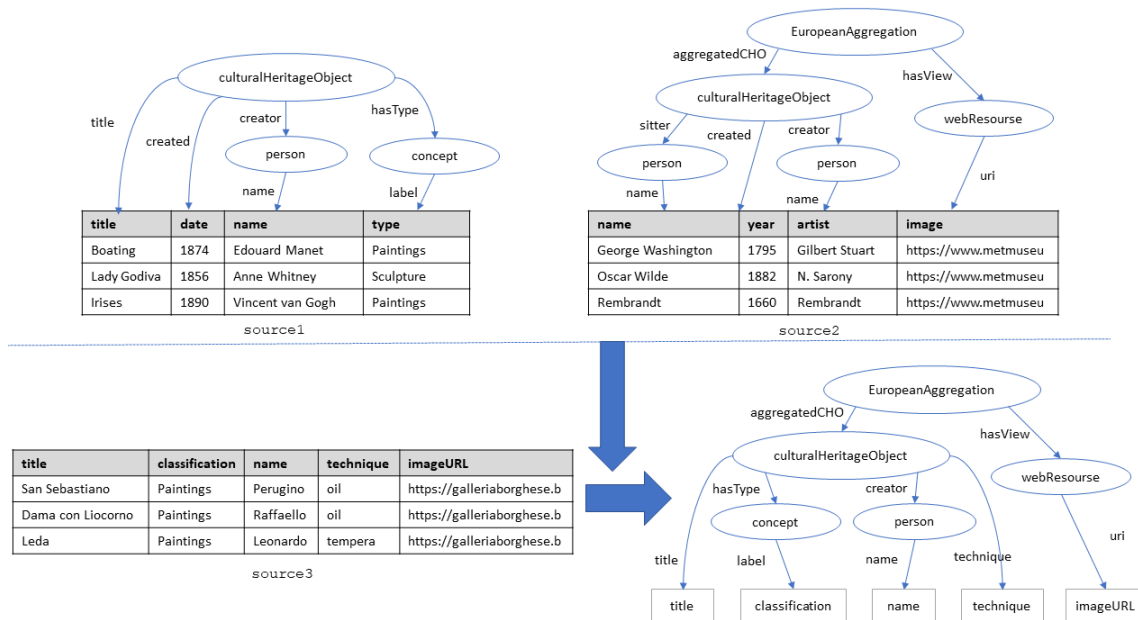


Figure 1.8: Semantic annotation of a table (proposed by [154]).

In [65], the authors extend the work proposed in [154] in several directions. First, instead of using the semantic models associated with different data sources, they propose to use directly the domain ontology for identifying the semantic model of the new source. As in [154], they use an approach for associating the concepts to the table columns and for the creation of a graph that contains all the possible relations in which these concepts can be involved. Then, by weighting the edges with a weight that depends on the specificity of the relations (with respect to the ontology hierarchy) they identify a minimal graph that can be used as a semantic model of the source. At this point, they consider a GNN model (based on R-GCN) that is constructed on a consolidated knowledge graph for inferring relations that are commonly present between pairs of concepts in KG. In this way, they avoid using complex semantic models for representing different kinds of relationships among data and exploit the vector representation of a consolidated knowledge graph for automatically learning latent features of their entities and relationships by exploiting the local neighborhood structures.

KB-mapping Approaches. Several approaches have been proposed that consider both the values of the table cells and the table schema for the annotation with the semantic classes and relationships. According to [34], most of the approaches can be classified as joint inference models and interactive approaches.

In [162], annotations are added to a web table [30, 29] to describe binary relations among columns by considering triples $(arg_1, predicate, arg_2)$ extracted without supervision from the Web and by exploiting a maximum-likelihood model for identifying the predicate that most likely occur in the considered pair of columns. The authors discuss the issues in identifying useful triples from the Web that can be applied in this context (low recall) and that their approach is able to identify a few relations among the considered columns.

A probabilistic graphical model is proposed in [116] that highlights different matching and searches for value assignments of the variables that maximize the joint probability. The model is used for simultaneously choosing entities for cells, types for columns and relationships for column pairs extracted from the YAGO KB [141].

A similar approach is described by [127] where a probabilistic graphical model is proposed that captures more semantics than the one proposed by [116], including relations between column headers and between row entities that are extracted by querying DBpedia. Starting from a Markov network graph in which table columns and the cell values represent the variable nodes and the edges between them represent their interactions, through their probabilistic model they identify the most likely relations existing among columns by taking into account also cross-columns' relationships. Moreover, the approach does not consider numeric values, but only strings.

Another probabilistic graphical model is proposed in [14] in which this assumption is weakened but assigns a higher likelihood to entity sets that tend to co-occur in Wikipedia documents. The incompleteness of the KB in terms of coverage of the values in the table is also considered by [40], where the authors further include the need to considering the presence of dirty data in the table since it makes it much harder the identification of a matching with the KB. To face these problems, [40] introduces the Katara system that first discovers different table patterns containing the types of columns and the relationships between them. More than one table pattern is discovered because the same cell values can be associated with different concepts and the pattern. Moreover, the pattern can be partial because some values are not described in the KB. Then, by posing specific questions to the user, the system is able to identify the best table pattern, introduce new facts in the KB to be used in the next matching activities, and also correct mistakes when inconsistencies are identified. The system relies on a probabilistic model for the identification of the top-k table patterns and for choosing the best one.

Example 4 *Figure 1.9 shows the approach proposed in Katara for interpreting the content of the table reported on the left-hand side (the example is extracted from [40]). By exploiting a KB, Katara is able to identify table patterns between a table and a KB by checking the existence of correspondences between the cell values and the KB instances. A table pattern*

A	B	C	D	E	F	G
Rossi	Italy	Rome	Verona	Italian	Proto	1,78
Klate	S.Africa	Pretoria	Pirates	Afrikans	P. Eliz	1,69
Pirlo	Italy	Madrid	Juve	Italian	Flero	1,77

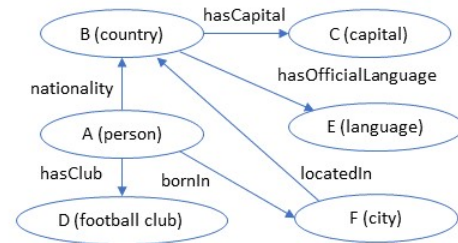


Figure 1.9: Annotating table columns by exploiting the Katara system [40].

is a labelled graph (like the one reported on the right-hand side of the figure) in which nodes represent attributes (with the associated type) and direct edges represent the relationship existing between the attributes. Depending on the considered table, several candidate table patterns can be identified, and the selected one is determined through a pattern validation mechanism. This mechanism depends on the cardinality and size of the candidate patterns. When the size is small, the system asks a group of users to pick the right table pattern for the processed table. When the size of the candidate table patterns is large, they are decomposed into smaller patterns to formulate simpler questions, which crowd workers can easily answer. Each tuple of the table is finally annotated with "validated by the KB", "validated by the KB exploiting the user answers" or "erroneous". Indeed, users can identify mistakes occurring in the table and fix them. All the users' interactions are then used in subsequent activities of table interpretation. □

TableMiner+ [179] and T2K Match [144] are two iterative approaches. A bootstrapping pattern is first adopted by TableMiner+ to learn an initial interpretation with a sample of table data and the interpretation is refined by means of the remaining data and by taking into account the relationships existing among columns. The author considers both an external KB and information around the table (e.g. header, footers or the caption associated with the table). Finally, the approach distinguishes between columns for which named entities can be identified in the KB and literal columns containing data values of entities. In T2K the authors use an iterative schema and entity matching approach by means of which the algorithm can benefit from entity correspondences for finding schema correspondences and vice-versa that improve its effectiveness and efficiency in dealing many web tables against a cross-domain KB.

DL-base Approaches By following the success of semantic embedding techniques like word2vec [123] developed in the context of DL, a few approaches have been proposed for learning semantic table annotations. [57] uses the contextual semantics of an entity in the KB for disambiguation in cell-to-entity matching, whereas [155] accelerates searching and

dealing with the missing linkage in the column to class matching with probabilistic graphical models as the one adopted by [116]. [34] proposes an approach for the semantic annotation of table columns assuming that names and table structures are unknown. Given a column of the table, the approach first retrieves the column's candidate entities from a KB and then adopts the classes of the matched entities as a set of candidate classes for annotation. For each class, a customized binary CNN classifier which is able to learn both inter-cell and intra-cell locality features is trained and applied to predict whether embedded cells of the column are of this class. A key issue that is faced in this paper is the annotation of text phrases (e.g. "Apple", "MS", "Google") by considering the context. In the case of the example, the approach allows the identification of the annotation "IT Company" instead of "Fruit" and "Operating System" because the term "Google" allows the identification of the proper context. The ColNet approach has been extended in [33] to leverage inter-column semantics through a hybrid neural network (HNN). Sherlock [89] uses a deep feed-forward neural network to make type predictions based on different kinds of features (column statistics, paragraph and word embedding, and character embedding) of column values. SATO [176] extends the Sherlock approach by including a topic modelling module and a structured prediction module for dealing with the low prediction accuracy for underrepresented types and for taking into account the context in which the table columns are located. Finally, C^2 [98] exploits a Maximum Likelihood approach on the actual cell values of a column. By taking into account different sources of structured data (data table lakes like Wikipedia tables, and knowledge graphs like DBpedia) to maximize the likelihood of a concept. These techniques are robust to noise in data and superior to traditional approaches.

In the context of semantic web, several KBs, e.g. WebTable [30], YAGO [141], DBpedia [9], and Freebase [16], have been defined for the semantic annotation. Moreover, these KBs have been used for the creation of benchmarks adopted for training the proposed prediction models. T2D [144] contains 779 tables (with around 400 entity columns encompassing topics like persons, species, and organizations), with around 26'000 DBpedia matches, and 420 DB property matches. [116] contains around 400 manually annotated tables from Wikipedia with 428 entity columns and 5'600 DBpedia entity matches. [57] contains 485'000 tables from Wikipedia with around 4.5M DBpedia entity matches. IMDB contains over 7'000 tables from IMDB movie web pages and Musicbrainz contains 1'400 tables from MusicBrainz web pages annotated with Freebase [179]. [88] proposes Viznet that contains 31 million datasets mined from open data repositories and used in the context of column-to-type matching of tables in the Sherlock system [89]. Sherlock provides a total of 11'700 crowdsourced annotations from 390 human participants. [96] proposes NumDB, a dataset of 389 tables generated from DBpedia where the primary emphasis is on assigning semantic labels to numerical values in tables.

1.3 Extracting and transforming tables

While the aforementioned table understanding approaches are ultimately aimed at automatically understanding the semantic content of the tables (e.g., for answering user queries or for exchanging information), some other table understanding approaches and/or programming languages have been proposed, that allow an “understanding” of the table specifically aimed at data repair, extraction and layout transformation (Section 1.3.1) or at table coding into a relational or Resource Description Framework[102] (RDF) format (Section 1.3.2). Since the latter methods need knowledge about the functional and structural table relationships, they either apply some of the functional and structural analysis techniques described in Section 1.2.2, or they define programming languages to allow users to program such relationships. In the second case, since programming may be difficult for some users, programming-by-examples approaches have been proposed (Section 1.3.3), which automatically codes programs that allow reproducing users’ examples.

1.3.1 Basic extraction and transformation tools

Tables often contain errors, missing values or require layout transformations that, due to the great amount of data, are not easy and require a lot of effort from the user in the identification and correction of the variety of occurring errors. Systems like OpenRefine, Potter’s Wheel [139] and Wrangler [97] have been developed for supporting the user in cleaning dirty values and generating new tabular representations.

OpenRefine (<https://openrefine.org/>) is a tool that allows users to load data, clean the errors occurring within it, apply transformations, and reconcile and match data. The system has a clustering feature useful to fix issues occurring in a group of data through the graphical interface. Some user-defined heuristics can be applied for tuning the clustering algorithm, and data transformations between formats are also allowed by using a set of operations such as replace string commands, filter functions, and split functions. Though useful, the system works only on data in a proper tabular format.

Potter’s Wheel [139] is an interactive data cleaning system that combines the application of data transformations with discrepancy detection. More precisely, based on the assumption that each data value in a table (where a data value may, e.g., be a string with the list of values in a table-column) is composed of “sub-components” (in case the data value is a table-column, the sub-components may, e.g., be the column cells) corresponding to different constrained domains, Potter’s Wheel uses a visual interface to allow users defining custom domains and their constraints, which are input to the *discrepancy detector*, and examples of required table transformations, which are input to the *transformation engine*. The *discrepancy detector* uses the provided domains and constraints to learn both the parsing

algorithms that allow dividing each data value into sub-components and the discrepancy check algorithms, tailored to each domain, for checking discrepancies between the parsed values and their domain constraints. Once trained, the discrepancy detector runs in the background and continues parsing novel user input to check for discrepancies. Simultaneously, the *transformation engine* applies a program-by-example approach (see Section 1.3.3, to learn the automatic algorithms for reproducing the data transformation examples and apply them to the data). Thanks to the visual interface, Potter's Wheel continuously shows the automatically transformed data and the results of the discrepancy checks, so that the user may decide to modify and/or improve them by defining different/more domain constraints and/or transformation examples.

Wrangler [97, 157] is a visual-interactive system that, similarly to Potter's Wheel, learns domains, domain constraints, and table transformation rules from user examples, and continuously shows the transformed tables to allow an interactive user-guided refinement of the transformed table and of the transformation rules. However, Wrangler extends Potter's Wheel language with additional operators for common data cleaning tasks; moreover, it allows a more interesting human-machine interaction, since it uses the graphic interface and an "inference engine" to suggest a series of applicable transformations. The "inference engine" has three components and uses as an input the user interactions, the data descriptors (e.g. column data types), and the historical usage statistics. The first component uses user interactions to infer the transformation parameters (row, column, text selection). The second component generates a list of transform suggestions. The third component rank-orders the suggestions depending on their data type and the previous user choices.

All the aforementioned systems exploit a graphic interface to give feedback to the users and to help them during the process of error detection, error fixing and data transformation. However, Potter's Wheel allows users to define arbitrary domains to enhance the discrepancy detector algorithm. OpenRefine does not always allow the specification of transformations through the graphic interface and sometime the knowledge of a command language is required. Differently from OpenRefine, both Wrangler and Potter's Wheel exploit the user's interactions; Wrangler uses them to provide suggestions for further transformations, while Potter's Wheel uses them to improve discrepancy detection.

Example 5 *Figure 1.10 shows the application of the steps identified by [97] for the transformation of the table reported in the top left corner into the final one. By means of its graphical facilities, the system enables the user to preview the effects of the application of different transformation operations on the current version of the table and to select the one that will easily lead to the final format.*

Starting from the first version of the table reported in the top-left part, the user selects row

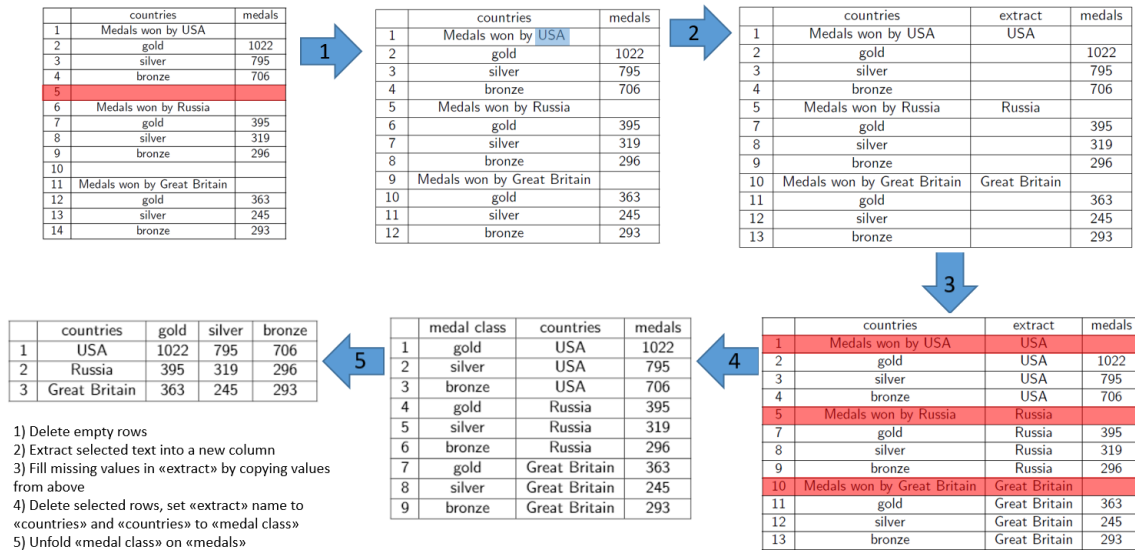


Figure 1.10: Application of the transformation operations proposed in [97].

number 5. The system suggests two options (remove all the empty rows occurring in the document or remove the fifth one only) and the user chooses the first one. In step 2, the user selects a string in row number 2, and the system suggests using the extract operator to copy the selected string into a new column. At this point, the inference engine also suggests applying the same modification to all the strings in the same column that occur after the "by" particle. In the third step the user chooses, among the list of suggestions, the option "fill column extract by copying values from above" whose effect is reported in the table in the right lower corner of the figure. At this point (step 4) the user selects the three rows that wishes to remove from the table. Finally, in step 5, the user requires the application of the unfold operation to create a cross-tabulation. This operation moves the information from data values to column names. Therefore, the basic operations made available by the system are proposed to the users by means of graphical interfaces and the inference engine. In any case, the final decision of the operation to apply is left to the user.

1.3.2 Transformation tools to the relational/RDF models

Beyond the table localization, segmentation, and functional and structural analysis, several methods designed algorithms for expressing the inferred structural relationships as relational models (JSON or XML formats) or RDF formats. TranSheet [90] and TabbyXL [150] introduce languages for the specification of the transformations needed to simultaneously define the table's functional and structural relationships and then convert them into a re-

lational format. Moreover, Senbazuru [37] and HaExcel [49] automatically extrapolate the table functional and structural relationships by applying some of the algorithms described in Section 1.2.2 and then exploit rule-based algorithms to define and combine the relational tuples representing the structural relationships.

Specifically, TranSheet [90] uses a “formula-based” (hierarchical) language for the definition of: (i) value mappings, which group one or more data cells by labelling it with a unique atomic label, and (ii) structured mapping, which map a set of atomic labels into a unique structured label. Such labelling procedure allows defining hierarchical structural relationships, which are then transformed into a relational format. With TabbyXL [150], users may define functional and structural relationships by using either the standard DROOLS language [142], or CRL [151], a specifically developed rule-based language for implementing programs aimed at: (i) cell cleansing issues (merging, splitting, updating data), (ii) role analysis, to recover entities and labels as functional data items presented in tables (marking all the cells having the same role or located in the same functional region with a fixed value), (iii) structural analysis (entity-label association), (iv) interpretation (labelling with a category).

Differently from the aforementioned approaches, Senbazuru [37] automatically infers the table functional and structural relationships through CRFs (Section 1.1.4); next, based on the retrieved structural information, a tuple builder generates a set of annotated relational tuples, which associates each header to a data value and annotates such tuple with the header attribute; finally, the relation constructor assembles the tuples into a relational table. HaExcel [49] exploits the Fun algorithm [129] to automatically extract the functional dependencies in a spreadsheet file and then determines the relational schema, with candidate foreign and primary keys, by using a standard inference procedure from the relational database theory. In the last step, a Relational Intermediate Direct Graph is generated. The nodes of the graph represent the schema and the directed edges represent foreign keys between those schemas. The graph represents the relational schema of the used spreadsheet. The advantage of HaExcel as well as Senbazuru’s, is that the functional and structural relationships, and consequently the data and layout transformations, are automatically determined. By contrast, the TabbyXL and TranSheet approaches require a higher effort by users, since the set of rules for data transformation must be explicitly coded.

Example 6 *Figure 1.11 reports the Senbazuru data extraction process for extracting from a spreadsheet a relational table with the average grade of exchanging and regular students of a CS department that passed exams and the number of those that did not pass exams. The frame finder component of the system identifies the stub (with its hierarchical organization), the content, and the column header highlighted in Figure 1.11(a): Senbazuru automatically extracts a potential hierarchy from the stub as reported in Figure 1.11(b) and, when needed, supports the user in its modification through a drag-and-drop interface (Figure 1.11(c)). Sen-*

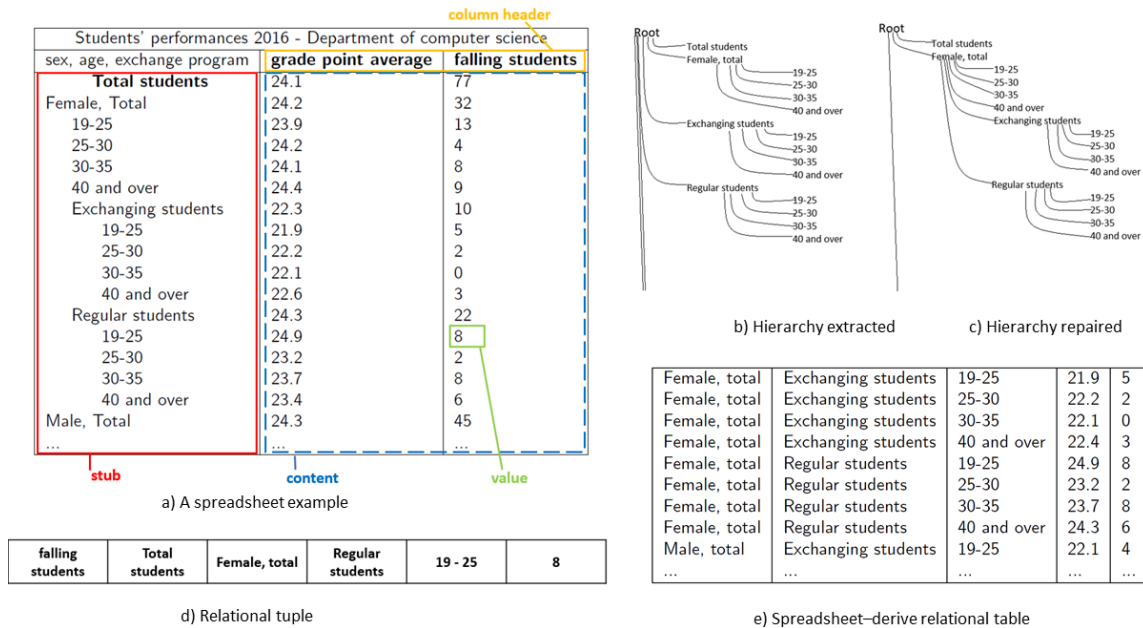


Figure 1.11: Extraction phase in Senbazuru [37].

bazuru uses a tuple builder to generate a relational tuple for every single value in the content area. The value is indexed through the column header and each stub value in the identified hierarchy. An example of a relational tuple is reported in Figure 1.11(d) for the value highlighted with a green box in Figure 1.11(a). Finally, these relational tuples are collected into a relational table (Figure 1.11(e)) by clustering together attributes in different tuples into consistent columns (e.g., the first column of Figure 1.11(e) reports the student gender). □

Despite these approaches that transform the content of generic tables in the relational format, few techniques have been proposed for generating RDF triples. R2RML [50] is a W3C mapping language that supports users in the specification of mapping rules between relational tables and a target ontology in order to publish data in RDF format. The use of this language is not really easy because the user needs to learn its syntax and understand how to map the source schema to the target ontology. RDATE [160], RDF123 [76] and WLWrap [113] are tools that have been developed for supporting the users in this activity. RDATE offers graphical facilities for specifying mapping rules, whereas RDF123 and WLWrap have been specifically tailored for working with spreadsheets.

1.3.3 Programming by example approaches

Since the data extraction and transformation tasks are tedious, error-prone, time-consuming, and often require expert programmers for their development, in the past decade programming-by-examples approaches, which automatically synthesize programs by examples, have been proposed. The general approach used by these systems [74], ProgFromEx [73], FlashRelate [10], and Foofah [95] is to infer the set of transformation rules (program) from a set of user-provided examples composed of (input) tables and their corresponding and desired transformed (output) version. Given such examples, the synthesized program should be able to similarly transform spreadsheets having the same structure into relational tables. While such examples are mostly composed of some of the (input) tables and their transformed desired (output) version, some other methods, e.g. [10], do not need the whole transformed output table, but only some representative input-tuple to output-tuple transformations. The transformations carried out by the programs synthesized by the aforementioned systems can affect different aspects of the tables: the layout; the syntax; and the data semantics. A table layout transformation changes the arrangement of the cells, a syntax transformation reformats their content, (e.g. string splitting), while a semantic transformation involves manipulating strings that need to be interpreted as more than a sequence of characters. ProgFromEx and FlashRelate are limited to layout transformations, Foofah provides also syntactic transformations and [74] faces all the aspects.

The ProgFromEx approach infers a table program using two different types of components: filter programs and associative programs. A filter program selects a subset of cells of an input table according to a condition; if the mapping condition is verified, the cell content is copied to the output table. An associative program defines a mapping from the coordinates of the cells in the input table to those the output table. The map is defined starting from a filter program that is then altered to obtain the desired table. To define a transformation program, ProgFromEx selects a set of filter components from a fixed set of candidate map rules using the input and output examples. The filter is selected if, when applied to the input table, it produces the output table. The set of selected filters is then used as the initial component of the associative programs.

Instead of using input and output tables, FlashRelate needs only some “positive” and “negative” examples showing how the layout of some tuples in the input table needs to be transformed to create a novel layout in the output table. Given the positive example, which represents the desired transformation, the regular expression learning algorithm presented in [7] is firstly used to infer the rules, which are then translated into the Flare language, for generating the initial layout. Negative examples (i.e. unwanted, wrong transformations) are then used to infer the rules for detecting transformation errors.

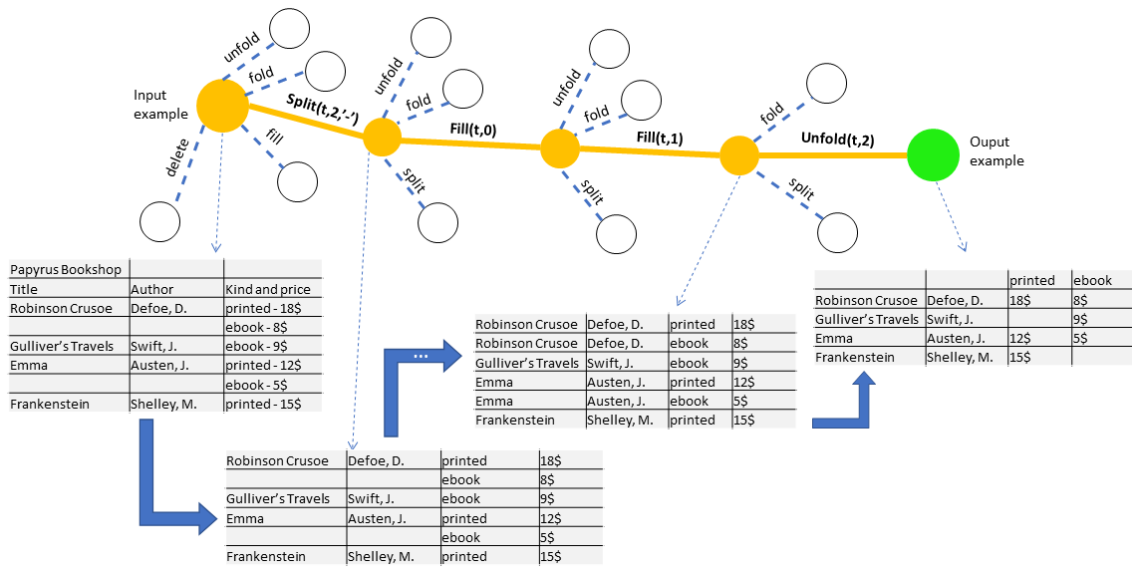


Figure 1.12: Generation of a synthetic program through Foofah [95].

After defining a Domain Specific Language (DSL) for tabular data transformation, [74] defines an algorithm for processing user-provided examples to infer synthesized programs for layout, syntactic, and semantic transformations of tabular data. Though different programs are inferred for each transformation, the algorithm for inferring them always uses a "generate and intersect" strategy. More precisely, the generation step processes each example to generate a set of Direct Acyclic Graphs (DAGs), each representing one of the possible expressions for mapping the input example to the output example. After obtaining different sets of DAGs for all the examples, in the intersection step, "similar" DAGs are merged into a unique partition, that represent a unique transformation. To choose the most representative transformation among all the transformations in a partition, authors heuristically define rules for ranking the transformations relying on their computational efficiency, so that the most efficient transformations are selected from each partition, and are then used to create the final layout, syntactic, or semantic synthesized program (coded through the DSL).

Foofah exploits the user-provided example table, v_0 , and the desired transformed table, v_n , to initially compose a state-graph, $G=(V,A)$, were two nodes, v_{t-1} and v_t , and the edge, $e(v_{t-1}, v_t)$, connecting v_{t-1} to v_t represent the change of state (from v_{t-1} to v_t) when transformation e is applied. Starting from v_0 the graph G is firstly built so as to represent all the possible table transformations applicable to v_0 , and to the resulting tables, until v_n is ob-

tained². Since the paths from v_0 to v_n are many, the best walk, that is the best sequence of transformation which constitutes the inferred program, must be chosen. To this aim, after opportunely pruning G to reduce the number of available choices, inspired by the graph-based search A* algorithm [78], Foofah starts from v_0 and applies a greedy approach to choose, at each step, when the walk is at table v_t and must choose to which neighbouring node to move (i.e. the transformation to apply) the transformed table at the minimum edit distance, which is the minimum total cost of table edit operations needed to transform v_t into v_{t+1} . When the walk reaches v_n , after opportunely pruning the walk, the consecutive traversed edges form the program.

Example 7 *Figure 1.12 shows the approach for the generation of the synthetic program in Foofah. The system takes in input a sample of the initial spreadsheet and a sample of the desired final table. The proposed approach allows the identification of the path in the top part of Figure 1.12, where each basic transformation operation allows obtaining the transformed table reported in the bottom part of the figure. So in this case, a synthetic program of four operations is generated (the operations on the yellow path).* □

1.4 Knowledge graph construction

Knowledge graphs (KGs) can be defined as a multi-relational graph of data for conveying real-world knowledge, where nodes represent entities and edges represent different types of relations [93]. KGs are becoming more and more relevant for the transformation of huge amounts of multidisciplinary and heterogeneous data extracted from a plenitude of heterogeneous sources into a more reusable network of entities. On top of this, different kinds of services can be offered to the user (recommendation systems, question answering, semantic search, decision making) by taking into account the relationships existing among the involved entities. When KGs are built relying on a given ontology, the quality of the source is higher because interoperability can be easier guaranteed [130]. An ontology is a formal description of the concepts used in a domain, the properties and the relationships of each concept, and restrictions on facts. In our context, many kinds of relationships can exist among ontology concepts that can be organized in an inheritance hierarchy. Moreover, we consider the possibility that mandatory properties can be specified for each ontology concept and this information can be exploited for the generation of the instances' identifiers.

The term “knowledge graphs” (KG) is often associated with the term “knowledge base” (KB) and many articles discuss and contrast them. A recent article on KGs [84] describes them as *graphs of data intended to accumulate and convey knowledge of the real world, whose*

²Each transformation is obtained by applying the operators defined in Potter's Wheel, some of which, e.g. “split” and “merge” operators, allow applying syntactic transformations to the input.

nodes represent entities of interest and whose edges represent potentially different relations between these entities. Similarly, the paper [58] defines a KG as a dataset having formal semantics that can contain different kinds of knowledge such as rules, facts, axioms, definitions, statements, and primitives. In several cases the two terms are used as synonyms or do not provide a clear distinction, however, a KB can be seen as a system that contains a KG and a reasoning engine [58].

RDF is the standard language for the representation of KGs by means of triples. A triple indicates that two entities (subject and object) are connected through a relation/property. The use of IRI (International Resource Identifier) for the identification of entities allows the integration of information from different sources. Moreover, by means of RDFS (Resource Description Framework Schema)[28] and OWL (Web Ontology Language)[152], the pre-defined vocabularies can be described at schema level for the representation of abstract relations, like classes (concepts), instances (objects), subsets, and properties. RDFS and OWL are the main description languages for an ontology.

KG construction is a challenging activity and many surveys present the issues and approaches that can be exploited for their generation ([168, 93, 91]). Early approaches rely on the manual creation of the KG by means of domain experts or through the collaboration of an open group of volunteers. The obtained KGs are of good quality with few or no noisy facts, but with high human efforts. Many approaches are currently under investigation for the automatic construction of KGs through the application of algorithms for enhancing the existing ones and for the automatic construction of new customized KGs by extracting and integrating knowledge from several kinds of data sources. Even if the quality can be reduced, its coverage and completeness are much higher. In order to reduce the noise that can occur in the KGs, different approaches have been proposed for knowledge extraction, knowledge fusion, and knowledge refinement [41]. Approaches for knowledge extraction aim to acquire useful entities, attributes and relationships from several data sources and to represent them in a normalized form. Fusion techniques are then applied to the generated triples for identifying multiple representations of the same real-world entities and for solving consistency issues among contradicting triples. Then, different refinement methods, like entity classification, link prediction, and anomaly detection can improve the quality of the constructed KG.

1.5 Concluding remarks

In this chapter, we have revised some of the most representative table understanding methods published after 2006. The analysis has primarily highlighted an increasing interest recently devoted to the table content interpretation. Anyhow, for all the investigated Hurst's

steps there has been a radical shift in the way the problems are approached mainly due to the change of the source formats (e.g. from ASCII or image formats to structured documents) and to the increasing success of DL methods, which seem robust with respect to weakly labelled, noisy, heterogeneous, incomplete, and ambiguous data.

Though a great deal of research work has been devoted to the problem of table understanding, some issues, reported below, are still open.

Many techniques have been proposed for the different steps of table understanding and common evaluation metrics have been employed for assessing their effectiveness. However, each technique assesses the computed results using specific evaluation criteria. Moreover, several methods either exploit private document collections, or randomly extract them from public datasets without publishing the list of the sampled documents, or crawl them from the web, without making the crawled data available or specifying the parameters used for the crawler. This raises the urgent need of generating a common benchmark for the objective comparison of techniques developed for different table understanding steps. In the context of table interpretation, the recent creation of the “Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab)” [94] is the first commonly available benchmark for comparing semantic annotation approaches.

Even if some ML methods have been recently presented to learn from user examples by ensuring the generalizability required to avoid overfitting, most of them do not yet deal with weakly labelled or dirty data. Moreover, only a few works exploit an incremental learning approach, where the user advice and corrections are used to generate novel training data, and/or transfer learning techniques, where the knowledge of models trained for similar tasks in a different domain is “transferred” to a novel domain where the labelled collections are limited. The use of these ML techniques in the context of table understanding are promising.

DL techniques are becoming more and more effective in several contexts. However, with regard to the problem of table understanding, where the generation of an exhaustive training set covering all the sample variability is often difficult, user-interaction systems such as the recently proposed [19], which provide a visual user-machine-interaction interface for allowing users to check and fix the semantic models computed by ML automatic annotations, are advisable since the user checks may be used to improve the model adaptability to novel contexts, and to increase its robustness to dirty data.

Chapter 2

Background

The purpose of this chapter is to introduce notations and notions that will be exploited throughout the thesis. Specifically, we introduce the concept of table that is extracted from a spreadsheet and the type system that we can use for the initial annotation of table columns and values. Then, we introduce the representation of an ontology and the knowledge graph that can be defined starting from it. Functions and procedures that will be exploited on the knowledge graphs are also introduced. Finally, we define and discuss the characteristics of the semantic description of a table. A semantic description is a graph that is used for the representation of the mapping between the table columns and the concepts of the ontology.

The chapter is organized as follows. Section 2.1 defines the table extracted from a spreadsheet and the type system used in the next chapters. Section 2.2 defines the concept of knowledge graph and ontology, which is used for the definition of a semantic description of a table, as detailed in Section 2.3

2.1 Table representation and type system

A table T extracted from a spreadsheet is a triple $\langle Col, Rows, Ann \rangle$, where Col denotes the list of column names $[col_1, \dots, col_j, \dots, col_m]$ (when available, otherwise the symbol $?$ is used to denote its absence), and $Rows = \{row_1, \dots, row_n\}$ is the set of table rows (each row row_i , $1 \leq i \leq n$, is a list of values $row_i = [val_{i,1}, \dots, val_{i,j}, \dots, val_{i,m}]$, one value for each column identified in the column schema).

For associating a type to each value and column of a table we consider the type system \mathcal{D} which is formed by *simple types*, *mixed types* and *union types*. *Simple types*, denoted \mathcal{S} , can be either basic types (e.g. integer, Boolean, decimal, date) or domain-specific

types (e.g. Social Security Number (SSN), VAT, currency, email, municipality name, province or metropolitan area, region, zip code). On top of the simple types, we consider *mixed types*, named \mathcal{M} , which are record-types associated with a set of patterns, and *union types*, denoted \mathcal{U} , for representing the occurrence of values of different types in the same column.

The peculiarity of the extracted table, differently from relational tables, is that for each $v_{i,j}$ and for each col_i more than one data type can be identified. We denote with $type(v_{i,j})$ the data type associated with the value $v_{i,j}$ and with $type(col_i)$, $1 \leq i \leq n$ the data types that can occur in a single column of the table T .

2.1.1 Mixed types

Mixed types are record-types that are associated with patterns. Indeed, in our tables, it is quite common to identify strings from which the same record structure can be extracted even if they are organized and expressed differently. For this reason, a set of patterns is associated with the record-type for extracting its components from the string.

A **pattern** is a sequence $[c_o, \{t_1\}, \langle c_1, Occ(c_1) \rangle, \dots, \{t_n\}, \langle c_h, Occ(c_h) \rangle]$, where c_o, \dots, c_h are terminal symbols, t_0, t_1, \dots, t_n are data types¹ or the symbol `void` and $Occ(c_1), \dots, Occ(c_h)$ are the occurrences of each terminal symbol that should not be considered terminals. The `void` symbol is introduced for skipping string parts whose content should not be maintained (e.g. the serial number of invoices).

Terminal symbols are non-empty strings (except for c_o and c_h which can be empty) that are used for separating values of simple types. A terminal symbol can be a blank space (represented as \emptyset), a hyphen, a parenthesis and so on, empty strings are denoted through Λ . The number of occurrences of each terminal symbol is required when it also appears in the previous non-terminal values. The case is described in the following example.

Example 8 Consider the string “Mary Ann Smith”. We need to create a pattern for splitting the string in the person’s name (i.e. “Mary Ann”) and the person’s surname (i.e. “Smith”). However, if we simply consider the blank space as a separator, we are not able to correctly split the string into two parts. For this reason, we consider the number of occurrences of the blank space to identify the correct occurrence of the terminal symbol. In this case, the pattern would be $[\Lambda, \{\text{name}\}, \langle \emptyset, 1 \rangle, \{\text{surname}\}, \langle \Lambda, 0 \rangle]$. \square

Thus, patterns are rules for extracting the same record-based value from heterogeneous strings expressing the same kind of information.

¹To distinguish terminal symbols from types, types are delimited by brackets.

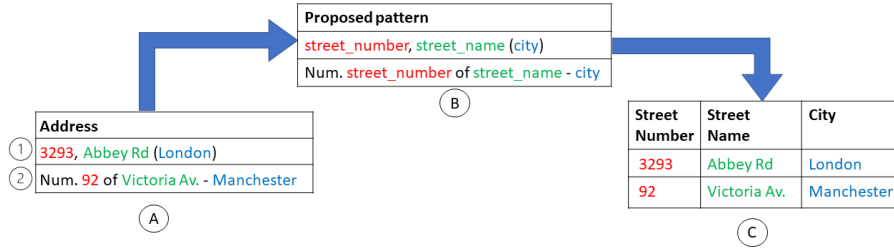


Figure 2.1: Mixed types and pattern examples

Example 9 Consider the strings in Figure 2.1 (A):

(1) "3293, Abbey Road (London)" (2) "Num. 92 of Victoria Av. – Manchester"

Both strings present the same record-type $\text{rec}(\text{streetNumber}, \text{streetName}, \text{cityName})$, however, the representation format of string (1) is very different from string (2). The identification of a pattern for each string allows the extraction of the sub-components in different positions. Figure 2.1 (B) reports the two patterns that we have developed:

$$[\Lambda, \{\text{streetNumber}\}, \langle ', 0 \rangle, \{\text{streetName}\}, \langle ', 0 \rangle, \{\text{cityName}\}, \langle ', 0 \rangle,]$$

$$['\text{Num.}', \{\text{streetNumber}\}, \langle 'of', 0 \rangle, \{\text{streetName}\}, \langle '-', 0 \rangle, \{\text{cityName}\}, \langle \Lambda, 0 \rangle]$$

All the patterns are applied on the strings of the column, and when one of them can extract the sub-components of the correct type (as shown in Figure 2.1 (C)), the values are shown according to the record-type (in this specific case as three columns). \square

We can now introduce the definition of mixed type. Let m be a system/user-defined name of the mixed type; $t_1, \dots, t_h \in \mathcal{S}$ be a set of simple types; and $\{p_1, \dots, p_k\}$ be a set of patterns; a mixed type is a triple $\langle m, \text{rec}(t_1, \dots, t_h), \{p_1, \dots, p_k\} \rangle$.

2.1.2 Union types

Union types are used for representing the occurrence of instances of different types in the same column. Formally, let $\{\bar{t}_1, \dots, \bar{t}_l\} \subseteq \mathcal{S} \cup \mathcal{M}$ be a set of simple or mixed types, $\text{union}(\bar{t}_1, \dots, \bar{t}_l)$ is a union of data of different types. The components of a union type can be simple types, mixed types or a combination of both.

Example 10 The column SSN/VAT in Figure 2.2 (A) contains occurrences of two simple data types: SSN (Social Security Number) and VAT (Value Added Tax), thus the type for the entire column is $\text{union}(\text{SSN}, \text{VAT})$. By contrast, $\text{union}(\text{mixed}_1, \text{companyName})$ is the type of column Name/Company, where mixed_1 is $\langle \text{mixed}_1, \text{rec}(\text{name}, \text{surname}) \rangle$,

Name/Company	SSN/VAT
Peter Arnold	CE 06 91 43 C
Robert Morley	RA 98 07 81 A
Alice Simons	RT 92 80 15 D
Collier Inc	14-5537560
James Green	MR 75 33 03 A
Barbara Howard	NM 00 98 06 B
Glover Inc	64-8163968

A

Address
3293, Abbey Rd (London)
Victoria Avenue; Num. 92 - UK
8-10 Silver St, EH42BS
13A, Stafford Rd (Newport)
4300 Maple Lane, 35901
343, Oxford Rd (Reading)
Church Rd; Num. 310 - UK
2918 Green Road, 22304

B

Figure 2.2: Union type and union of mixed types columns

$[\Lambda, \{\text{name}\}, \langle \emptyset, 0 \rangle, \{\text{surname}\}, \langle \Lambda, 0 \rangle]$. Figure 2.2 (B) shows a column containing occurrences of mixed types that follow different patterns. Three patterns can be identified:

- $mix_1: [\Lambda, \{\text{streetNumber}\}, \langle ', 0 \rangle, \{\text{streetName}\}, \langle '(, 0 \rangle, \{\text{cityName}\}, \langle ')', 0 \rangle]$
- $mix_2: [\Lambda, \{\text{streetName}\}, \langle '; Num.', 0 \rangle, \{\text{streetNumber}\}, \langle '- ', 0 \rangle, \{\text{countryInitials}\}, \langle \Lambda, 0 \rangle]$
- $mix_3: [\Lambda, \{\text{streetNumber}\}, \langle \emptyset, 0 \rangle, \{\text{streetName}\}, \langle ', ', 0 \rangle, \{\text{ZIP}\}, \langle \Lambda, 0 \rangle]$

Therefore, $\text{type}(\text{Address})$ is $\text{union}(mix_1, mix_2, mix_3)$. □

2.2 Ontology and knowledge graphs

A domain ontology O [156] contains a set of concepts $\mathcal{C} = \{C_1, \dots, C_n\}$ and relationships $\mathcal{R} = \{(C_1, r, C_2) \mid C_1, C_2 \in \mathcal{C}, r \in R\}$, where R is the set of relation names. Concepts can be organized in an inheritance hierarchy: $C_1 \sqsubseteq C_2$ denotes that C_1 is a sub-concept of C_2 (and also that C_2 is more general than C_1). Each concept can have associated basic properties taken from a set $\mathcal{P} = \{(p_1, D_1), \dots, (p_m, D_m)\}$, where $\mathcal{D} = \{D_1, D_2, \dots\}$ is the set of basic types whose values can be used for the corresponding properties and $\{p_1, \dots, p_n\} \subseteq P$ are the property names; the properties of concept C are denoted $\mathcal{P}(C)$ and include those specifically specified for C and those inherited from the more general concepts. Some properties can be used for uniquely identifying an instance of a concept. Each concept has a property id that is used for representing the identifier of its instances.

Example 11 Figure 2.3 shows an excerpt of the DBpedia² for the representation of films,

²www.dbpedia.org

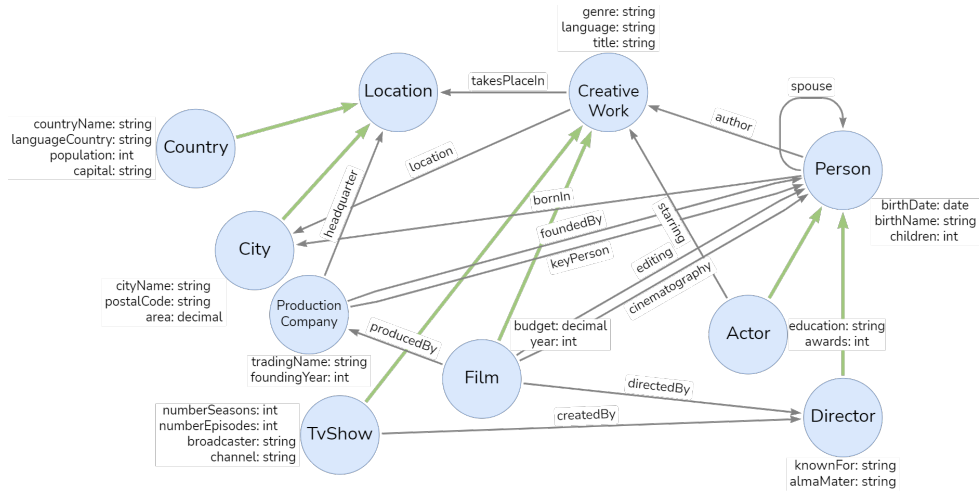


Figure 2.3: Excerpt of the dbpedia ontology

TV series, the locations where they were filmed, and their casts. Light blue circles represent concepts (e.g. Person, City, Film), light green arrows denote the subclass relationships (Film and TVShow are subclasses of CreativeWork), and straight arrows denote relationships among concepts (e.g. directedBy). Pairs (name : D_i) represent property names with their domains and are reported closer to the corresponding concepts (e.g. CreativeWork has the properties genre, language and title of type string). \square

Given concept C , we introduce the following functions: *ancestors*, *dist*, *closure*, and *classes*. $ancestor_O(C)$ is the set of concepts \bar{C} in O such that $C \sqsubseteq \bar{C}$ (i.e. the concept C itself and all its ancestors according to the \sqsubseteq relationship); given $C \sqsubseteq \bar{C}$, $dist(C, \bar{C})$ is the length of the path in the inheritance relationships between the two concepts, that is $dist(C, C) = 0$, $dist(C, \bar{C}) = k$, when there exist k distinct concepts in O such that $C \sqsubseteq C_1 \dots \sqsubseteq C_k \sqsubseteq \bar{C}$; $closure_O^\Delta(C)$ represents the set of relationships in which the concept C or one of its ancestors is involved directly ($\Delta = 1$) or using paths of length Δ . $closure_O^\Delta(C)$ is inductively defined as follows:

- $closure_O^1(C) = \{(C_1, r, C_2) | C_1 \in ancestor_O(C), (C_1, r, C_2) \in \mathcal{R}\} \cup \{(C_1, r, C_2) | C_2 \in ancestor_O(C), (C_1, r, C_2) \in \mathcal{R}\};$
- $closure_O^\Delta(C) = closure_O^{\Delta-1}(C) \cup \{(C_2, r_2, C_3) | (C_2, r_2, C_3) \in \mathcal{R} \setminus closure_O^{\Delta-1}(C), \exists (C_1, r, C_2) (or (C_2, r, C_1)) \in closure_O^{\Delta-1}(C) s.t. C_1 \neq C_3\} \cup \{(C_3, r_2, C_2) | (C_3, r_2, C_2) \in \mathcal{R} \setminus closure_O^{\Delta-1}(C), \exists (C_1, r, C_2) (or (C_2, r, C_1)) \in closure_O^{\Delta-1}(C) s.t. C_1 \neq C_3\}$

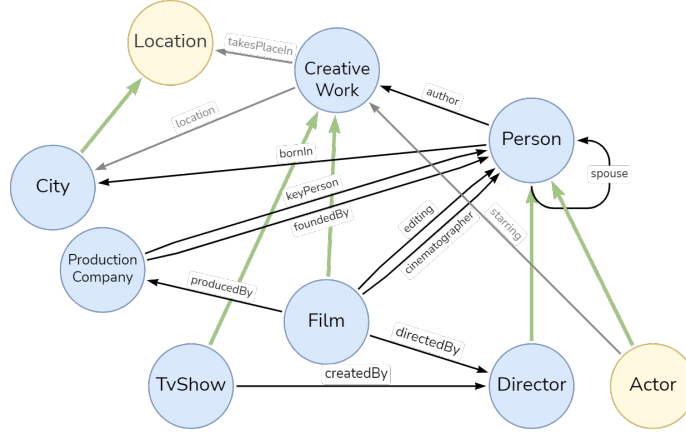


Figure 2.4: Closure of the concept Director at $\Delta = 1$ and at $\Delta = 2$

Example 12 Figure 2.4 shows $closure_{\mathcal{O}}^2(\text{Director})$ in the ontology in Figure 2.3. Black edges and light blue nodes denote the direct relations with Director, whereas grey arrows and the yellow nodes represent the relations at $\Delta = 2$. For the sake of readability, inheritance relations are reported with a thick light green arrow, but they do not belong to the closure. Note that Creative Work, Production Company, and City are introduced because a relation exists with the concept Person (a generalization of Director). \square

Through the *closure* function, we can include in the semantic descriptions also instances of concepts for which none of their properties present values in the considered tabular data. However, their introduction could be relevant for the identification of relations among the concepts and also for identifying semantic annotations for unmatched columns.

Given a set of concepts $\{C_1, \dots, C_n\}$, function $classes_{\mathcal{O}}^{\Delta}(\{C_1, \dots, C_n\})$ identifies all the concepts that are involved in relationships obtained by the application of the $closure_{\mathcal{O}}^{\Delta}$ on each of the concepts in $\{C_1, \dots, C_n\}$. By denoting with A_i the i^{th} component of the triples in the set A , *classes* are defined as follows:

$$classes_{\mathcal{O}}^{\Delta}(\{C_1, \dots, C_n\}) = \bigcup_{i=1}^n closure_{\mathcal{O}}^{\Delta}(C_i)_{|1} \cup closure_{\mathcal{O}}^{\Delta}(C_i)_{|3}$$

A knowledge graph is a heterogeneous graph KG where nodes represent instances of the ontology concepts and edges represent relationships among them. Basic properties can be associated with the concepts. Formally, a knowledge graph KG is an n-ple $KG = (V, E, R, Prop)$, where, $V = \{v_1, \dots, v_n\}$ is a set of nodes, E is a set of triples (i.e. $E \subseteq V \times R \times V$), R is the set of relation names, and $Prop \subseteq V \times 2^{P \times D}$ are the properties with the corresponding values associated with the concepts. With $\mathcal{I}(C)$ we denote the nodes

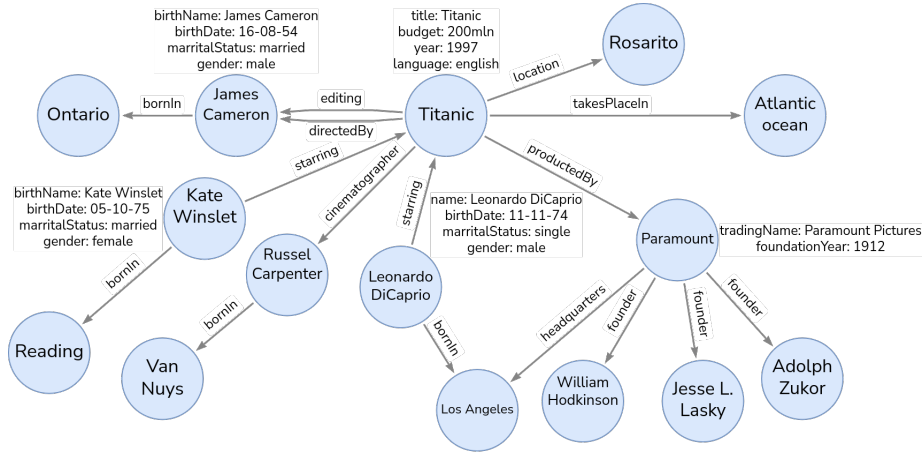


Figure 2.5: A KG sample adhering to the constraints imposed by our ontology

in KG that represent a given concept C , and with $E(r)$, $r \in R$, the triples involving the relation r . Moreover, given a property $p \in P(C)$, $\mathcal{I}_p(C)$ denotes the nodes in $\mathcal{I}(C)$ that present the property p . We denote with $|\cdot|$ the cardinality of a set.

Example 13 Figure 2.5 shows an excerpt of the knowledge graph KG corresponding to the ontology introduced in Figure 2.3. It represents information about the film “Titanic”, the actors starring in it, the director, the production company and the locations. The `id` property associated with each concept is reported in the circle representing the concept and should contain the IRI. For the sake of readability, in our examples, we report a human-readable string that identifies the node. \square

2.3 Semantic description

A semantic description for a table T is a graph SD containing two kinds of nodes. Nodes that represent the columns in T and nodes representing *meta-instances* of the concept of the ontology. A meta-instance is a generic instance of the ontology concept and more than one can appear within SD . In order to distinguish among the different meta-instances, the index/superscript j is used for the node identifier ($j = 0$ means the first instance of the concept and can be omitted). Formally, a semantic description for a table $T = \langle Col, Rows, Ann \rangle$ is a graph $SD = (U_{C_s}, U_T, E_R, E_T)$, where:

- U_{C_s} is a set of nodes representing meta-instances of the concepts in \mathcal{C} ; $u_C^j \in U_{C_s}$ denotes a vertex corresponding to the j^{th} occurrence of the concept C ;
- U_T is a set of nodes corresponding to the columns in T ($|U_T| \leq |Col|$);

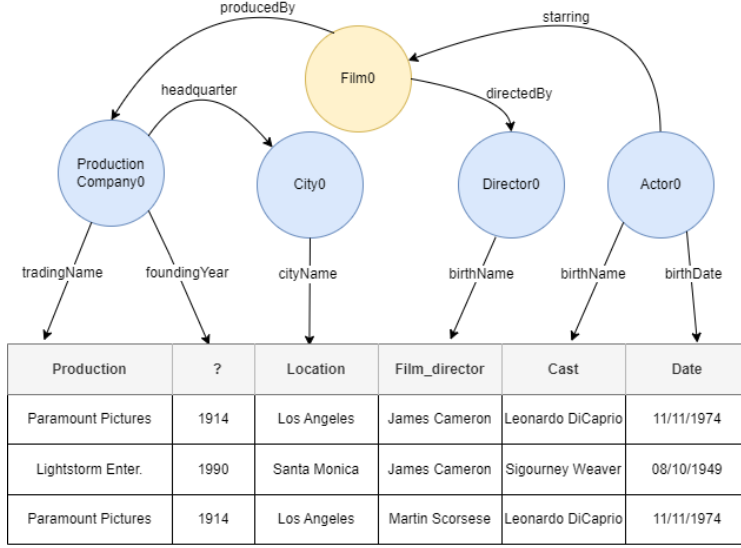


Figure 2.6: Example of SD generated on top of a table.

- $E_R \subseteq U_{Cs} \times R \times U_{Cs}$ represents the relationships among concepts in U_{Cs} ;
- $E_T \subseteq U_{Cs} \times P \times U_T$ denotes the properties associated with the columns of T .

Example 14 Figure 2.6 shows a semantic description SD for a table that contains details about a list of films, such as the production company, its founding year and headquarter city, the name and birth date of the starring actor and the name of the director. Without the semantic description, we are not able to establish if the location is the headquarter of the production company, the place of birth of the director or the one of the actor. Through the links between the concepts, the kinds of relationships that bind the concepts of the table are made explicit. Note that the meta node u_{Film}^0 has been introduced that is missing in the table but it is mandatory (according to the used ontology) for creating a connection between the three main concepts (production company, actor, and director). From a formal point of view, SD is a graph with the meta-nodes $u_{\text{PC}}^0, u_{\text{City}}^0, u_{\text{Film}}^0, u_{\text{Director}}^0, u_{\text{Actor}}^0$ (where PC stands for Production Company), and 6 terminal nodes $u_{T0}, u_{T1}, \dots, u_{T5}$ representing the columns of the table (the number represents the position of the column in the table). Edges $((u_{\text{PC}}^0, \text{tradingName}), u_{T0})$ and $((u_{\text{Director}}^0, \text{birthName}), u_{T3})$ are samples of terminal edges in E_T , whereas $((u_{\text{PC}}^0, \text{headquarter}), u_{\text{City}}^0)$ and $((u_{\text{Actor}}^0, \text{starring}), u_{\text{Film}}^0)$ are samples of relations in E_R . \square

Meta-instances occurring in SD can be classified into *specific* or *variable* nodes. Specific nodes are associated with identifying properties occurring in the table. Therefore, they

correspond to a single real-world entity. Suppose, for example, that `year` and `title` are identifying properties for the concept `Film`. If values are provided for these two properties in a table, a single instance of the concept `Film` can be identified. The variable nodes, by contrast, are not associated with identifying properties in the table. For example, consider the concept `Date` and suppose that only the property `year` is specified for it. In this case, a specific date cannot be identified, because the `day` and `month` are missing.

Chapter 3

Semi-automatic type inference approach

Spreadsheets are largely used and are a fundamental tool for organizing and classifying data of different types into a logical format. Users model the data in the most convenient way for their business and the consequence is that a common representation format is missing and the possibility of the integration of their content with other resources is hard to realize. A spreadsheet can contain data that are not in a tabular form, such as titles, descriptions, aggregated columns, and footers; moreover, the content can be allocated within each cell in arbitrary ways. For example, a domicile address can be divided into three columns containing street name, odonym and zip code, or it can be inserted in a single cell.

In this chapter, we describe our approach for identifying and extracting a table within a spreadsheet, removing the extra information, and inferring a data type for each data cell and column. Since each value and each column of the table can be assigned to more than one data type, we model the problem as a multi-label classification approach and exploit a simple extension of decision trees with pre-defined thresholds that automatically identify possible types that can be assigned to a column. The model has been trained through the use of synthetic data based on real spreadsheets. This process has also the effect of identifying cells presenting errors because their values are not compliant with the type identified for the corresponding column. Furthermore, we describe the graphical interfaces that we have developed for supporting the user in the adjustment of the inferred types.

The chapter is organized as follows: Section 3.1 outlines the approach for the extraction of a table from a spreadsheet. Section 3.2 introduces simple type recognizers that are exploited in Section 3.3 for predicting the type of cells and columns. Section 3.4 presents our graphical user interfaces. Experimental results are discussed in Section 3.5. Finally, Section 3.6 reports some concluding remarks on this phase of our work.

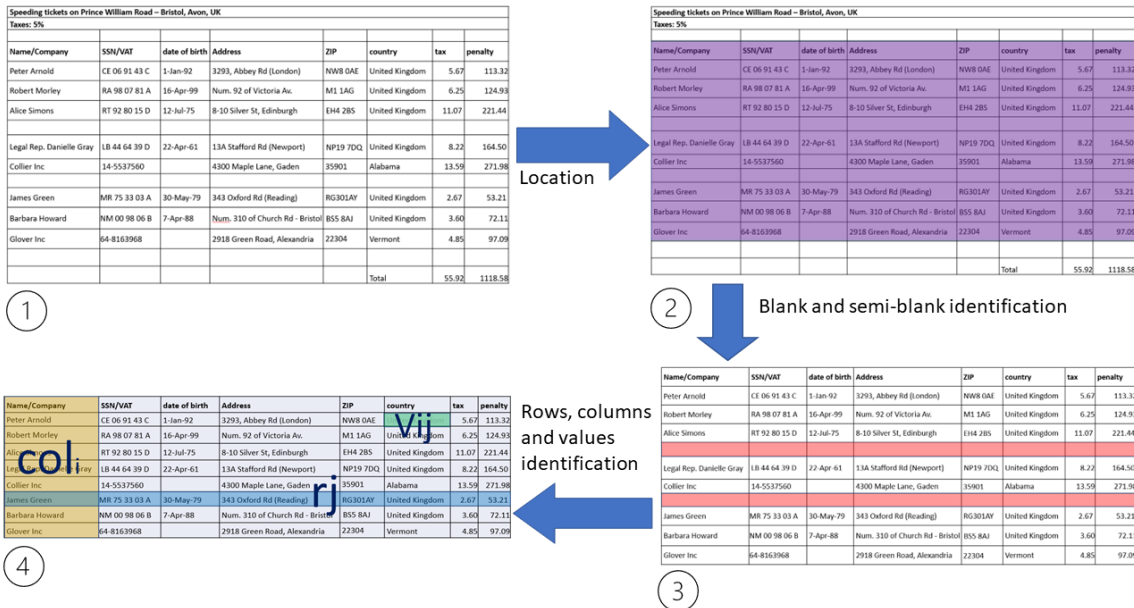


Figure 3.1: Table extraction

3.1 Table identification

In our scenario, data are organized in spreadsheets (according to CSV, TSV, and Excel formats). Within each spreadsheet a table can be located, it is usually formed by the list of column names (*table schema*), and the *rows* containing values for each column. Sometimes, a group of data rows are separated by *blank rows*, in some cases for aesthetic reasons while in others for grouping data representing the same concept. Moreover, *semi-blank rows* can be used for reporting grand totals or remarks about an invoice.

Figure 3.1 shows the table extraction process. Starting from a raw table, depicted in (1), the table highlighted in (2) is extracted. Therefore, headers, footers, and empty columns (when occurring) are removed. Moreover, as shown in (3), *blank* and *semi-blank* rows are deleted because their content is not relevant in our context. Even when *blank rows* are used to separate groups of rows having a common semantic, this additional information belongs only to human knowledge and it is hard to identify, use and understand it, in an automatic and machine-readable way. For the localization of the table, we consider the density of the information contained in the table w.r.t. external data. For removing *blank* and *semi-blank* rows, predefined thresholds are used on the minimal number of non-empty cells in each row w.r.t. the number of columns. The final table obtained, reported in (4), contains only the essential data and is structured in rows, columns (whose content is addressed by the

column schema), and cells. The distinction between table headers and cells is determined by the position of the rows (when the headers are present they are in the first row) and the use of dictionaries containing the headers of the already processed spreadsheets.

3.2 Type recognizers

For identifying the values v that adhere to the constraints of a given type, each simple type $t \in \mathcal{S}$ is associated with a different type recognizer. A *recognizer* Rec_t for a type t is a function that, given a value v , determines if v is a valid instance of t or not. Two categories of type recognizers have been devised:

- *Exact recognizers*: a set of well-defined rules for identifying a data type that returns a Boolean result, `true` if the given value is an instance of the type identified by the recognizer, `false` otherwise. This type of recognizer can introduce false negatives (e.g. the type `year` can be recognized as a `zip code`).
- *Approximate recognizers*: beside considering well-structured values, these recognizers are tolerant to malformed values. In this case, the answer is a probability that can be interpreted as `true` above a given threshold and `false` below it. The approximate recognizers have been introduced for managing conversion problems and typing errors occurring in the data (e.g. if a VAT is shorter than usual because during a format conversion a set of 0 at the beginning has been removed). Using these types of recognizers, it is possible to introduce false positive elements since a value can erroneously be recognized as a valid instance of a type.

Both approaches present the problem of false positives or negatives. These problems can be minimized by checking the frequency of that type in the column and by setting a threshold to determine whether the recognition is correct. Depending on the application domain and the data types, we adopt the exact or approximate approach for each recognizer. Four types of recognizers have been identified:

- *set*: this kind of recognizer can be used when the instances of a type t are finite and enumerable. In this case, $v_{i,j}$ belongs to a type t when $v_{i,j}$ belongs to the set of valid values for t . This kind of recognizer has been applied for the types `country`, `region`, `municipality`, `zip code`, `Boolean`, `gender`, `name`, `surname`
- *regular*: these recognizers are used when the type instances can be determined through regular expressions. For example, for the recognition of Italian vehicle plates, more than 20 regular expressions have been developed for the identification of the different variations (e.g. `^[a-zA-Z]{2}\\s?[0-9]{3}\\s?[a-zA-Z]{2}\\$` is an example of the regular expressions that we have developed).

- common: this category is referred to the common types that can be recognized using a pre-defined package, usually there are several packages available online for their recognition. Specifically, we have adopted python code for checking emails, SSN, VAT, addresses, integer, decimal, year, empty strings.
- custom: the recognition of data types can be extended by defining a custom recognizer, however, in our use case it was not necessary.

During the recognition of the type of a value $v_{i,j}$, the occurrence of a value of another type in the same row is sometimes considered and exploited to increase the strength of the prediction. For some types, $t_1, t_2 \in \mathcal{S}$, the occurrence of a value of t_1 along with a value of t_2 is used for identifying the third value (e.g. the occurrence of street name and city can lead to the identification of a zip code).

Recognizers have been also generated for the identification of some mixed types. For extracting name and surname from a value $v_{i,j}$, the value is split in a list of terms $[w_{i,j}^1, \dots, w_{i,j}^s]$ by considering the blank space and some usual terminal symbols. Then, a recogniser based on sets is used by considering a collection of common names and surnames. For the recognition of the address components, we have used the *libpostal* library¹ which is frequently used in the context of Natural Language Processing and exploits the OpenStreetMap (OSM) database of location names. Finally, a recognizer that exploits a collection of patterns extracted from common strings used in our domain has been developed. A value $v_{i,j}$ is compared with each of the available patterns and whenever a single match is identified, $v_{i,j}$ is considered an instance of the mixed type associated with the pattern.

3.3 Type inference approach

Due to the abundance of the simple types of our type system and the errors that can occur in real data, a value $v_{i,j}$ of a table $\langle S, V \rangle$ can belong simultaneously to different types or none of them. This requires the use of an ML approach for inferring the most likely type for a single value $v_{i,j}$ and also for the entire column. We have compared different ML models and ended up using decision trees for the identification of the type of a single value $v_{i,j}$. The developed technique is a simple multi-label classification approach for inferring the types to be associated with the column. Indeed, in our type system, a column can be associated with a union type and therefore the same column can contain values of different types. In the remainder, we first present the adopted model for inferring the type of each single $v_{i,j}$ and the type of the corresponding column and compare it with other ML models. Then, we discuss the approach adopted for training the model.

¹<https://github.com/openvenues/libpostal>

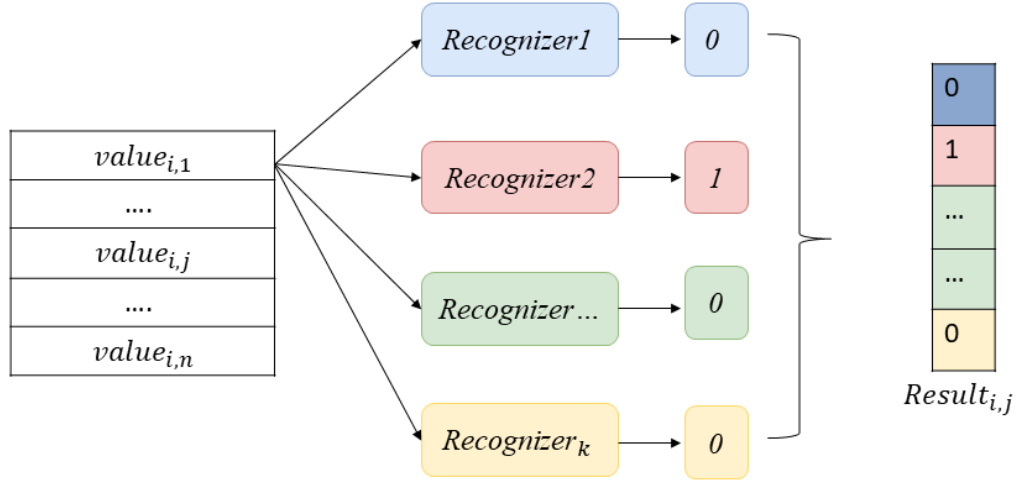


Figure 3.2: Recognizers compute the DT features for each value of a column k of the table

3.3.1 The main model

To determine the type to be associated with every single value and each column of the table, we adopted an ensemble of recognizers that feed their results into a decision tree.

The entire set R of type recognizers $Rec_1, Rec_2, \dots, Rec_R$ defined for the type system is applied to each value $v_{i,j}$ of a given column k , generating a vector of R Boolean values for each $v_{i,j}$, as shown in Figure 3.2. The Boolean results are inserted into a vector that represents the feature embedding for the value $v_{i,j}$.

$$\mathbf{Pr}_{i,j} = \langle \Pr_{i,j,1}, \Pr_{i,j,2}, \dots, \Pr_{i,j,R} \rangle \quad (3.1)$$

Moreover, each type recognizer Rec_r computes its average value $\overline{\Pr}_{j,r}$ across a given column k , as shown in Figure 3.3, formally:

$$\overline{\Pr}_{j,r} = \frac{1}{n} \sum_{i=1}^n Rec_r(v_{i,j}) = \frac{1}{n} \sum_{i=1}^n \Pr_{i,j,r} \quad (3.2)$$

The generated result is a vector of average values of the recognizers for the column k :

$$\overline{\Pr}_k = \langle \overline{\Pr}_{k,1}, \overline{\Pr}_{k,2}, \dots, \overline{\Pr}_{k,R} \rangle \quad (3.3)$$

In this way for each value $v_{i,j}$ we can construct a vector $\mathbf{Pr}_{\text{conc}}^{i,j} = \langle \overline{\Pr}_k, \mathbf{Pr}_{i,j} \rangle$ simply concatenating the vectors of Eq. 3.1 and Eq. 3.3. The list of vectors $\mathbf{Pr}_{\text{conc}}^{i,j}$ is the input for training the Decision Tree to predict the type of the value $v_{i,j}$, as shown in Figure 3.4.

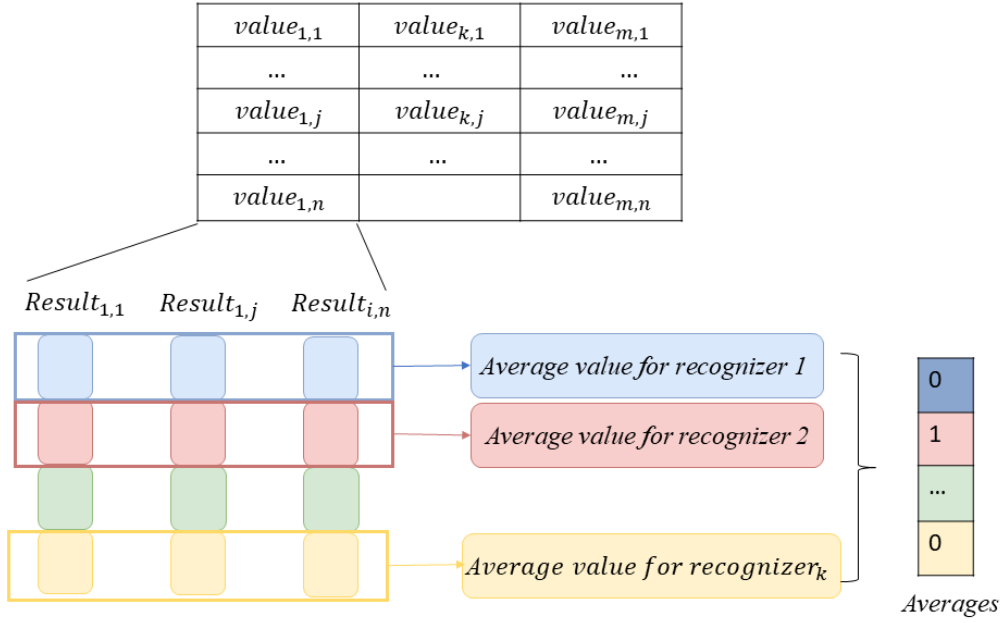


Figure 3.3: Each recognizer Rec_R evaluates the average value for the entire column

Starting from the types predicted for each value contained in the column j , named $Type_{i,j}^{val}$ ($1 \leq i \leq n$), we express as $PType(j)$, the set of potential types. This set contains pairs (t, o_t) , where o_t is the number of occurrences of type t in column j . Then, by considering a predefined threshold σ , representing the minimal allowed frequency of value types in a column, the type of the column j is determined as follows:

$$Type_j^{col} = \{t | (t, o_t) \in PType(j) \wedge \frac{o_t}{\sum_{(\bar{t}, \bar{o}) \in PType(j)} \bar{o}} \geq \sigma\} \quad (3.4)$$

$Type_j^{col}$ can contain a single type or the members of the union type to be associated with the column j . The values whose type does not belong to this set and are not empty, are marked as errors for this column and the user can check and fix them using the graphical user interfaces discussed in the next section.

Example 15 Consider the column 3 (date of birth) of the CSV in Figure 3.5 that contains the values $\langle '01-01-92', '16-04-99', '12-07-75', '22-04-61', ', ', '300579', '17-04-1988', ' \rangle$ and suppose we wish to identify the type of $v_{1,3} = '01-01-92'$ and $v_{6,3} = '300579'$. For the sake of simplicity, consider the presence of only the following type recognizers: Rec_{string} , Rec_{date} , Rec_{number} . In this case $\mathbf{Pr}_{1,3} = \langle 1, 1, 0 \rangle$ and $\mathbf{Pr}_{6,3} = \langle 1, 0, 1 \rangle$. Moreover, $\bar{\mathbf{Pr}}_3 = \langle 0.428, 0.428, 0.142 \rangle$ by taking into account that the number of rows is 7. Supposing a

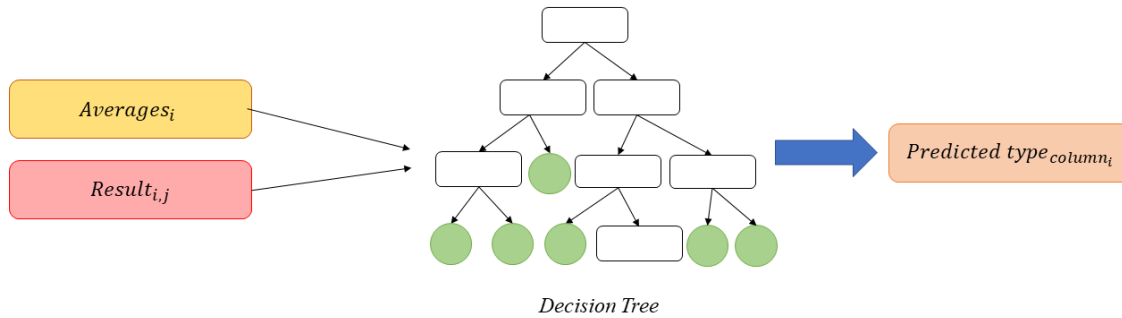


Figure 3.4: Type prediction with decision tree

Name/Company	SSN/VAT	date of birth	Address	ZIP	country	tax	penalty
Peter Arnold	CE 06 91 43 C	1-Jan-92	3293, Abbey Rd (London)	NW8 0AE	United Kingdom	5.67	113.32
Robert Morley	RA 98 07 81 A	16-Apr-99	Num. 92 of Victoria Av.	M1 1AG	United Kingdom	6.25	124.93
Alice Simons	RT 92 80 15 D	12-Jul-75	8-10 Silver St, Edinburgh	EH4 2BS	United Kingdom	1107	221.44
Legal Rep. Danielle Gray	LB 44 64 39 D	22-Apr-61	13A Stafford Rd (Newport)	NP19 7DQ	United Kingdom	8.22	164.50
Collier Inc	14-5537560		4300 Maple Lane, Gaden	35901	Alabama	13.59	271.98
James Green	MR 75 33 03 A	300579	343 Oxford Rd (Reading)	RG301AY	United Kingdom	2.67	53.21
Barbara Howard	NM 00 98 06 B	17-Apr-88	Num. 310 of Church Rd - Bristol	BS5 8AJ	United Kingdom	3.60	72.11
Glover Inc	64-8163968		2918 Green Road, Alexandria	22304	Vermont	4.85	97.09

Figure 3.5: Example of table

threshold $\sigma = 0.4$, the decision tree for $v_{1,3}$ returns date, whereas for $v_{6,3}$ returns error. Consider now the vector $\langle \text{date, date, date, date, empty, error, date, empty} \rangle$ of types predicted for column 3. The type for column 3 is date with the presence of an error. \square

3.3.2 Training of the main model

The data set available in our application context contains errors and is not labelled (manual labelling of these data would require too much time) and the quantity of the data themselves is too low for training even simple models (the available documents are around 200). For these reasons, we decided to generate synthetic $\langle \bar{S}, \bar{V} \rangle$ tables similar to the original ones. For creating these tables, a dataset has been realized for each simple type we are interested in. Then, we generated documents with a structure similar to the original ones and presenting values extracted from these datasets. The generated documents have a variable number of columns and rows for simulating the real ones. Specifically, in each row, data are correlated as in a real one: for example, the ZIP code, as well as the first digits of the SSN, are generated using the geographical area in the same row. For making more realistic

the generated documents, also empty values and errors have been added to the synthetic tables. Empty values have been included by randomly replacing some values (e.g. "-", "_", "." and other kinds of spaces and tabs) with `empty`. Some of the values automatically generated have been replaced with the wrong values for the column type. For instance, values of type `surname` can be placed in a column presenting values of type `ZIP`. However, we avoided the introduction of errors in columns presenting too similar values (like `ZIP` code and `integer`) to avoid misclassifications. Each value is finally labelled with its type, the label `empty` in case of empty values, or the label `error` in case of error.

To identify the validity of the decision tree for the prediction task of this setting, we have considered also a Random Forest composed of 500 decision trees, each one using the same parameters of the single decision tree, and an MLP composed of 3 dense layers (64, 32 and 1 neurons respectively), with activation `ReLU` except for the last layer which uses a `Sigmoid` activation. The optimizer used for the MLP model was `Nadam`. Python `Scikit-learn` [132] libraries have been used to implement DTs and random forests. `Keras/TensorFlow` [1] have been used for implementing MLP.

To train the considered models, we generated 1 million synthetic tables randomly split 8 times in 100.000 documents for training and 200.000 for the validation set. It must be noted that since these tables are generated synthetically using a simulation, they cannot be considered independent and identically distributed, but just an approximation. The Decision Tree was trained using a maximal depth of 20, using the Gini criterion and with class imbalance aware weights.

The average `AUPRC` and `AUROC` scores of the models were, respectively 0.98 and 0.995 on the validation set. The scores were then compared using a `Wilcoxon` signed-rank test, with a `p-value` threshold of 0.05. The test did not display any statistically significant difference between the different compared methods (i.e. Decision Trees, Random Forests and MLPs). Since the decision tree is the fastest to train and the most interpretable, we have chosen it.

3.4 Visualization and type adjustment

The approach so far presented is liable to errors, mainly because the application domain is very complex, and thus there can be syntactic errors that can not be automatically corrected. Moreover, the identification of the sub-components of mixed types following peculiar patterns can be problematic, and, since the ML approach is probabilistic, some predictions can be wrong. Finally, data related to the same subject can be divided into subsequent rows, such as in Figure 3.5, where the society “Collier Inc.” is legally represented by the physical person “Danielle Gray” whose data are reported in the above row, therefore, information related to the same concept can be spread in different rows of the table.

#	Name/Company	Identifier	date of birth	Address	ZIP code	Country	tax	penalty
	mixed1, text	SSN/VAT	date,error	mixed1, text	ZIP, error	country	decimal	decimal
1	Peter Arnold	CE 06 91 43 C	14/01/1992	3293 Abbey Rd London	NW8 0AE	United Kingdom	5.67	113.32
2	Robert Morley	RA 98 07 81 A	16/04/1999	Num. 92 of Victoria Avenue - Manchester	M1 1AG	United Kingdom	6.25	124.93
3	Alice Simons	RT 92 80 15 D	15/07/1975	8-10 Silver St, Edinburgh	EH4 2BS	United Kingdom	1107	221.44
4	Legal Rep. Danielle Gray	LB 44 64 39 D	22/04/1961	13A Stafford Rd Newport	NP19 7DQ	United Kingdom	8.22	164.5
5	Collier Inc.	14 - 5537560		4300 Maple Lane, Gaden	35901	Alabama		
6	James Green	MR 75 33 03 A	300579	343 Oxford Rd Reading	RG301AY	United Kingdom	2.67	53.21
7	Barbara Howard	NM 00 98 06 B	27/04/1988	Num. 310 of Church Rd - Bristol	BS5 8AJ	United Kingdom	3.6	72.11
8	Glover Inc.	64-8163968		2918 Green Road, Alexandria	22304	Vermont	4.85	97.09

Figure 3.6: Extracted table with associated types

For these reasons, different Graphical User Interfaces (GUIs) have been developed for supporting the user in handling and fixing the results obtained through the ML algorithms. The interfaces can be classified according to their functionalities in four types. The first one is used for presenting the result of the ML algorithm and for supporting the user in identifying errors occurring in the table. The second one allows the editing of values and types of each column and the application of modifications. The third kind of interface is used for the identification and management of mixed types, i.e. the assignment of a semantic description for each sub-component of a string, and the application of the description to similar values. Finally, the last kind of interfaces provides the user with a set of utilities for defining correlations among consecutive rows. In the remainder, we describe their characteristics.

3.4.1 Main interfaces and error identification

Figure 3.6 shows the main interface we have developed for reporting the result of the application of the ML approach described in Section 3.3 to the table reported in figure 3.5. Each cell and column is associated with the predicted data type.

In the resulting table, the first line reports the column schema and it is followed by a drop-down menu containing the inferred types for each column. If more than one type is reported in a single column, this means that each type is a member of a union type; different background colours are used to distinguish their instances, such as in the second column of the table in Figure 3.6 where the occurrences of SSN and VAT are distinguished using two different shades of green. Similarly, the presence of mixed types is represented using different text colours for each component of the pattern, while terminal symbols are hidden. If a column presents data of the same type, the background remains white, if a value is missing,

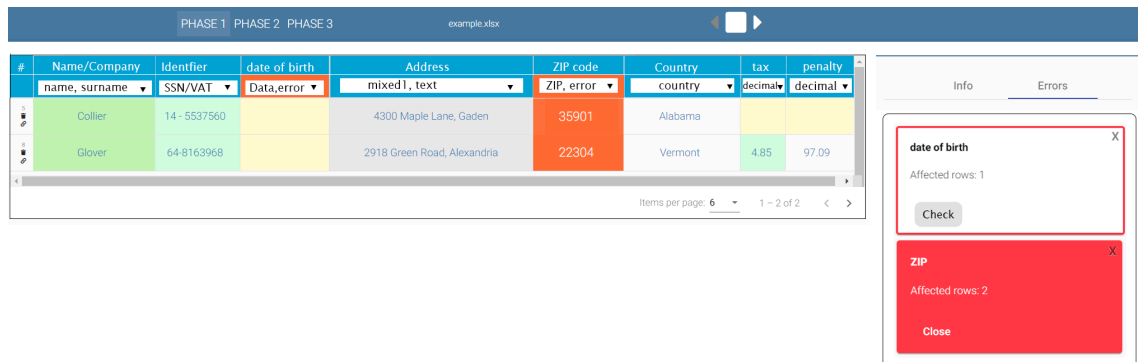


Figure 3.7: Error panel in the Web application

the cell background colour is yellow. The usage of different colours can help the users in the process of checking the type predictions and the empty values (in some cases a value should be provided) and performing error corrections. A cell is considered an error when the type of the contained value is not compliant with the types identified for the column; in this case, the cell background colour and the column type background are marked in red. Facilities are provided for showing only rows presenting values in a column of a given type (this is important for checking that values are correctly typed). Moreover, it is possible to remove rows from the table.

When the number of rows contained in a document is high, it can be difficult to detect all the red cells; for this reason, we provide an error panel, on the right part of the screen, that summarizes the issues that need to be solved. An example of the panel is shown in Figure 3.7. When the user clicks the check buttons on one of the tabs that are present in the panel, only the rows presenting the error are shown in the main interface. Once the errors are removed from the rows, the corresponding tab is removed from the panel.

3.4.2 Data type modification

Since our ML approach can produce false positives or negatives (e.g. a ZIP code that has been labelled as Integer), we decided to develop GUIs for supporting the user in modifying the predicted types and easily applying the modification to the entire column or subset of cells. In the modification process, the user should be supported in the specification of domain-specific types instead of the basic ones that can be obtained through the ML algorithms. Indeed, users can easily identify the types of columns and, in their modification activity, can introduce further knowledge to our Web application. The user is also supported in the modification of the value of a cell when it contains errors. Consider for example the red cell in the date of birth column in Figure 3.6. It contains a value not compliant with

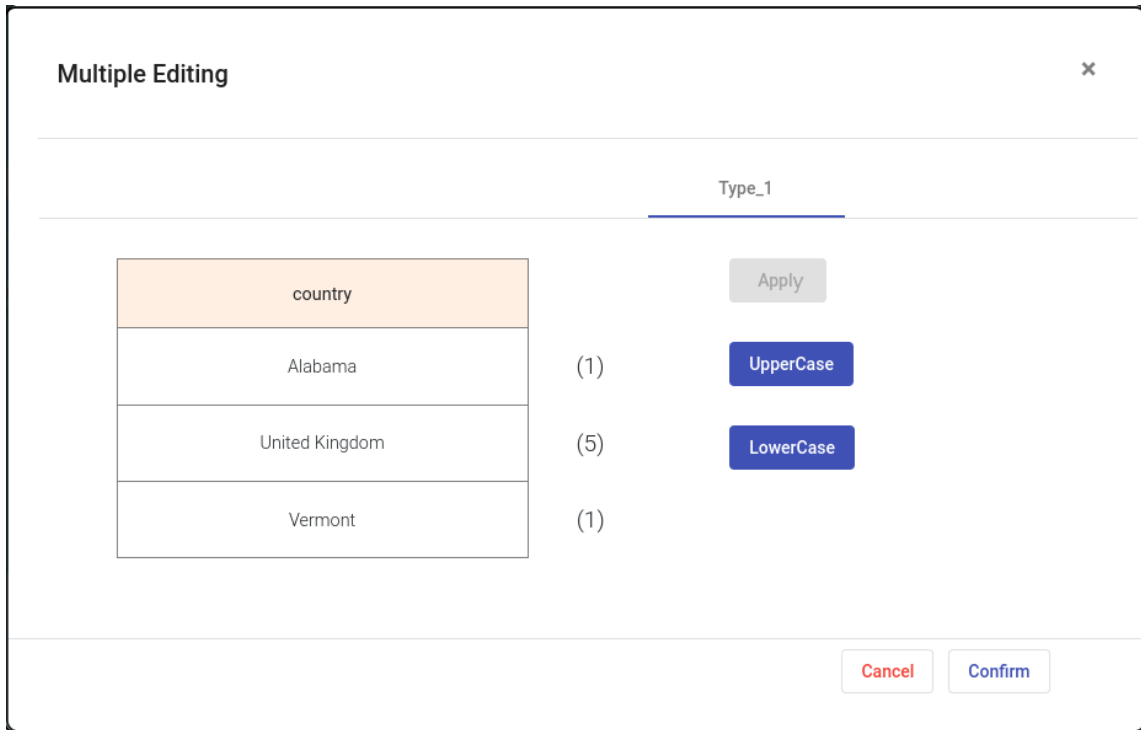


Figure 3.8: Editing of multiple rows interface

the column data type since the separators of the date are missing. In this case, the user must edit the value contained in the cell by adding separators between the day, the month and the year; moreover, it might want to add the complete year instead of the short form. This edit operation can be done directly in the cell. Once it is complete, the system executes again the ML technique to determine the new data type.

For performing bulk modifications on the values, the interface in Figure 3.8 was developed. For each column, the interface reports the unique occurrences of the column, with the number of occurrences. Similar strings are clustered together relying on the edit distance and then reported closer in the list of strings visualized in the interface. In this way, it is easier to visually detect the errors and correct them with the aim of obtaining a homogeneous representation of the same kind of information. The user can edit the single value, and the proposed modification is applied to all the occurrences (note that when corrections lead to a value already present in one table row, the two rows are collapsed). Moreover, the user can edit the case of the text (i.e. from lower to upper). For example, by modifying the value “United Kingdom” in “England”, the five occurrences are modified accordingly.



Figure 3.9: Data types editing

The interface in Figure 3.9 has been developed for easily changing the type of an entire column or a subset of its values when data types are not correctly identified. The interface can be activated on a single cell, which becomes the current target of the modification, or it can be activated on the entire column. Through this interface, the inferred data type can be modified into a new type, or into a concept and a property of the domain ontology. This interface can be also used for modifying mixed types to be extracted from a string. The interface is organized into 5 areas. In (1) the hierarchy of concepts available in the considered domain is reported along with basic types (collected in the button `General`). The user can select one of the available concepts, and the corresponding properties are reported in (2) (when the `General` button is pressed, the basic types are reported). In (3), when the interface is activated on a target cell, a single type is reported (the value type), otherwise, the components of the union types specified for the column are reported. In this way, it is possible to change the type for each component of the union type. In (4) it is reported the target value or the column name and it is highlighted with the current type for the column. The user can remove the current labelling (by clicking on the `x` button reported on the top right corner of the string) and apply a new concept and property. In (5) values of the same type present in the column are reported and the user can select those to which the type modification should be applied (all the values are the default behaviour). The user

can also decide to select the “text” checkbox reported in (6) to unify undesired union types, such as decimal and integer, and to treat the whole column as an instance of a single type. Then, the user can select the new type to be assigned to all values.

Example 16 *Two errors (false negatives) occur in the ZIP code column in Figure 3.6. Through the interface in Figure 3.9, the user can navigate to Type_2, choose the concept Address and the property ZIP and substitute the error with it. Moreover, Type_1 can be modified to the same concept and property leading to a single type for the column.* □

The possibility to modify types according to the concepts contained in the domain ontology can also introduce some issues that need to be properly managed.

Example 17 *Consider the column Name/Company in Figure 3.6 that the ML algorithm has typed `union(mixed_1, text)`, where `rec(name, surname)` is the structure associated with `mixed_1`. The value Legal Rep. Danielle Gray is of type `text` and can be changed with the mixed type `mixed_2` whose structure is `rec(Person.name, Person.surname)`. So, a more complex type than the one expected is generated.* □

To face this issue, a re-writing system based on rules [51] has been developed for the simplification of the type expression after the modifications applied by the user. The re-writing rules express correspondence between simple types and concept properties of the domain ontology occurring in the same table column. Once applied the re-writing rules, the union-type components presenting the same structures are compacted. The union type is finally transformed into a simple type when a single component is identified. In the previous example, the application of the re-writing system leads to the type `mixed` with a record structure `rec(Person.name, Person.surname)` and two patterns for the extraction of values.

3.4.3 Identification of a mixed type

The identification of sub-components of a mixed type is quite hard to be handled automatically, especially when errors and variability in the pattern might occur.

Example 18 *Consider the column address in Figure 3.6. The ML algorithm was able to identify the type `mixed_1` for some of its values, whereas the others are marked of type `text`. From the textual values, two different mixed types can be manually detected through the interfaces described in this section.* □

Figure 3.10 shows the main interface for extracting a value of a mixed type from a string. The interface is organized into four areas, (1) and (2) are equivalent to the one presented in Figure 3.9 and report the concepts with their properties. In (3), by contrast, the string on which the interface has been invoked is shown. Once the user has selected the property of a concept (in this case the `municipality` of an `Address`), he can highlight the part of the



Figure 3.10: Definition of a mixed type

string of such a type. This behaviour applied to all the components will lead to the situation reported in the area (3). In this way, we identify the terminal and non-terminal symbols that form our pattern. The non-labelled items are considered terminal symbols, while the labelled items are exploited for the generation of the pattern. Note that the void symbol can be applied for skipping variable parts of the string. Once the labelling is complete, the user can check if the generated pattern can be applied to other strings occurring in the same column (4) that adhere to the same pattern. When the user tries to apply the labelling to other strings, the interface in Figure 3.11 is shown. The top part of the figure reports the labelled string, whereas the left panel reports strings that do not present the same pattern and the right panel contains the strings that have been re-written according to the identified pattern. The user can check the correctness of the applied pattern in the right panel and move to the left panel those that have been erroneously annotated. Moreover, he/she can take note of the strings in the left panel because they require the specification of a different pattern or the identification of a different type.

Algorithm 1 is applied for checking if a string is valid for the identified pattern. In the al-

Algorithm 1 Check a pattern on a string

```
Input  $w \leftarrow$  string to be checked  
     $\mathcal{P} \leftarrow [c_0, \{t_1\}, \langle c_1, Occ(c_1) \rangle, \dots, \{t_n\}, \langle c_n, Occ(c_n) \rangle]$   
     $i \leftarrow 0$   
    result  $\leftarrow []$   
1: if  $c_0 \neq \text{empty}$  then  
2:      $i \leftarrow |c_0|$   
3:     if  $w_{0,i} \neq c_0$  then  
4:         return []  
5:     for  $s \leftarrow 1$  to  $n - 1$  do  
6:          $f \leftarrow \text{INDEXAT}(w, c_s, i, Occ(c_s))$   
7:         if  $f \neq 0 \wedge w_{f,f+|c_s|} = c_s$  then  
8:             if  $\text{TYPEOF}(w_{i,f}) = t_s$  then  
9:                 result.append( $\langle w_{i,f}, t_s \rangle$ )  
10:             else return []  
11:         else return []  
12:          $i \leftarrow f + |c_s|$   
13:     if  $c_n \neq \text{empty}$  then  
14:          $f \leftarrow \text{INDEXAT}(w, c_n, i, Occ(c_s))$   
15:     else  
16:          $f \leftarrow |w|$   
17:     if  $\text{TYPEOF}(w_{i,f}) = t_n \wedge w_{f,|w|} = c_n$  then  
18:         result.append( $\langle w_{i,f}, t_n \rangle$ )  
19:     else  
20:         return []  
21:     return result  
  
22: function INDEXAT( $w$ , lookFor, occ, start)  
23:      $p \leftarrow 0$   
24:     localStart  $\leftarrow$  start  
25:     for  $i \leftarrow 0$  to  $occ + 1$  do  
26:          $p \leftarrow w.\text{indexOf}(\text{lookfor}, \text{localStart})$   
27:         localStart  $\leftarrow p + 1$   
28:     return position
```

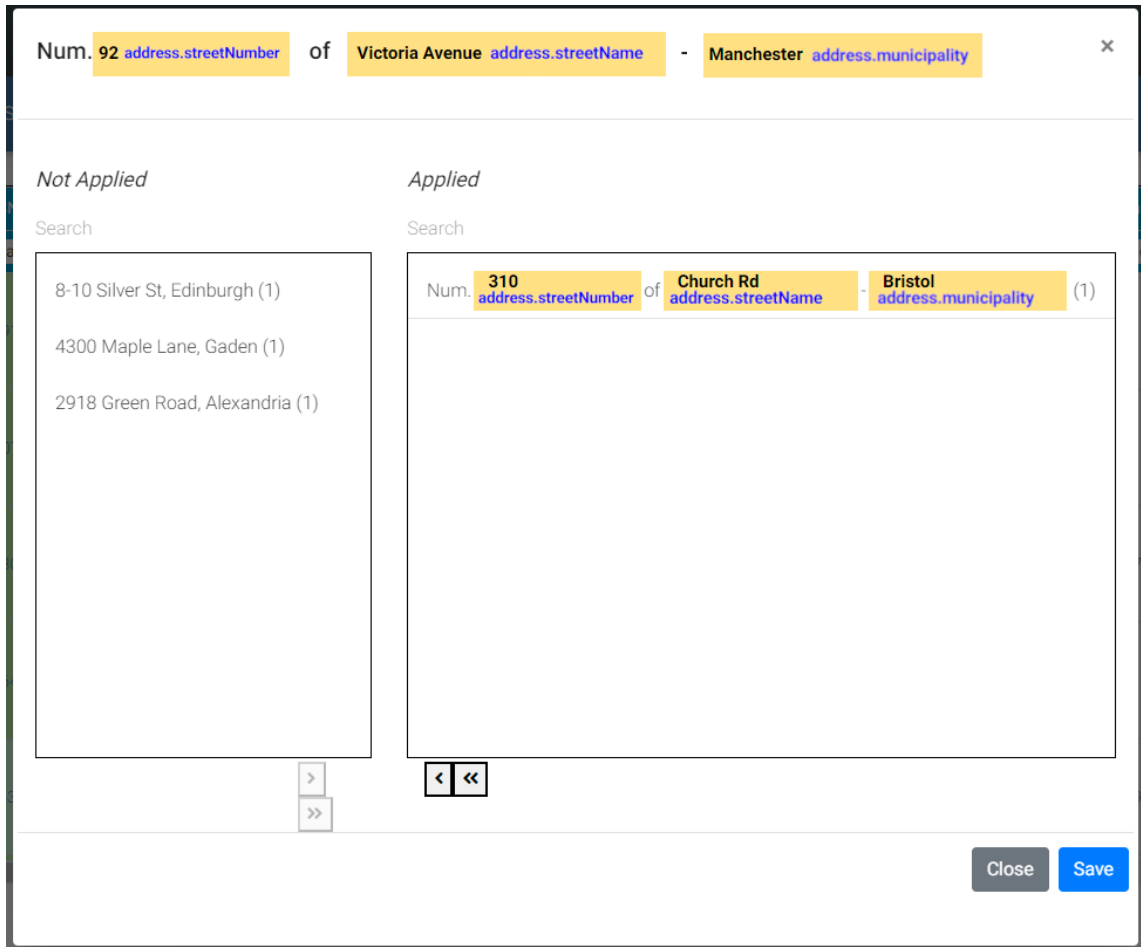


Figure 3.11: Application of a pattern to other strings in the same column

gorithm, the notation $w_{i,j}$ is used for identifying the sub-string of w contained within the position i and j , and $|w|$ is the length of the string w . Moreover, the algorithm exploits the *indexOf* function that simply identifies the initial position of occ^{th} occurrence of the string `lookFor` within the string w starting from the position *start*. The position 0 is returned when `lookFor` is not identified.

The algorithm first checks the presence of the possible first terminal (c_0). If it occurs, the index i (used to identify the initial position of the next non-terminal) is set to the length of c_0 , otherwise, i is set to 0 (the initial value). Then, for the rest of the pairs $[\{t_s\}, \langle c_s, Occ(c_s) \rangle]$ where $1 \leq s \leq n - 1$, the algorithm identifies the position f of the occurrence $Occ(c_s)$ of the terminal c_s . This means that the string $w_{i,f}$ is the non-terminal that should be of type



Figure 3.12: Pattern definition for column Address

t_s . When this is true, the pair $\langle w_{i,f}, t_s \rangle$ is included in the result. Otherwise, the empty list is returned. At the end of the loop, the last type t_n should be checked (by taking into account the eventual presence of terminal c_n). Its management is similar to the general case, but we have to consider the possibility that it is not present. The algorithm returns the list of sub-strings of w with their associated type. The algorithm is not tolerant to errors or slight modifications of the terminals. However, in our scenario, terminals are quite regular and when a string follows a different pattern, the process here described can be applied again.

Example 19 *The remaining text and the mixed type automatically identified by the ML approach in the column Address are labelled as reported in Figure 3.12. The terminal symbols of the second row are hidden from the visualization since the labelling was already made, while the terminal symbol of the second row (the comma) is shown since the screenshot was taken during the labelling process.* □

It should be noted that in this example we intentionally added several strings that are organized in several ways for illustrative purposes. However, usually, we find just one or two patterns in the same column, and therefore the usage of the mixed types identification process is limited. At the end of the definition of the new mixed type, the terminal symbols are hidden from the data visualization.

3.4.4 Correlation between rows

As outlined in our problem formulation, it might be possible that correlations can be identified among different instances of the concepts identified in our domain and that such correlations can occur in one or more rows of our extracted table. The correlation can be induced by the presence of strings, sub-strings or the position of rows in the extracted table. Several kinds of correlations expressed in different ways can be identified in a single document.

Example 20 *Figure 3.13 shows some examples of correlations. A correlation can be identified between the first row (case (1)) and the second one; in this case, the debtor Mary*

Electric Bills - Albuquerque, New Mexico					
Full/Company name	SSN/VAT	date of birth	address	penalty	tax
① Mary White	616-03-4712	22-apr-96	641 Oxford Court	4900	980
Co-debtor William Cooper	616-03-4712				
② Legal Representative James Brown	502-31-6521	26-Sep-85			
Collier Inc	14-5537560		4300 Maple Lane	12930	2586
③ Barbara Howard HEIR Kelley Petrie (419-36-3244)		07-apr-88	Ritter Street	560	112
Angela Ghent	364-70-9455	11-Aug-90	3188 Bird St.	644	128,8

Figure 3.13: A subset of possible correlations among rows

White has a co-debtor that should be considered. The presence of a correlation can be identified by the existence of the word “co-debtor” in the cell of the second row. Moreover, the remaining cells of the second row are empty, and finally, the SSN of the first and second rows are the same. In case (2) the second row is correlated with the first one; the company “Collier Inc” has a legal representative whose information is inserted in the above row. The correlation can be identified by the presence of “Legal Representative” in the first cell of the first row. Finally, in case (3) the correlation occurs in a single row. In this case, the first full name represents the heir of the person that follows. The keyword contained within the cell and the absence of an SSN are essential for the identification of the correlation. □

As shown in the example, correlations can be expressed in different forms and it is not possible to determine a unique way to parse them. It should be noted that the existence of a correlation alters the homogeneity of the column's data types, therefore it is necessary to identify it and adjust the data so that they are compliant with the rest of the structure of the spreadsheet. We made a distinction between correlations between two consecutive rows and correlations that occur within the same row. For identifying a correlation between consecutive rows, we defined a declarative pattern-based language that allows the user to define rules to express their knowledge about the existence of a relationship. These rules are specified through a GUI, they can be grouped in sets and applied before the processing of each spreadsheet. When two consecutive rows match a rule, then a correlation is identified. More precisely, each rule is identified by a unique name *name*, and is applied to consecutive rows, the current row, r_i , and the next row, r_{i+1} . A rule is composed of the conjunction of basic *conditions*, that check for the existence of a relationship, and an *action*, that expresses the way the information from the two rows should be joined when the condition is verified. The following two basic *condition* can be specified:

- $r[k] \text{ op } v$ (named *basic condition*) requiring that the k -th cell in the row $r \in \{\text{current}, \text{next}\}$ is compared - according to the operator *op* with a value v , where

Figure 3.14: Correlation rule specification

$op \in \{=, \neq, \text{startwith}, \text{endwith}\};$

- $\text{current}[k] = \text{next}[k']$ (named *equijoin* condition) imposing that the k -th cell of current row is equal to the k' -th cell of next row.

action is specified by a tuple $(\text{conc}, \text{rel})$ determining the way in which r_i and r_{i+1} should be concatenated ($\text{conc} \in \{\text{natural}, \text{inverse}\}$) and the kind of relationship that exists among the two rows. In our application domain, we identified a set of relationships, like *extra*, representing further information about the invoice; *LR*, when the row contains the legal representative of the invoice; *heir* when the invoice is titled to a subject that is dead and one of his/her heirs should be contacted, *co-debtor* when a debt is shared among more than one person and so on.

Example 21 In the case of Figure 3.13 (1), the information about the co-debtor is reported in the second row. For its identification, the rule contains two conditions: one on the presence of the string *Co-debtor* and one on the equijoin between the SSNs'. If a pair of rows satisfy the condition, the second row is concatenated after the first one (normal order). \square

It is possible to define a correlation among two rows on the fly, by clicking on the operations

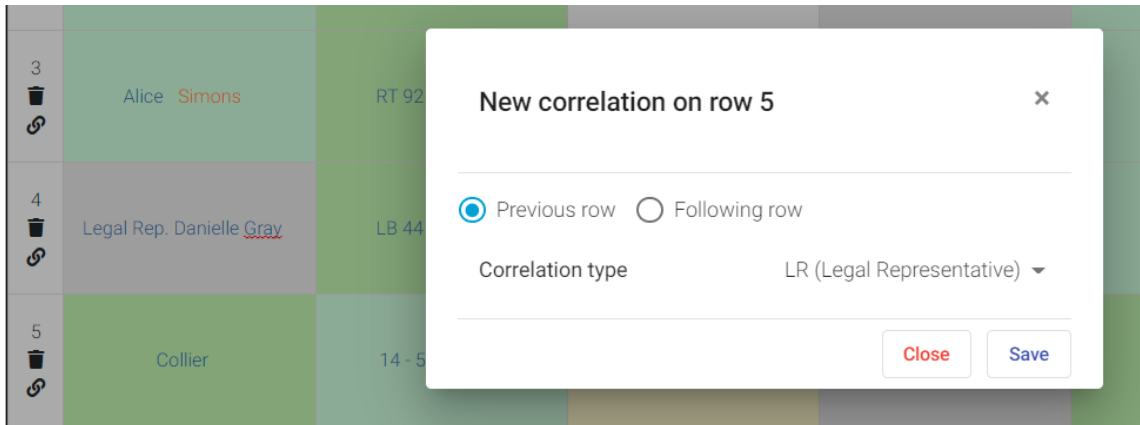


Figure 3.15: Definition of a correlation on the fly

on tuples loaded in the interface (the link icon). In the example of Figure 3.6, we did not apply a group of correlation rules before processing the file. Therefore, we specify the correlation using the interface of Figure 3.15. When the user wishes to determine a correlation, he/she can click on the link icon in correspondence with the **current** row, and then he/she can specify the type of the correlation. As a result, the rows are concatenated together and the not empty columns of the concatenated row are added at the end of the main row. Moreover, a new column containing the name of the correlation is added between the two rows. At this point, in the example of Figure 3.6, the prefix “Legal Repr.” of the correlated row is no longer useful, therefore the user can apply semantic labelling only on the name and the surname of the legal representative using the interface of Figure 3.10; this operation would remove the prefix, implicitly labelled as void, from the visualization.

The interface of Figure 3.14 cannot be used for the definition of a correlation on the same row. Therefore, we exploit the interface of Figure 3.10 through which the user can manage the correlations occurring on a single row. Indeed, we treat the text containing the information about the correlation as a mixed type. To do that, we extended the domain ontology and introduced the concept of “correlation”, which contains the types of relationships. Then, we use it as depicted in Figure 3.16 with respect to example (3) of Figure 3.13. At this stage, no additional columns is added, however, the correlated person will be treated in Phase 2.

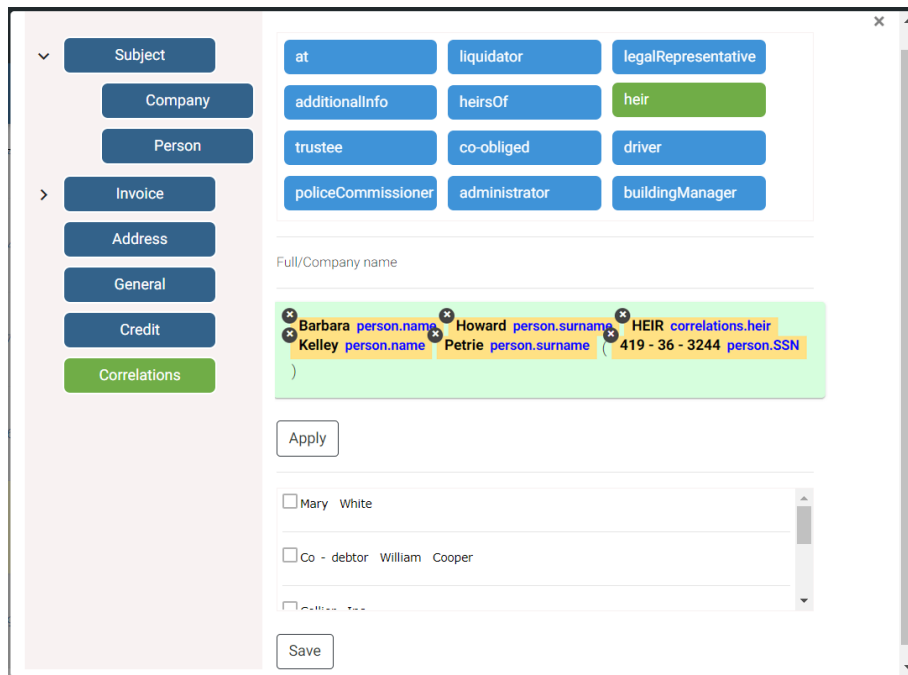


Figure 3.16: Definition of a correlation on the same row

3.5 Experimental results

Besides the experiments with 1 million synthetic data that we have already described and whose purpose was to evaluate the type prediction capabilities of the ML algorithm, we have considered also other experiments for assessing the quality of the proposed solution. On one hand we have evaluated the behaviour of the approach on real documents made available from the debt agency and, on the other hand, we have evaluated the usability of the proposed interfaces. In the remainder of the section, we discuss the obtained results.

3.5.1 Validation of real documents

The trained model on synthetic data has been used for inferring the type of 50 different real CSV documents provided by the debt agency with at least 100 rows and between 8 to 70 columns. The minimum threshold for inferring union types has been set at 20%. A table has been extracted from each of them and the type of values and columns has been inferred as described in Section 3.3.1. Two experiments have been done on the obtained tables.

In the first experiment, the tables have been manually checked for determining:

- **correct columns:** If the predicted type matches the content of the column;

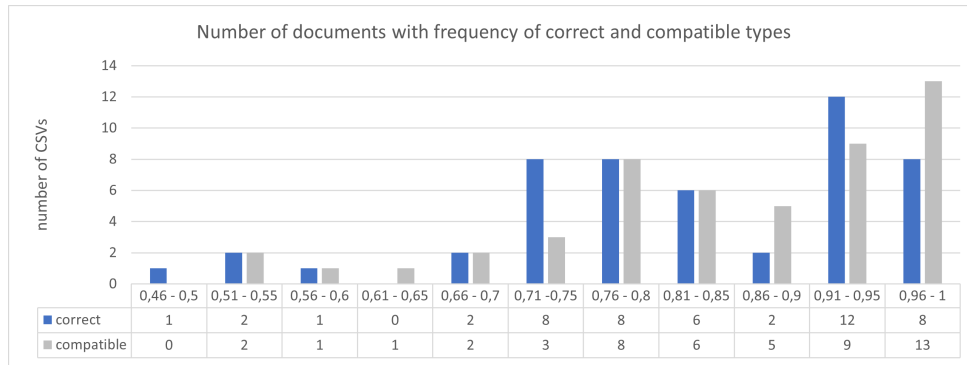


Figure 3.17: Number of documents for which the frequencies of the correct columns (series A) and the correct and compatible columns (series B) with respect to the total number of columns fall in the range $[0.5, 1]$ with a step of 0.05.

- **compatible** columns: If the predicted type is correct but is not the most appropriate one (e.g. a street number is predicted of type integer);
- **incorrect** columns: otherwise, e.g. if the predicted type does not match the column content (e.g. the type is SSN but it also contains VAT).

The frequency of the “correct columns” and “correct and compatible” columns with respect to the total number of columns has been computed. We then defined ten ranges of frequencies that are used for classifying the documents. Fig. 3.17 reports the number of documents whose frequencies (“correct columns” and “correct and compatible”) fall in the range $[0.5, 1]$ with a step of 0.05. The figure points out that for 33 documents we were able to infer correct and compatible types for more than 80% of the columns (by considering only correct types, the number of documents is 28). Moreover, for only 4 documents we were able to infer correct and compatible types for less than 65% of the columns. By looking more carefully at these last documents, the low performances are due to the high number of mixed types that are not included in our type system. Indeed, in these cases, the algorithm fails to correctly identify the subcomponents. Concerning the compatibility errors, usually, they occur for the `streetNumber` type (the values are identified as integers). All the mistakes identified in these documents have been fixed by means of the interfaces described in Section 3.4. We remark that the fixing process is quite easy and fast. Indeed, in the case of the CSV file with the highest number of errors, we fixed it in less than 5 minutes.

The second experiment uses the same set of CSVs. For each CSV we considered the result obtained with the ML approach and the result obtained with manual labelling of each column and cell. In these experiments, we evaluate the accuracy of the data type inference

		Actual values	
		DT_1	error
Predicted values	DT_1	TP	FP
	error	FN	TN
	DT_2	FN	FP

Table 3.1: Confusion matrix: the prediction for a data type DT_1 is compared with the presence of errors or another data type DT_2 . TP: true positive, TN: true negative, FP: false positive, FN: false negative.

approach. The manual editing does not affect the values of the CSV, it works only on the data types identified for each column and cell.

Table 3.1 reports the confusion matrix for evaluating the *actual* values (i.e. the types identified in the manually edited table) with respect to the predicted type (i.e. the same type D_1 , an error or another data type DT_2). In the matrix, a true positive is obtained when the predicted type coincides with the manually labelled one (a false negative, otherwise); whenever the cell has been manually labelled with an error, a false positive is obtained independently from the predicted type. This confusion matrix is different from the standard ones because it has an extra row for handling the presence of an error and thus discriminate the presence of a true negative. Note also that what we evaluate is the ability of the system to automatically predict the cell type, so user intervention for identifying the presence of an error and correcting it is not taken into account.

Since we consider the data type of each cell and not the type of the column, the evaluation of union types is done by considering the same confusion matrix as simple types. In the case of mixed types, a data type is true positive if all the components of the predicted type are equal to the corresponding manually labelled components. If a single component is different, it is classified as a false negative. Figure 3.18 shows the obtained accuracy for each type of column. Specifically, we have considered *basic single types* (simple or mixed types), *union of basic types*, and *union of mixed types* (when at least one of the components is a mixed type). The average accuracy is higher than 80% in the considered columns. The accuracy is better for columns presenting basic types or union of basic types, whereas it decreases in presence of columns with mixed types (or union of mixed types).

We evaluated also the compatibility between the ML predictions and the manually labelled data. We defined a compatibility mapping between the data types that the ML approach is able to identify and the types that the user can use to label the cells. For each mismatched cell, we evaluated the compatibility. As a result, we obtained slightly better performance (an

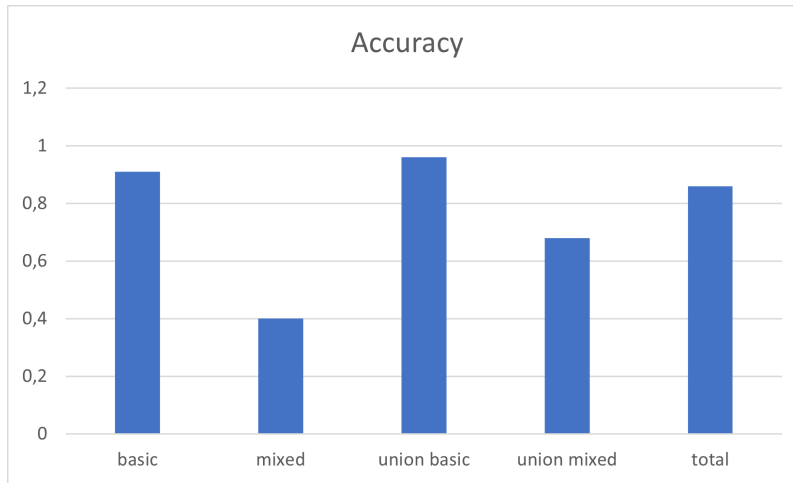


Figure 3.18: Accuracy of each data type category

increase of 10% in accuracy), which could be improved using different data domains.

The accuracy has been selected for evaluating the performance of our approach because of the scarcity of false positives in our data set. Indeed, the precision, in this case, would result in a too-high evaluation. The lack of false positives is due to the approach we have followed in the construction of the ground truth. Indeed, the only way to have a false positive is when an error occurs in the data and the ML approach does not identify it.

Finally, we checked the existence of a correlation between the number of columns occurring in the table and the accuracy of our prediction mechanism by exploiting the Pearson correlation coefficient. A moderate statistical evidence of such a correlation has been identified.

3.5.2 Evaluation of usability - phase 1

We organized a usability test of the Web application. The aim of this test is to evaluate if the users can smoothly interact with the application and use the provided tools, what level of knowledge in computer science is needed, and check the existence of critical aspects that should be fixed or improvements to be applied. This test is composed of three parts: first, the user watches a video that introduces him/her to the problem and shows the system usage. Then, two tasks are assigned to be carried out on specific files. Finally, the user should fill out a questionnaire about the experience he/she had with the system. The questionnaire is composed of three parts: *i*) personal information (age, gender, level of instruction) and technical abilities (computer skills in general, knowledge of operating systems, skills in the use of spreadsheets, ...); *ii*) users' opinions about the assigned tasks and their complexity

#	goal	time	success	failure
1	mixed type	6 min	Definition of a mixed type and application of the labelling to other strings through the “apply” function	Lack of the pattern definition or application of a new procedure every time
2	errors	6 min	Detection and correction of the errors on values/types through the error panel	The errors are not corrected and the error panel is not exploited
3	bulk editing	4 min	Rows are updated in a single operation	Rows are updated one at a time
4	correlations	4 min	Identification of the rows correlation, verification of their cell data types	The correlations are not defined or the correlated row types are wrong

Table 3.2: Tasks identified for the usability test

iii) users’ opinions about the functionalities of the proposed tool. These questions have been rated using a Likert scale (from “strongly disagree” to “strongly agree”).

We selected 20 participants, 12 males and 8 females, 60% of them were between 21 the ages of and 23 years old, 20% between 24 and 26 years old and the remaining ones were more than 26. Most of the users were recruited among personnel and students of the department of computer science of the University of Milan and therefore they have good technical skills. However, they are not involved in this project and they have little knowledge of the application domain. Only a small part of the participants (50%) feels confident in using Excel. Most of the students are currently attending to their bachelor’s degree, therefore they have only a high school diploma. Users have an average knowledge of different operative systems and they use a computer or a laptop mostly for working or studying.

Table 3.2 reports the tasks that we have identified for checking the main functionalities of our system. Each task requires the processing of a spreadsheet that is specifically created for the purpose of the task and whose content can be easily understood also by non-expert of the domain. Even if the spreadsheets correspond to real documents of our domain, their content has been anonymized for preserving user privacy. For each task, Table 3.2 reports the main goal, the time required for completing the task and when the task can be considered successfully completed or when it is completely a failure. In order to reduce the user efforts in completing the tasks, we decided to assign to each participant *task 1* and *task 4*, or *task 2* and *task 3*. In this way, the maximum time required is 10 minutes (without considering the time required for watching the instructional video).

All the individuals were able to complete at least a part of the assigned tasks within 10 minutes. Some of them (30%) were not able to finish the assigned task because of time limits. However, the average time required has been of 8 minutes. A good fraction (70% of the users) thought that the assigned tasks were easy and enough intuitive.

For *task 1*, most of the individuals (85%) were able to specify a mixed type through the interface. All of them used the “apply” button to label all the mixed types in a single column. The main reason for the failure of this task was the choice of the wrong interface (they selected the interface for the modification of column type instead of the one for modifying the cell type). For *task 2*, 75% of the individuals used the error panel and the general impression about its usefulness is very positive (from partially to strongly agree). The users that did not exploit the error panel, tried to increase the number of rows per page in order to identify the errors. In these cases, the identification and correction of the errors required an additional time of up to 3 minutes. Concerning this task, only 28% of users had trouble in distinguishing errors occurring on the data type (i.e. the component of a union type was not identified by the ML algorithm because it was under the considered threshold) from errors occurring on the data (i.e. a date is written without separators). For *task 3*, most of the individuals (86%) were able to use the bulk editing functionality and all of them thought it sped up the editing process. The remaining part did not notice the error occurring within the data (usually an additional letter in the name of a city) and they corrected it by editing the data type. Finally, for *task 4* all the individuals were able to detect and define the correlations but only 75% of them modified the name of the correlation, while the others used the default one (the first of the list). The reason is due to the excessive use of the horizontal scroll bar for the visualization of the columns associated with the correlated tuples which reduce the visibility of the table components.

Figure 3.19 shows the box plots of the average execution time for each task. Each task required an average time between 3.5 and 5 minutes. The most time-consuming tasks are *task 2* and *task 4* which also include most of the cases in which users were not able to complete the assignment. *Task 1* was the one that required less time than the others since most of the participants used the “apply” button to label the data.

The 95% of the users agreed that the application is easy-to-use and intuitive and the 85% declared that they did not have problems during the error correction process. The greater difficulties were related to the understanding of the specific domain; most of the users did not know the meaning of the concepts of the domain ontology and tried to identify the most suitable one. Moreover, the application provides a lot of functionalities and the user needs time to gain confidence in the system.

Even if the users did not provide suggestions concerning improvements of the GUI, all of them believe that to gain full confidence with the application, a one-day experience should be considered. More in general, the users agreed that the Web application could be easily used by other users having the same level of knowledge in the field of computer science.

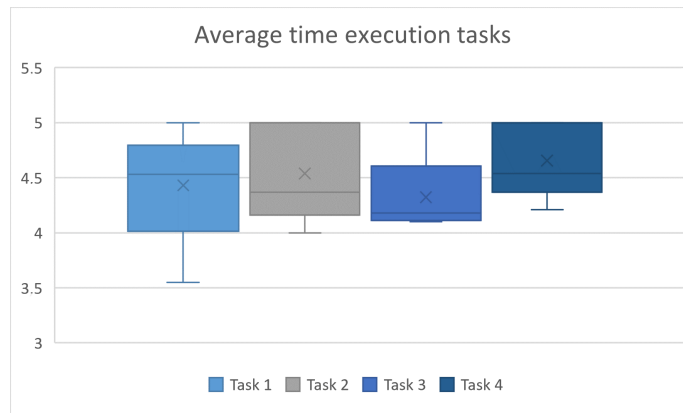


Figure 3.19: Average execution time of each task during the usability tests.

3.6 Concluding remarks

Type inference is a well-known problem in computer science and many approaches have been defined ([92], [165], [112], [38], [154]). The peculiarity of our context is the presence of mistakes in the set of values for which we wish to identify a type and the presence of types that share values (e.g. the year type is included in the integer type) making the type assignment problem more complicated.

To face these issues, in this chapter we have proposed a semi-automatic approach for determining the types of values and columns contained in a table extracted from a spreadsheet that relies on the adoption of a decision tree on top of several basic type recognizers. This approach is used for deciding the most likely type to be assigned to the values contained in a table column. The approach takes into account also the possibility of assigning more than one type to the same column (in the spirit of union types).

Our approach is useful in the interpretation step of the table understanding problem when the values of a single column can assume different types and can present wrong values. Due to the variability of the data that need to be handled and the lack of a significant corpus on which an ML technique can be trained (especially for the Italian language), we have proposed an approach that combines a decision tree trained on synthetic data for inferring the type of values and columns and the use of GUIs for correcting mistakes.

Once the automatic approach has identified the possible set of types, the user is supported with GUIs in fixing the issues identified by the machine learning approach. In particular, the purposes of our interfaces are, on one hand, to allow modifications on a single sample and to propagate them to all the column values presenting the same characteristics, and on the

other hand, to fix types and values. The user can also specify semantic types for single cells on entire columns; indeed, he has specific knowledge of the domain and, therefore, his annotations should be trusted. For example, a document value can be labelled as the balance of an invoice. This information, which is gathered during the manual fixing of the values contained in the table, is of paramount importance in the subsequent phase of semi-automatic semantic characterization of the table content.

The approach has been tested on a collection of documents made available by a debt collection agency that needs to handle every day these kinds of document that is highly heterogeneous and contains many mistakes. Our experiments proved the feasibility of the approach and the utility of the developed interfaces for easily fixing mistakes and extracting types from many columns presenting heterogeneous information.

Moreover, a usability test has been run on the developed application. As a result, we concluded that the 95% of the users believe the application is easy-to-use and intuitive. However, during the tests, some problems concerning the error correction task of a data type have arisen. In particular, 15% of the users had trouble in distinguishing the panel associated with a column and the one associated with a cell, and 25% of the users did not exploit the error panel. As a solution, a button could be included in the modal for editing a column containing a mixed type to be used to open the correct interface for labelling the sub-components. Concerning the error panel, "help" buttons can be included for explaining to the user the steps that he/she has to do for completing the activity.

The work discussed in this chapter can be extended in several directions. By means of the graphical interfaces described in Section 3.4 it is possible to define new patterns and include them in the ML process described in Section 3.3.1. This is an interesting research direction in the spirit of incremental learning and thus being able to adapt the model without the entire re-training. Then, we wish to include other recognizers in our system that are approximate. They would be particularly useful for dealing with many errors. At the current stage, we have considered approximate recognizers for ZIP and VAT values that try to slightly modify these values (by padding some zeros at the beginning of the string) before checking the validity. They have been particularly useful and their systematic use can improve the performance of the type inference approach. In addition, the recently proposed boosted trees [43] that use a combination of decision trees and boosting algorithms to improve the accuracy of predictions and reduce the risk of overfitting might be considered. XGBoost (eXtreme Gradient Boosting)² is a popular implementation of this approach. One of the key features of XGBoost is its ability to handle large datasets efficiently and the ability to handle missing values in the dataset. Furthermore, we observe that we applied a basic

²<https://xgboost.readthedocs.io/en/stable/tutorials/model.html>

multi-label approach to predict column types, and we plan to adopt more refined multi-label techniques to significantly improve predictions of union types [177]. Finally, the approach described in this chapter relies on the use of synthetic data that have been generated for the specific domain for training the ML approach. Whenever a different domain needs to be considered, specifically tailored synthetic data should be considered. As a future work, we wish to consider the possibility of using data generative approaches [131] for the creation of positive and negative examples that can be used for training the algorithms.

Chapter 4

Semantic descriptions of the table content

By exploiting the facilities described in Chapter 3, the table extracted from a spreadsheet has been cleaned and removed from syntactic errors. Moreover, by means of the interaction with the user, the basic types, identified by the machine learning approach, have been translated into properties of the ontology concepts, thus providing a better characterization of the table content. However, in our approach, we do not force the user to provide a semantic characterization for all the table columns, some columns can be left "unmatched" and we will try to characterize them with the methodology introduced in this chapter.

The purpose of this chapter is to determine a semantic characterization of the table content in terms of the domain ontology. This is obtained by generating a *semantic description* (SD) of the table content (as described in Section 2.3 of Chapter 2) starting from the content of a table T that exploits: *i*) the concepts initially identified in T through the approach described in the previous chapter; *ii*) a domain ontology O for the construction of a complete SD containing all possible direct and indirect relations involving the concepts initially identified in the table T ; *iii*) different weighting systems for identifying the best relations among those included in the complete SD . The weighting system takes into account the specificity of the relations occurring in SD with respect to the domain ontology O (inherited relations are less relevant than specific ones) and the weights generated using a GNN. The model is trained on the consolidated knowledge graph KG_C that adheres to the constraints imposed by the ontology O and contains instances already known valid in the considered domain.

The chapter is organized as follows. Section 4.1 presents our methodology for the creation of SD . Section 4.2 presents the approach used for the creation of a GNN model that

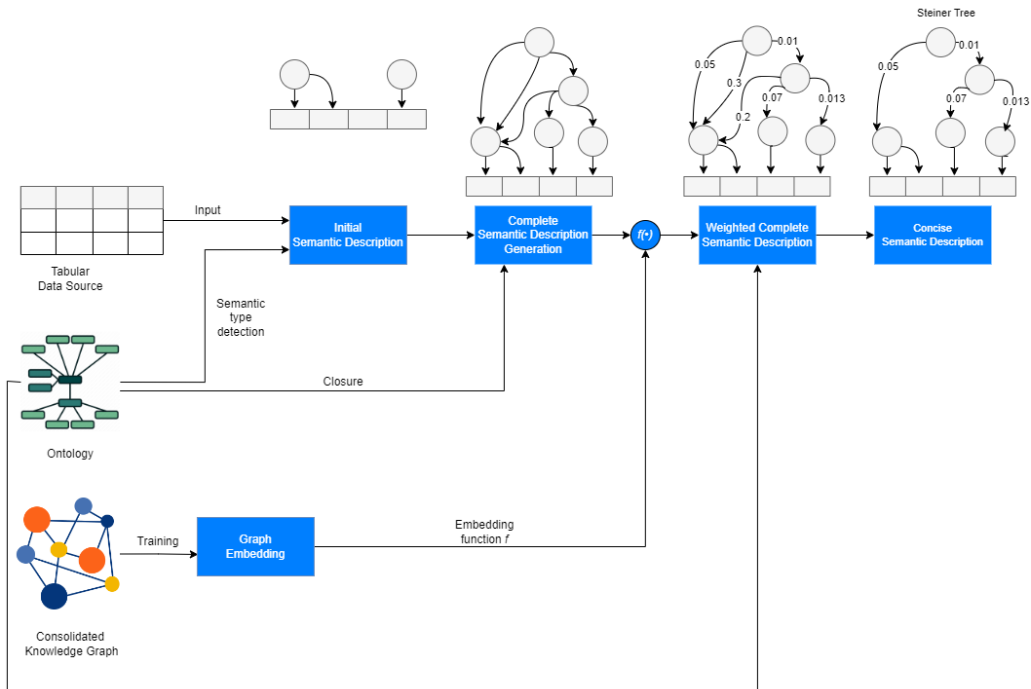


Figure 4.1: Pipeline of our methodology for the generation of SD .

learns the KG_C and the kinds of predictions that can be derived, Section 4.3 discusses the creation of the complete SD starting from the initial annotation. Section 4.4 presents the weighting system that can be exploited for tagging the complete SD . Section 4.5 deals with the unmatched columns of Table T and proposes an extension of the complete SD for identifying properties associated with them. Section 4.6 discusses the creation of the concise SD . Section 4.7 discusses our experiments and compares our work with the previous one and, finally, Section 4.8 draws our concluding remarks.

4.1 Overview of the Methodology

A pipeline of our proposed methodology is shown in Figure 4.1. The SD construction can be divided into four steps. In the first step, we take into account the annotations obtained for the table columns to build an initial semantic description. The complete semantic description (second step) is obtained by considering direct and indirect relationships among the identified entities in the domain ontology. The third step consists in weighting the identified relationships according to the ontology and the link strengths obtained from the embedding function. In the last step, we generate a concise SD by applying the Steiner tree algorithm

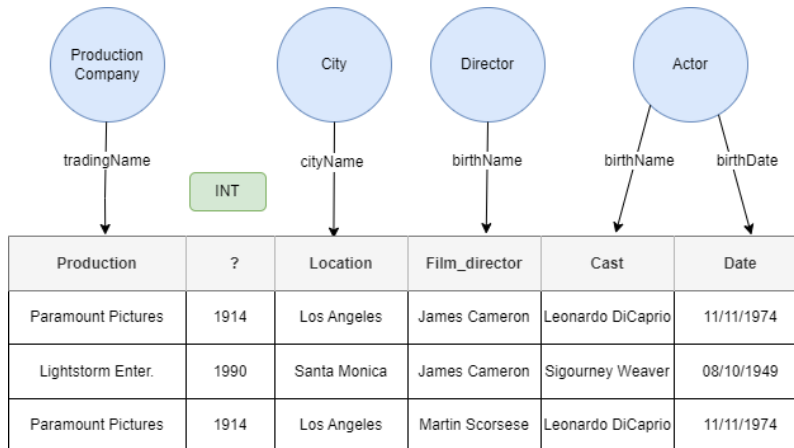


Figure 4.2: Starting point for the construction of the semantic description of a table.

on the complete SD and thus obtaining a minimal cover tree of the columns that have been associated with concepts of the domain ontology.

For the sake of simplicity, in this chapter, we present simpler tables with respect to those considered in the previous chapter. Specifically, columns with a mixed type are substituted with their components. This transformation allows dealing with columns with simpler values. Moreover, we do not deal with union types. Actually, when columns with union types are present, the methodology presented in this chapter can be applied to groups of tuples presenting the same type leading to the creation of SD_1, \dots, SD_n semantic descriptions (where n is the number of combinations of union types occurring in the table). Then, these descriptions can be merged together in a single SD containing the union types. The presence of union types is made evident from the presence of more edges incident in terminal nodes of SD requiring to extract values of different types. In the next chapter, we will present a graphical representation of SD that supports the presence of union and mixed types. This representation will be used by the user for checking the result of the automatic approach and deciding the modification to be applied.

In the presentation of the methodology, we will refer to the excerpt of the DBpedia ontology proposed in Figure 2.3 of Chapter 2, because it has a complex structure which includes hierarchies and multiple labelled edges among the entities, and the initial annotation of the table in Figure 4.2. The figure shows the result that can be obtained from Phase 1 and is the starting point of the methodology presented in this chapter.

4.2 Graph embedding

Graph embedding is an ML approach for transforming a graph into a lower dimensional vector space whilst maximally preserving properties like graph topology and ancillary data [75]. Initial methods based on GNNs [101] considered homogeneous nodes and edges, whereas new approaches are specifically tailored for heterogeneous graphs [146]. In our case, the embedding of KG_C is computed using a full heterogeneous GNN model [164] in which the basic property values are treated as nodes of the graph.

This model can handle both heterogeneous nodes and edges through two hidden layers with a graph attention mechanism [161] for computing the embedding. These layers are implemented as hetero convolutional layers (HeteroConv). Each relation name $r \in R$ has its attention mechanism and the node embedding is obtained as the sum of the contributions of each convolution defined on the relations in which it is involved. The graph attention mechanism has been introduced since many real-world KGs can contain relationships from multiple sources of varying quality (e.g. interactions extracted from unstructured texts are less reliable than manually curated ones).

Concepts and properties are considered graph nodes. The constant value 1 is used as a node feature for each concept to capture its structural information, whereas properties are treated as nodes and specific features for each basic type are extracted. Table 4.1 resumes the property features we considered for each basic type. Besides the specific value, the minimal and maximal value that the property of that type can assume is computed for the `int`, `double`, `gYear`, and `date` types. For the values of the `string` type we computed their length, number of white spaces, open brackets, and punctuation marks. We also performed language detection to distinguish English texts from others. All these features can help the model understand the difference between the relations. For instance, most locations have parenthesis in their string values. These node feature vectors are also used as initial representations of tabular data (i.e. the table rows, after the annotation process).

The function $emb_{KG}(\cdot)$ associates each node and edge of KG_C with its embedding. The computed embedding depends on the GNN structure and on the weights associated with the connections between the neurons. The model takes into account the characteristics of KG_C (i.e. the list of all node and edge types, the network topology, the node feature and the adjacency matrix divided by node and edge types). These weights are updated via backpropagation until the loss function \mathcal{L} [25] is minimized.

Once the embedding is generated, it is used for two prediction tasks. First, for identifying the relation that might exist between two nodes of SD . For this purpose, a scoring function \mathcal{S}_{KG} (e.g. DistMult [172]) is adopted.

Property type	Features	Property type	Features
int/double	The integer/double value min/max value	int/gYear	Distance from 1970 min/max value
string	length num whitespaces, num brackets num punctuation isEng	date	distance from the epoch day of the week month of the year min/max value

Table 4.1: Property features that are extracted for each basic type.

For each edge $(u_1, r, u_2) \in E_R \cup E_T$ of a SD , \mathcal{S}_{KG} is computed, relying on the embedding of the nodes u_1, u_2 and the relation r , and the sigmoid function σ , as follows:

$$\mathcal{S}_{KG}((u_1, r, u_2)) = \sigma(\text{emb}_{KG}(u_1) \cdot \text{emb}_{KG}(r) \cdot \text{emb}_{KG}(u_2))$$

The second prediction task, denoted $Property_{KG}$, consists in identifying the property p that should be assigned between a node $u_c \in U_{C_s}$ and $u_t \in U_T$ among those belonging to the set $P \subseteq P(C)$ by taking into account the value of u_t . The scores for the triples u_c, p, u_t for each $p \in P$ can be computed through the function \mathcal{S}_{KG} and the triple corresponding to the highest score returned. Formally, $Property_{KG}(u_c, u_t, P) = \text{argmax}_{p \in P}(\mathcal{S}_{KG}((u_c, p, u_t)))$

Figure 4.3 shows the proposed heterogeneous GNN architecture. Two HeteroConv layers with a graph attention mechanism act as an encoder to generate node embeddings, and then DistMult followed by a sigmoid function generates the link scores. The first HeteroConv is followed by a ReLu function [5]. The second HeteroConv layer has two neurons because we want to solve a link prediction task (i.e. a binary classification task with two possible outcomes), while the first has a configurable hidden dimension. The ReLu function has been selected among other possible activation functions because it is non-linear and it provides a good approximation for the GNN result. We also chose the GAT architecture because, differently from other architectures, it uses an attention mechanism and it is the most consolidated approach. We suggest using a number of hidden neurons that is not much bigger than the number of features. In contrast to Feed Forward or Convolutional Neural Networks, it is typically discouraged to add more than two or three graph convolutional layers to a GNN because, due to the small world property that characterizes a large fraction of complex networks [166], it would end up aggregating node features with those of all the other nodes, causing a problem known as over-smoothing [32].

The computational complexity of our model is $O(\sum_{r \in R} |E(r)| \cdot (f_s(r) + f_t(r)))$, where $|E(r)|$ is the cardinality of the set of triples with relation r in KG_C , and $f_s(r)$ and $f_t(r)$ are the number of features for the node types that are the source and target of the relation r .

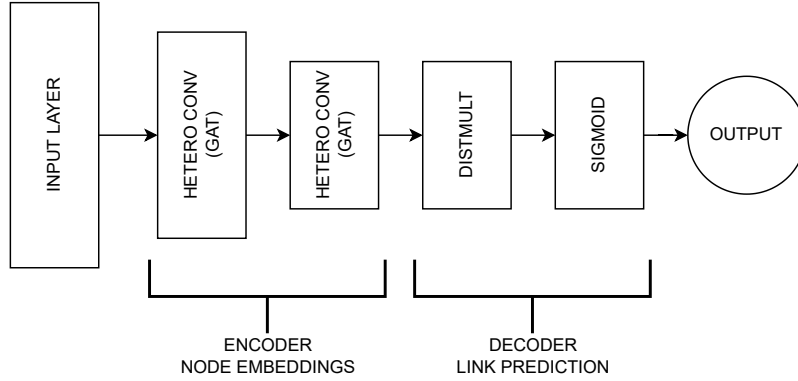


Figure 4.3: Heterogenous GNN architecture. Two HeteroConv layers with graph attention mechanism act as an encoder to generate node embeddings, and then DistMult followed by a sigmoid function generates the link weights.

It is noticeable that it is not computationally more expensive than well-established machine learning models such as Logistic Regression, which employs $O(m \cdot f)$, where m is the total number of edges and f is the number of features.

4.3 Construction of the complete SD

In this section, we discuss our approach for the generation of a complete SD of a table T according to the ontology O .

4.3.1 Construction of the initial SD

In the first step, we take into account the annotations obtained for the table columns. For each column col that has been annotated with a pair (C, p) , we check in U_{C_s} whether a node u_{C_s} exists in SD (i.e. another column has been annotated with the same concept C). If it does not exist or it exists and p is an identifying property for the concept C , then: *i*) a new node u_c^{h+1} is created in U_{C_s} (where h is the number of nodes associated with concept C); *ii*) a new node u_t^h is created in U_T ; and, *iii*) the edge (u_c^h, p, u_t^h) is included in E_T . Otherwise (i.e. the node u_c already exists and p is not identifying), a new node u_t is created in U_T , and the edge (u_c, p, u_t) is included in E_T . Note that, non-identifying properties are always added to the node corresponding to the last processed concept. For example, suppose we have two nodes u_C^1 and u_C^2 associated with the same concept C , and p_1, p_2, p_3, p_4 are properties for the concept C , and p_1 and p_3 are identifying properties, then p_2 is associated with the node u_C^1 , whereas p_4 is associated with the node u_C^2 . Once completed this activity, we check the presence of two nodes u_{C_i} and u_{C_j} , such that $C_i \sqsubseteq C_j$ (with the same degree of repetition h). In such case, u_{C_j} can be removed and its edges

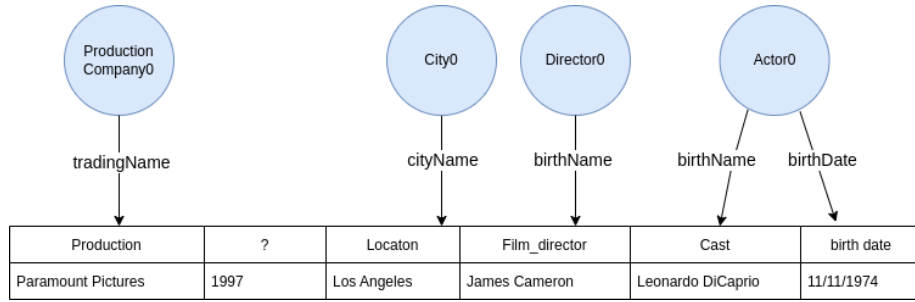


Figure 4.4: Initial SD for our running example

moved to u_{C_i} . Thus, only the most specific concepts are maintained in SD .

Example 22 Figure 4.4 shows the initial SD obtained from the annotations in Figure 4.2. U_{C_s} contains the nodes $u_{City}^0, u_{ProductionCompany}^0, u_{Director}^0, u_{Actor}^0$ whereas U_T contains the nodes $u_{production}, u_{location}, u_{film_director}, u_{cast}, u_{birthdate}$. Among them, $u_{ProductionCompany}^0$ and $u_{Director}^0$ are variable meta-instances because in our context a production company is identified through the `tradingName` and `foundingYear` properties and a director through the `birthName` and `birthDate` properties. Note that the relations among the concepts are missing and SD does not report whether `location` is the headquarters of the production company, the place of birth of the director, or the place where an actor lives. \square

4.3.2 Requirements for the complete SD

The initial SD contains meta-instances u_{C_1}, \dots, u_{C_n} for which properties in the table have been identified. Starting from the associated concepts, the functions $classes_{\mathcal{O}}^{\Delta}$ and $closure_{\mathcal{O}}^{\Delta}$ (presented in Section 2.2 of Chapter 2) can identify the concepts and relationships that could be potentially taken into account for the introduction of nodes and edges in the complete SD . Before presenting the algorithm that we have developed for this purpose, we introduce and explain some requirements that are at the base of the algorithm.

Requirement 1 The meta-instances contained in SD represent the most specific concepts among those contained in $classes_{\mathcal{O}}^{\Delta}(\{C_1, \dots, C_n\})$. \square

This requirement points out that the inheritance relationships among classes are resolved by including in SD only nodes corresponding to the most specific class of the inheritance hierarchy among those in $classes_{\mathcal{O}}^{\Delta}(\{C_1, \dots, C_n\})$ with only the exception of the nodes belonging to the initial SD (that cannot be specialized). This requirement has the purpose to reduce the nodes to include in SD and associate to the introduced nodes all the direct and inherited relationships of the ontology. The exception is only for the nodes that occur in the initial SD because they are established in advance and validated by the user.

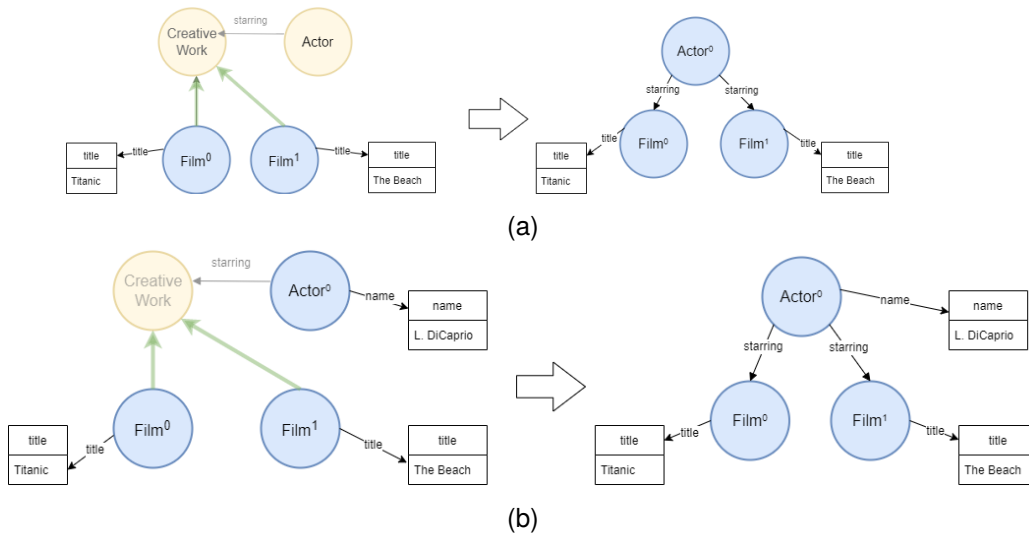


Figure 4.5: Basic cases in the management of inherited relationships

To better explain this requirement, consider the use-case in Figure 4.5a which is tailored for the management of meta-instances of the same concept (but applies in general). Suppose that in the initial SD we have two nodes (the blue ones in the figure) representing the films *Titanic* and *The Beach*. Relying on the ontology \mathcal{O} , these concepts are related to the concept *Actor* by means of the relation *starring* in which the parent concept *CreativeWork* is involved (these nodes are colored in yellow because they are induced by the ontology but do not belong to SD). In this case, a single node u_{Actor} is included in SD and the *starring* relation is included in both films (right part of the figure). Since no column of the table is associated with u_{Actor} , this is a variable node, that is it can refer to the same actor or two distinct actors that were starring in one (or both) films. The behaviour is similar in the second situation reported in Figure 4.5b, but in this case an identifying property is present for *Actor*. Even if the generated SD is similar to the previous case, u_{Actor} is a specific node and thus its interpretation is that the actor is starring in both films.

Requirement 2 *When multiple nodes represent the same concept, relations among them should be rationalized.* \square

The need for this requirement is explained through the use-case in Figure 4.6. In this case, specific properties are present for two actors and two films. Therefore, the new SD will contain the relationship between all pairs of meta-instances with the same degree (i.e. $Actor^0$ with $Film^0$ and $Actor^1$ with $Film^1$). This is an arbitrary choice whose purpose is to limit the explosion of combinations of relationships among the involved meta-instances. When the

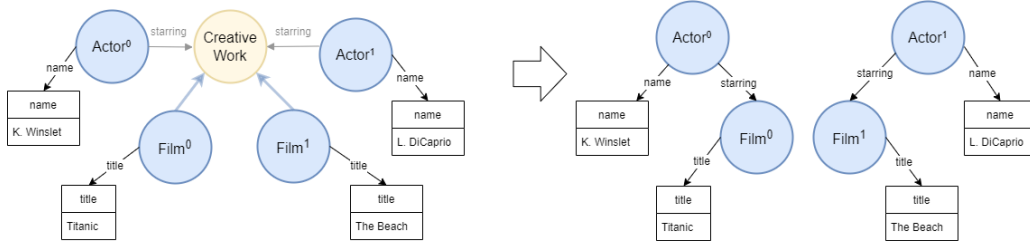


Figure 4.6: Distribution of meta-instances among duplicated concepts

number of nodes representing actors and films is not equal, the last node corresponding to the minimal number of occurrences should be associated with all the remaining unmatched nodes. This means that, if we have 3 nodes representing actors and 5 nodes representing films, we create the relations $Actor^0$ with $Film^0$, $Actor^1$ with $Film^1$, and $Actor^2$ with $Film^2$, $Film^3$, and $Film^4$.

Requirement 3 *Self-relations in $closure_{\mathcal{O}}^{\Delta}(\{C_1, \dots, C_n\})$ should be included only among distinct meta-instances.* \square

In our environment, self-relationships (i.e. relationships involving instances of the same concept) are introduced only when SD contains different meta-instances of the same concept (e.g. the relation spouse is included only when there are at least two instances of the concept Person). There are two reasons for this behaviour. First, the presence of two instances means that the table contains information for two distinct real-world entities. Then, self-relationships on the same meta-instance would be eliminated by the algorithm for the construction of the concise SD .

4.3.3 Generative algorithm of the complete SD

Algorithm 2 starts from the initial SD and generates the complete version that adheres to the requirements previously outlined. For each u_C^h of the initial SD , it determines the relationships at a distance Δ in which concept C can be involved by means of the function $closure_{\mathcal{O}}^{\Delta}$. Let (C_1, r, C_2) be one of these relations (the behaviour of the algorithm is the same for all relations). The algorithm determines the list of nodes in SD that can be a source of the relation (denoted U_s) and those that can be the target of the relation (denoted U_t). These nodes can be already included in SD or they can be introduced relying on the function $CREATELISTNODES$ (that will be discussed shortly). The pairs $(u_{C_s}^j, u_{C_t}^k) \in U_s \times U_t$ are the potential pairs of nodes for which the relation r could be introduced.

A complex condition (line 10) is evaluated that relies on the maximum number of occurrences of concept C_s and C_t (denoted respectively J and K). First, $u_{C_s}^j$ and $u_{C_t}^k$ should be

Algorithm 2 Inclusion of nodes and edges in SD

Input: $SD = (U_{C_s}, U_T, E_R, E_T)$ the initial semantic description
 O the domain ontology, Δ path length (default 1)
Output: SD updated with further nodes and edges relying on O

- 1: $U_{C_s}^{ini} := U_{C_s}$
- 2: $\mathcal{C}l := \text{classes}_{\Delta}^{\Delta}(U_{C_s}^{ini})$
- 3: **for** $u_C^h \in U_{C_s}^{ini}$ **do**
- 4: **for** $(C_1, r, C_2) \in \text{closure}_{\Delta}^{\Delta}(C)$ **do**
- 5: $U_s := \text{CREATELISTNODES}(u_C^h, C_1, C_1 = C_2, \mathcal{C}l)$
- 6: $U_t := \text{CREATELISTNODES}(u_C^h, C_2, C_1 = C_2, \mathcal{C}l)$
- 7: **for** $(u_{C_s}^j, u_{C_t}^k) \in U_s \times U_t$ **do**
- 8: Let J/K be max number of occurrences of concept C_s/C_t in U_{C_s}
- 9: **if** $(u_{C_s}^j \neq u_{C_t}^k \wedge (u_{C_s}^j, r, u_{C_t}^k) \notin E_R \wedge (u_{C_t}^k, r, u_{C_s}^j) \notin E_R \wedge (C_s = C_t \vee j = k \vee$
- 10: $(J \leq K \wedge j = J - 1 \wedge k > j) \vee (J > K \wedge k = K - 1 \wedge j > k))$ **then**
- 11: $E_R := E_R \cup \{(u_{C_s}^j, r, u_{C_t}^k)\}$
- 12: **return** SD

- 13: **function** $\text{CREATELISTNODES}(u_C^h, C_1, \text{selfLoop}, \mathcal{C}l)$
- 14: $list := []$
- 15: **if** $C \sqsubseteq C_1$ **then**
- 16: **if** (selfLoop) **then** $list := [u_{C_0}^0, \dots, u_{C_J}^J]$ **where** $\bar{C}_j \sqsubseteq C_1, 0 \leq j \leq J$
- 17: **else** $list := [u_C^h]$ **end if**
- 18: **else**
- 19: **if** $u_{C_1}^0 \notin U_{C_s}$ **then**
- 20: **if** $\forall u_{\bar{C}} \in U_{C_s}, C_1 \not\sqsubseteq \bar{C} \wedge \bar{C} \not\sqsubseteq C_1$ **then**
- 21: $U_{C_s} := U_{C_s} \cup \{u_{C_1}^0\}$
- 22: $list := [u_{C_1}^0]$
- 23: **else**
- 24: **for all** $\bar{C} \sqsubseteq C_1$ s.t. $\bar{C} \in \mathcal{C}l$ **do**
- 25: **if** $u_{\bar{C}}^0 \notin U_{C_s}$ **then** $U_{C_s} := U_{C_s} \cup \{u_{\bar{C}}^0\}$
- 26: $list := list + [u_{\bar{C}}^0, \dots, u_{\bar{C}}^K]$ **where** K occurrences of \bar{C} in U_{C_s}
- 27: **for** $u_{\bar{C}}^k \in U_{C_s}$ s.t. $C_1 \sqsubseteq \bar{C} \wedge u_{\bar{C}}^k \notin U_{C_s}^{ini}$ **do**
- 28: Rename $u_{\bar{C}}^k$ as $u_{C_1}^k$
- 29: $list := list + [u_{C_1}^k]$
- 30: **else**
- 31: $list := [u_{C_1}^0, \dots, u_{C_1}^K]$ **where** K occurrences of C_1 in U_{C_s}
- 32: **return** $list$

distinct nodes and the relation r is not already included in SD (in any direction). Then, one of the following conditions should be verified:¹

¹The conditions 2, 3, and 4 are introduced for guaranteeing Requirement 2.

1. $C_s = C_t$, (i.e. a self-relation is considered, and according to Requirement 3, the relation should be included among all the meta-instances of the same concept).
2. $j = k$, the relation r should be included between nodes representing the same number of occurrences.
3. The occurrences of C_s are less than those of C_t (i.e. $J < K$), the current $u_{C_s}^j$ is the last occurrence (i.e. $j = J - 1$), and the occurrence of $u_{C_t}^k$ is higher than j (i.e. $k > j$).
4. The occurrences of C_s are greater than those of C_t (i.e. $J > K$), the current $u_{C_t}^k$ is the last occurrence (i.e. $k = K - 1$), and the occurrence of $u_{C_s}^j$ is higher than k (i.e. $j > k$).

When one of these conditions holds, the edge $(u_{C_s}^j, r, u_{C_t}^k)$ is included in SD .

Function `CREATELISTNODESIS` used for creating the list of meta-instances to be associated with the source/target of a relation (C_1, r, C_2) . The behaviour of the function is the same for the source and target concept, and in the remainder, we describe the treatment of the source concept. Function `CREATELISTNODES` considers the current node u_C^h of the initial SD , the source concept C_1 , a Boolean value indicating when the relation is a self-relation, and the set of all possible concepts that can be introduced for the current table T . If the class C of the current node inherits from C_1 (line 19), the list of nodes to include depends on the kind of relation. When it is a self-relation (line 20), all the nodes in SD that inherit from C_1 are included in the list. Otherwise (line 21) only the current node u_C^h is included in the list. Through this case, we have handled the situation in which the current node u_C^h can be directly associated as the source of the relationships. When the condition at line 19 is not verified, the algorithm looks for other nodes in SD that can be associated as a source for the relation. It first checks whether at least a node already exists in the SD of class C_1 . In the positive case, all the instances of C_1 are included in the list (line 35). In the negative case, the algorithm checks (line 24) whether other nodes exist in the SD which are bound through the inheritance relation with C_1 (either C_1 inherits from the concept of this node, or the concept of this node inherits from C_1). If a node with such a property is not identified, a new node for the concept C_1 is introduced in the SD and included in the list. Otherwise, (line 28), the two following disjoint cases should be handled.

- SD contains nodes whose concepts inherit from C_1 . In this case, it means that a more specific class than C_1 is already included in SD . Therefore, for guaranteeing Requirement 1, the algorithm looks for sub-concepts \bar{C} of C_1 that can be potentially included in SD (belonging to \mathcal{Cl}). If a node is not already included in SD for \bar{C} , it is added. In any case, all the instances of \bar{C} in SD are included in the list of sources.

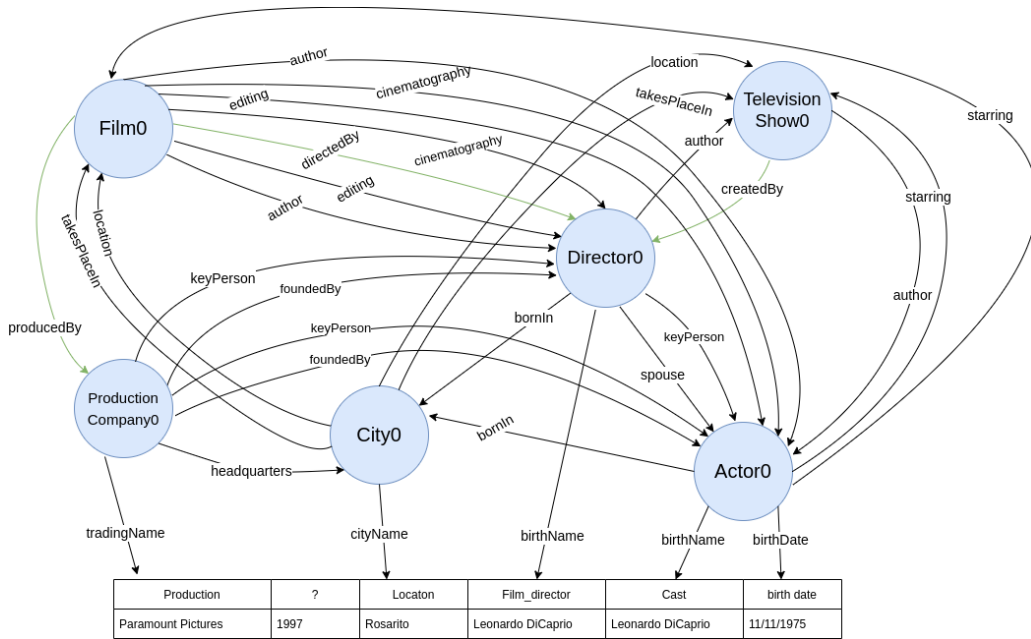


Figure 4.7: Complete SD obtained through Algorithm 2

- SD contains nodes $u_{\bar{C}}$ such that C_1 inherits from \bar{C} and these nodes do not belong to the initial SD . In this case, the concept \bar{C} is substituted with C_1 (which is more specific) and the node is introduced in the list of source nodes. Note that only new introduced nodes can be specialized because they are introduced by our algorithm.

At the end (line 41), the list of sources is returned to the main algorithm.

Example 23 Figure 4.7 shows the SD obtained from the one in Figure 4.4 by applying Algorithm 2. Two colors are used for representing the level of specificity of the relations between two meta-instances. The obtained graph contains all the possible relations that might exist in the considered domain for the reported concepts. Note that the city can be both the production company's headquarter, the place where an actor or a director was born, and the place where a film was developed. \square

The following lemma introduces some properties of the complete SD obtained through the algorithm.

Lemma 4.3.1 Let $SD = (U_{C_s}, U_T, E_R, E_T)$ be the graph obtained by Algorithm 2. The following properties hold on SD .

1. Two distinct nodes $u_C^1, u_C^2 \in U_{C_s}$ are associated with the same concept C if there

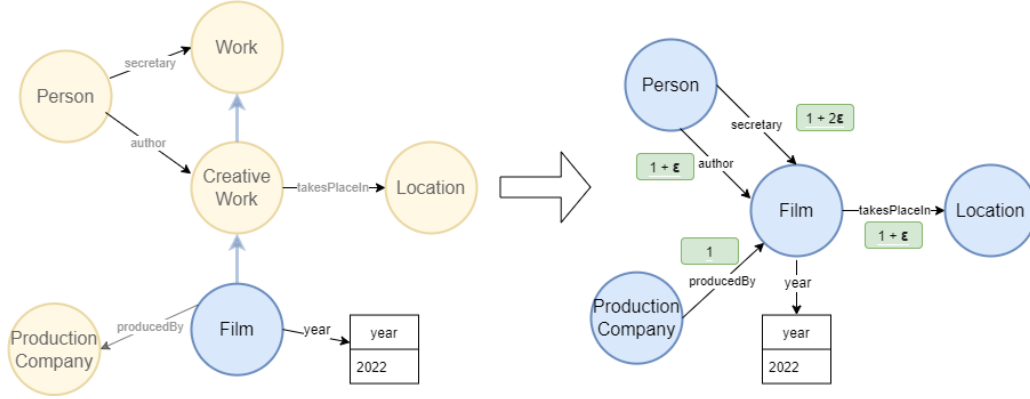


Figure 4.8: Weights to assign in presence of \subseteq relationships

exist two distinct nodes $u_{T_1}, u_{T_2} \in U_T$ s.t. $(u_C^1, p, u_{T_1}), (u_C^2, p, u_{T_2}) \in U_T$ and p is an identifying property for the concept C .

2. for each $i_C \in U_{C_s}$, $C \in \text{classes}_O^\Delta(U_{C_s}^{\text{ini}})$.

3. for each self-relation r that involves a node $u_C \in U_{C_s}$, an edge (u_C^h, r, u_C^k) is included in E_R only if $h \neq k$. \square

Proof The first point of the lemma claims that distinct nodes associated with the same concept occur in SD produced by Algorithm 2 only when they are associated with distinct columns of the table T . According to the algorithm for the construction of the initial SD , two nodes associated with the same concept C are introduced when table columns present different identifiers (or different identifying columns). We have to prove that no further nodes associated with the concept C are introduced by the application of Algorithm 2. However, new nodes are introduced by the `CREATELISTNODES` function only when processing a triple (C_1, r, C_2) and no nodes of SD is associated with the concept C_1 (or C_2) or one of its descendant (block of instructions between line 21 and 26). Therefore, the first claim holds for each step of Algorithm 2 and thus holds in general. The second claim is a direct consequence of the structure of Algorithm 2. Indeed, nodes $u_C \in U_{C_s}$ are introduced because they belong to $U_{C_s}^{\text{ini}}$ or because they are associated to a concept with which C is related according to a relation r at a distance Δ . However, this is actually the definition of $\text{classes}_O^\Delta(U_{C_s}^{\text{ini}})$. The last claim of the Lemma follows from the condition at line 9 of Algorithm 2 for the construction of triples. A triple is introduced only if the starting and ending nodes of the triple are distinct, not yet included in E_R , and (for the matter of this claim) they represent the same concept. Therefore, self-relationships on the same node are not allowed in the complete semantic description. \square

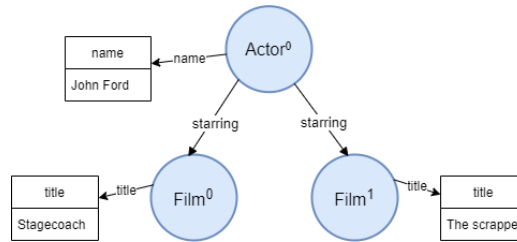


Figure 4.9: Different weights assigned to multiple instances of the same concept

4.4 Weighting systems

Starting from the complete SD we have just obtained, we need to choose the one that is the most likely according to the organization of the ontology and the current instances of the consolidated knowledge graph. For this reason, we introduce different weighting systems through which the relations on the complete SD can be properly weighted.

4.4.1 Ontology-based weighting system

Relations occurring in SD should be properly weighted by taking into account their specificity in the ontology specification. Indeed, relations can be declared for the concept they represent or can be inherited from more general concepts. By taking into account this principle, a weight of 1 is associated with all the edges in E_T because they are specified in the initial SD (and we consider them as validated) and thus their weight should be minimal. Moreover, a weight is assigned to the edges in E_R compliant with the following requirement.

Requirement 4 *Weights on relations should depend on their specificity: inherited relations present higher weights when the linked concepts do not contain properties in the table.* □

This requirement is explained by means of the following use cases. Figure 4.8 shows a node in SD associated with the concept `Film` for which one of its properties is associated with a table column. By means of the *closure* function, the edges and the nodes reported in the figure can be identified relying on the inheritance relationship (for the sake of simplicity we consider only the one reported in the figure). The relation `producedBy` directly involve `Film`, so the weight can be 1. By contrast, The relations `author` and `takesPlaceIn` are related to the parent concept `CreativeWork` and thus should be penalized. A further penalty should be applied to the `secretary` relation because it is related to the ancestor concept `Work`. The penalty should be proportional to the distance.

Requirement 5 *When several instances of the same relation outcome from one node of SD , different weights should be applied. Lower weights should be assigned to relations involving meta instances with the same index.* □

In our setting, SD can contain several meta-instances representing the same concept. Through this requirement, we wish to assign higher weights to relations among different occurrences as described in Figure 4.9. In this use-case, the weight of starring between Actor^0 and Film^1 should be penalized.

Definition 1 (Ontology-based weighting system) Let ϵ be a penalty on the specificity of the relationships and γ the penalty for relations between meta-instances with different instances. Let $SD = (U_{C_s}, U_T, E_R, E_T)$ be the result of Algorithm 2 relying on the ontology O . The weight function $weight_O$ is defined as follows. For each $e_T \in E_T$,

$$weight_O(e_T) = 1$$

For each $e_R = (i_{C_s}^j, r, i_{C_t}^k) \in E_R$, let $(C_1, r, C_2) \in \mathcal{R}$ be the specification of r in O , and $d_1 = dist(C_1, C_s)$, $d_2 = dist(C_2, C_t)$ the corresponding distances:

$$weight_O(e_R) = \begin{cases} 1 & \text{if } d_1 = d_2 = 0, j = k \\ 1 + \max(d_1, d_2) \cdot \epsilon & \text{if } d_1 + d_2 > 0, j = k \\ weight((i_{C_s}^0, r, i_{C_t}^0), C_1, C_2) + \gamma & \text{if } j \neq k, (C_1, r, C_2) \in \mathcal{R} \end{cases} \quad \square$$

Example 24 Consider the SD in Figure 4.9. The relation starring is specified in \mathcal{O} as $(\text{Actor}, \text{starring}, \text{CreativeWork})$. The triple $(\text{Actor}^0, \text{starring}, \text{Film}^0)$ receives the weight $1 + \epsilon$ because Film is a direct sub concept of CreativeWork . An extra weight γ is associated with $(\text{Actor}^0, \text{starring}, \text{Film}^1)$ to show a preference to the first relation. \square

4.4.2 KG-based weighting systems

The weighting system proposed in the previous section does not take into account the consolidated knowledge graph KG_C . For this reason, we introduce other two weighting systems that rely on the functions introduced in Section 4.2.

Definition 2 (KG-based weighting system) Let $SD = (U_{C_s}, U_T, E_R, E_T)$ be the result of Algorithm 2 and \mathcal{S}_{KG} be the scoring function introduced in Section 4.2. For each $e \in E_R \cup E_T$, the weight function $weight_{KG}$ is defined as follows:

$$weight_{KG}(e) = \begin{cases} 1 - \mathcal{S}_{KG}(e) & \text{if } e \in E_R \\ 1 & \text{if } e \in E_T \end{cases} \quad \square$$

This weight function relies only on the information obtained through the embedding of KG . It is a values between 0 (low probability of the presence of the edge e) and 1 (high probability of the presence of the edge e).

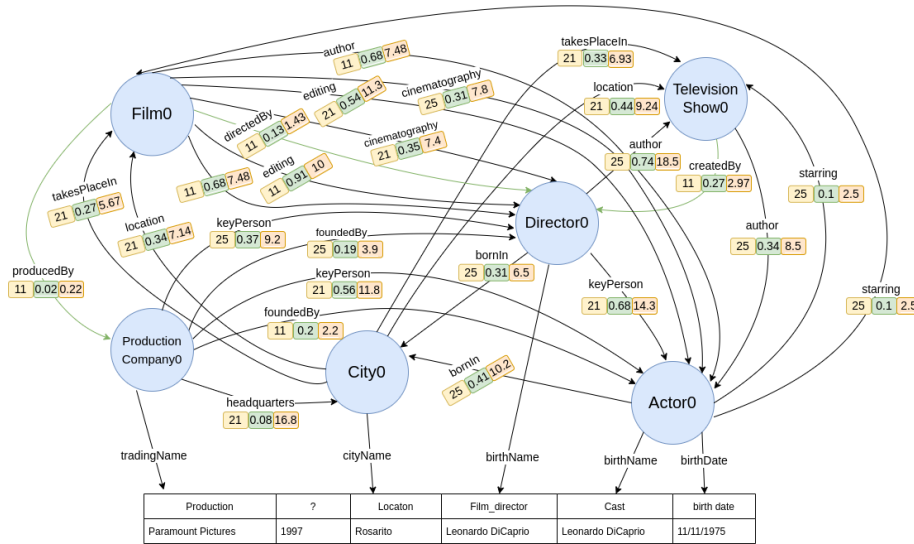


Figure 4.10: Complete SD with weights

Another weighting system that we consider is the one that combines the two previous weighting systems. The idea is to mitigate the contribution of the weight assigned by the ontology by considering the probability of its occurrence relying on the current status of KG .

Definition 3 (Combined weighting system) A combined weighting system can be obtained through the combination of previous ones. For each $e \in E_R \cup E_T$,

$$weight_{Comb}(e) = \begin{cases} weight_O(e) \cdot weight_{KG}(e) & \text{if } e \in E_R \\ 1 & \text{if } e \in E_T \end{cases}$$

□

Example 25 Figure 4.10 shows the triples of weights that are assigned to the relations on the graph reported in Figure 4.7 by exploiting the three weighting systems that we have proposed in our running example. The first component (yellow box) contains the ontology-based weight, the second component (green box) contains the KG-based weight, whereas the last component (red box) the combined weight. Since quite often relations present the same ontology-based weight (this is due to the organization of the relation through the inheritance relation), the use of the KG-base weighting system allows us to give more relevance to those with the highest frequency. □

4.5 Inclusion of properties for unmatched table columns

As outlined at the beginning of the chapter, some columns of the table might not be associated with meta-instances in SD . However, at the current stage, SD contains all the concepts that can be related to concepts identified in table T (at a distance Δ). Therefore, one of the properties of these concepts could be associated with the column when its domain is compatible with the one in SD .

By exploiting our embedding system that includes also the basic properties with atomic values, we provide an approach for trying to detect the property that can be associated with unmatched columns that take into account the following requirements.

Requirement 6 *In case SD contains distinct node-instances associated with the same concept C , an unmatched column of T can be associated to at most one of them.* \square

Our predicting model is not able to discriminate when the same property (e.g. `birthDate`) can be associated with one or another node instance of SD associated with the same concept (e.g. `Person0` or `Person1`). In this case, a single property is introduced. Among the possible instance nodes, we choose the one whose identifier (or identifying property) is positioned closer to the unmatched column. However, when the distance is the same, an arbitrary choice is made.

Requirement 7 *In the complete SD , an unmatched column can be associated with many properties, but the weights on the edges should be all distinct and guarantee that just a single property will be included in the concise SD .* \square

This requirement has the purpose to treat the unmatched columns, for which a property can be potentially detected, in the same way as the other columns, for which a property has been identified in the initial SD (i.e. they can be associated with exactly a single class). To guarantee this requirement, a very high weight should be associated with the identified properties. Otherwise, our technique for the minimization of the complete SD would erroneously introduce more than one property that is associated with the unmatched column and thus alter the structure of the concise SD .

The following weighting function is used for determining the weight of unmatched columns.

Definition 4 (Weight for unmatched columns) *Let u_t be a node in SD corresponding to an unmatched column in T . Let p_i be a property of $u_c^k \in SD$ among the N properties of the instance nodes of SD that can be associated with u_t . The weight to assign to (u_c, p_i, u_t) is:*

$$weight_{unmatched}((u_c, p_i, u_t)) = 1000 * |E_C| + (2 - S_{KG}(u_c^h, p_i, u_t)) + \frac{i}{1000 * |E_C|} \quad \square$$

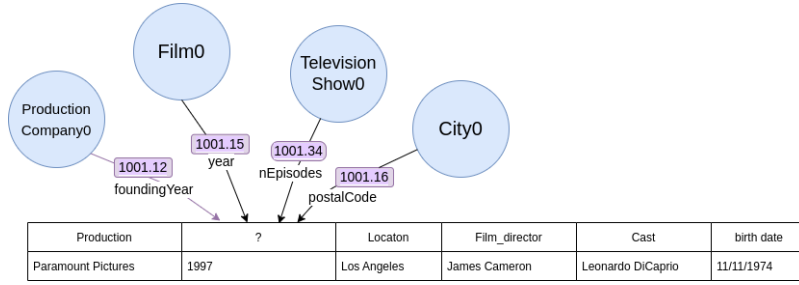


Figure 4.11: Possible properties for the unmatched column of T

In the formula of the weighting function, we can identify three parts. First, a value higher than all the weights can be associated with relations in SD . This value is constant for SD and guarantees that only one of the edges $(u_c, _, u_t)$ can be chosen through our minimization approach for the generation of the concise SD . Then, a value that depends on embeddings (\mathcal{S}_{KG}) and is higher than the one included in the initial SD for basic properties. This value is lower when the score is higher. Finally, a very small addendum is included that depends on the position of the identified property and guarantees that all the edges incident in u_t present different weights and would not alter the choice of a specific property.

Starting from our requirements and the definition of the weighting function, we can now sketch the algorithm for treating unmatched columns of T . For each node $u_C^h \in U_{C_s}$ and for each node $u_t \in U_T$ such that $(u_C^h, _, u_t) \notin E_T$, we consider the properties $P \subseteq P(C)$ that are compatible with the column corresponding to u_t . A property $p \in P$ is compatible with u_t when their basic types are identical, the values assumed in T are in the range of possible values specified for p , and p is not yet associated with an outgoing edge from u_C^h . The function $Properties_{KG}$ (introduced in Section 4.2) is then applied to the properties in P for determining the one with the highest priority ($\bar{p} = Properties_{KG}(u_C^h, u_t, P)$) for each u_C^h and the triple (u_C^h, p, u_t) is included in the set $Matching$ when the value $\mathcal{S}_{KG}(u_C^h, \bar{p}, u_t)$ is above the minimal prediction threshold θ . The set $Matching$ should be scanned for identifying triples of the form (u_C^h, p, u_t) and (u_C^k, p, u_t) with $h \neq k$, and only one of the two should be maintained (to guarantee Requirement 6). At this point, the set $Matching$ contains the triples corresponding to the properties p_1, \dots, p_N that can be potentially associated with the unmatched column u_t . Therefore, SD is extended with the edges $(u_C^h, p_1, u_t) \dots (u_C^h, p_N, u_t)$ and each edge is weighted with the function $weight_{unmatched}$ of Definition 4.

Example 26 *In the running example, the second column does not have a semantic annotation. By taking into account the identified basic type, it is possible to look for the compatible properties that are associated with the meta-instances occurring in the complete SD . Figure 4.11 shows the compatible properties and the weights that are assigned according to the*

weighting system so far discussed. For each meta-instance, we select only one property. Moreover, we exploit the minimum and maximum values used as features of the model and we prune those that are not in the range (e.g. children). \square

4.6 Generation of the concise SD

Starting from the complete SD whose edges have been weighted according to one of the proposed weighting systems, we now wish to extract a concise SD representing the actual relations that hold among the table columns.

In literature, the problem of extracting the concise SD from the complete one has been modelled as the Steiner tree problem [65, 154]. Specifically, given an undirected graph $G = (V, E)$ with non-negative edge weights and a subset of nodes U ($U \subseteq V$) denoted terminal nodes, the Steiner tree problem in graphs consists in the identification of a tree of minimum weight that contains all the terminal nodes U (but may include additional nodes). Even if this problem is NP -hard, there exist approximation algorithms able to create sub-optimal trees in polynomial time.

In our setting, the terminal nodes are those in U_T that are bound to meta-instances, and we wish to keep in the tree only the relations with the lower weights obtained through one of the weighting systems previously described independently from the direction of navigation.

Example 27 *Figure 4.12 shows the Steiner tree obtained from SD in Figure 4.10. All the paths that end in one of the table columns with the minimal cost are considered by the algorithm and those that can be included in a minimal spanning tree are included in the concise SD . Note that, in the unmatched column only one of the incident properties is selected. In this specific case, the obtained SD coincides with the expected semantic description. \square*

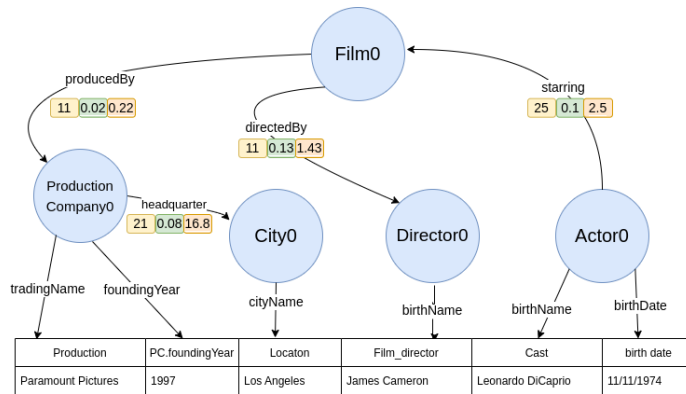


Figure 4.12: Concise SD

4.7 Experimental evaluation

The purpose of our experiments are *i)* the validation of the GNN model compared to the baseline approaches; *ii)* a discussion of the characteristics of the generated concise SD s and the possibility of identifying unmatched columns occurring in the initial SD . The experiments reported in this chapter can be reproduced using the code published in the GitHub repository ². On the first point, we chose to evaluate the GNN models on the link prediction task, as their final goal is to emit in output a weighting system for links, using the area under the receiver operating characteristic curve (AUROC) as the evaluation metric.

Whereas for the second point, we propose a new approach to check the effectiveness of the generated concise SD s by comparing them with the corresponding human-curated ground truth, using precision and recall on the set of triples in the semantic descriptions as the evaluation metrics.

The next subsections describe and discuss in depth the results we obtained.

Implementation details. The entire approach described in this chapter has been implemented in Python. The consolidated knowledge graph KG_C is stored in n-triple format and the ontology is stored in turtle format. The manipulation of these triples is done by means of the Python Library RDFlib³, which allows the representation of semantic triples in a graph, and the usage of SPARQL language for querying \mathcal{O} and KG_C . A graph database (AllegroGraph,⁴ has also been used in place of the n-triple file for testing purposes since

²https://github.com/SaraBonfitto/SAGA_tab

³<https://rdflib.readthedocs.io/en/stable/>

⁴<https://franz.com/agraph/support/documentation/current/python/api.html>

Datasets	Ontology				Consolidated Knowledge Graph				
	$ \mathcal{C} $	$ \mathcal{R} $	$ \mathcal{P} $	$\frac{ \mathcal{P} }{ \mathcal{C} }$	$ V $	$ type(E) $	$ E $	$ Prop $	$\frac{ Prop }{ V }$
Movie-set [9]	11	15	26	2.5	19,294	14	27,728	590,736	2.64
Area-set [18]	11	27	94	8.4	53,549	12	76,153	164,564	3.07
PP-set [65]	33	72	40	1.2	12,598	6	12,562	42,114	3.34

Table 4.2: Data sets considered for the evaluation of the developed approach

hyperparameter	value	hyperparameter	value
number of layers	2	hidden dimension	4
optimizer	Adam [100]	learning rate	0.01
weight decay	$5e - 4$	number of epochs	500

Table 4.3: Hyperparameters of HeteroGNN model for the link prediction task.

the SPARQL queries performed on the KG_C can be very slow using the RDFlib Library). Finally, the complete semantic description has been represented as a multiedge-directed graph using the NetworkX library ⁵ and their Steiner tree function has been exploited. The training and testing of the GNN model, described in Section 4.2, has been implemented by exploiting the library Pytorch Geometric (PyG) [63]. We reported the hyperparameter configuration of our model in Table 4.3. The implementation of our approach, the code to reproduce the experiments, and the datasets are available on a GitHub repository.

Datasets. For our experiments the following three datasets have been considered.

- **Movie-set.** The dataset was extracted from DBPedia [9] and contains information about movies, TV series, and actors and is a superset of our running example.
- **Area-set.** The dataset is made available by [18] and contains information about invoices to be rescued from different debtors of a debt collector agency.
- **Public procurements (PP-set).** The dataset was collected by [65] and is about public procurements, it contains tenders that are bid by business entities and are related to an offer.

These three real-world heterogeneous graph datasets are characterized by the presence of consolidated knowledge graphs $KG_C = (V, E, R, Prop)$ that adhere to the constraints imposed by the ontology $\mathcal{O} = (\mathcal{C}, \mathcal{R}, \mathcal{P}, R, P, \sqsubseteq)$. Table 4.2 reports their characteristics.

⁵<https://networkx.org/documentation/stable/reference/classes/multidigraph.html>

method	GNN layer	node features	nodes	edges
HeteroGNN	HeteroGAT	yes	heterogeneous	heterogeneous
MRGCN	R-GCN	yes	homogeneous	heterogeneous
SeMi	R-GCN	no	homogeneous	heterogeneous

Table 4.4: GNN models compared in terms of GNN layer, use of node features (i.e. properties), representation of nodes and edges.

4.7.1 Validation of the GNN model

We have validated our GNN model against some baseline models by describing *i)* a comparison of their prediction performances on the multi-relation link prediction task on the three datasets; *ii)* an empirical evaluation of their execution time that use node properties; *iii)* the results on the new proposed property detection (i.e. link classification) task. After presenting the characteristics of the baseline methods, we discuss the experimental results.

Baselines. HeteroGNN has been compared with two baseline methods: MRGCN [169], which is a multimodal approach that extends the basic R-GCN model [146], and SeMi [65], which exploit an R-GCN model [146] for the generation of concise *SD*. Both approaches exploit R-GCN as an encoder and DistMult as a decoder function. The implementations of the models consider the settings in their original papers in terms of hyperparameters and neural architectures. Table 4.4 summarizes their main characteristics.

- MRGCN [169]. The initial nodes’ embeddings are obtained by concatenating the identity vectors, which are used to capture the structural information, with nodes’ feature embeddings learned using dedicated neural encoders. In this way, they obtain multimodal node embeddings, which are fed to R-GCN. Despite being able to handle heterogeneous properties (e.g. string, dates, or numerical information), the method does not consider heterogeneous representation for nodes.
- SeMi [65]. It can handle multi-relational data but it does not take into account heterogeneous nodes (i.e. the representation of all the nodes is the same in terms of feature types and dimensions) and the node features.

Multi-relational link prediction. For performance evaluation on the link prediction task, since the outputs are the results of the application of a sigmoid function on link scores, AUROC score is used to compare their performance. Another reason to choose the AUROC score is the robustness of ROC curves and their associated areas to class imbalance, a problem that affects the link prediction domain. Yang *et al.* [173] suggest avoiding using accuracy, precision and recall, and fixed threshold metrics such as top K predictive rate.

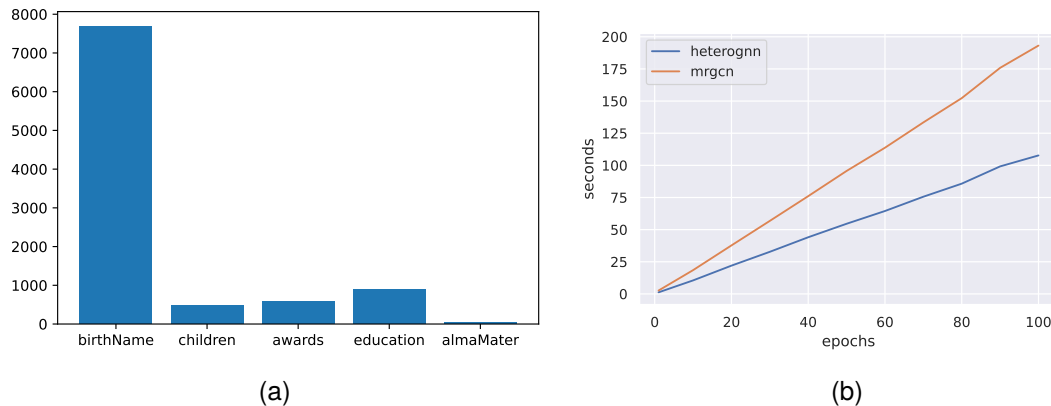


Figure 4.13: (a) Histogram of the distribution of possible relations between an Actor and a string property in Movie-set. Five unbalanced classes can be assigned. (b) Training time (in seconds) of HeteroGNN and MRGCN models for multi-relational link prediction task on Movie dataset, in function of the number of epochs. HeteroGNN takes about half of the execution time of MRGCN.

We perform a random link split for each edge type with 20% of the edges assigned to the test set and negative sampling on both train and test sets. In this way, the performance of the models can be evaluated also on a negative set (i.e. a set of triples that does not exist in the KG). The random selection of a subset of edges from the original complete set is one of the most common methods to perform test set sampling. Despite this strategy may conduct in over-optimistic results, there are evaluation measures (e.g. AUROC) for which subsampling negatives from the test set has no negative effects.

Table 4.5 reports the AUROC scores on the test set for the considered methods. For our model, we report the results both using node properties and not using them. HeteroGNN outperforms the other methods for all the datasets. SeMi has very low performances due to the rich heterogeneity of the KGs; however, it is not equivalent to flipping a coin because we are studying which kind of relationships exist between two entities, not only if the relation exists. Whereas MRGCN achieves better results than our model without node properties on the two most heterogeneous datasets (i.e. datasets with a large number of different kinds of properties and relations), showing the importance of node features.

Property detection. Property detection is a link classification task in which we assign the correct label to a link, that exists for sure, between an entity and a property in a KG. Given two certain entities and property types, different kinds of relation labels between them can exist. As an example, Figure 4.13a reports the distribution of the properties that can exist

Datasets	SeMi	MRGCN	HeteroGNN _{Struct}	HeteroGNN
Movie-set	0.55	0.67	0.757	0.84
Area-set	0.5	0.77	0.69	0.91
PP-set	0.51	0.67	0.626	0.81

Table 4.5: AUROC scores on the three datasets.

Datasets	HeteroGNN
Movie-set	0.8661
Area-set	0.98
PP-set	0.8448

Table 4.6: AUROC of HeteroGNN for property detection

between an Actor and a string value in Movie-set. We can observe that we can assign 5 possible unbalanced classes to a link between an Actor and a string property.

We evaluate the prediction performance of our model on property detection tasks using a link prediction setting. As described in Section 4.2, we detect properties by assigning the label of the relation that achieves the best link prediction scores, among the possible ones relatable to a certain entity. Therefore, the AUROC score can be computed on the triples in which properties are involved to estimate the overall goodness of the predictions (i.e. without treating each link classification task between an entity and its properties distinctly). Link splitting and negative sampling are computed in the same way as on the multi-relational link prediction task. Note that since we evaluated property detection in a link prediction setting with balanced negative samples, the AUROC score does not suffer the possible class imbalance on the different link classification problems.

The AUROC scores on the test set of the three datasets have been computed for our model on the property detection task and reported in Table 4.6. The results highlight the feasibility of this kind of approach for the property detection task. We achieve an AUROC score of 0.98 for the dataset Area-set mainly because it has a number of properties very higher w.r.t than the other two datasets, and so a lot more examples to learn.

Execution time for the training phase. Figure 4.13b reports the execution time of the training phase of HeteroGNN and MRGCN for the multi-relational link prediction task on the heterogeneous graph with the biggest number of properties (movie-set). HeteroGNN takes about half of the execution time of MRGCN on the first 100 epochs. The reason is related to the limited number of features (five features) that we consider, while MRGCN adopts dedicated neural encoders for the embedding of node features.

```

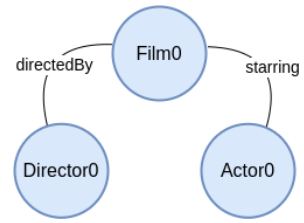
SELECT ?director ?actor
WHERE{
?film rdf:type dbo:Film.
?film dbo:directedBy ?director.
?actor dbo:starring ?film.
?director rdf:type dbo:Director.
?actor rdf:type dbo:Actor.}

```

(a)

director	actor
J. Cameron	L. DiCaprio
J. Cameron	K. Winslet
R. Zemeckis	T. Hanks
P. Weir	R. Williams

(b)



(c)

Figure 4.14: (a) SPARQL query for the generation of a table; (b) extract of the table generated; (c) ground truth *SD*

4.7.2 Validation of the concise *SDs*

The evaluation of the effectiveness of the concise *SD* obtained through our approach is not easy to demonstrate because it depends on the intent of the table designer. Even if the table designer had a specific intent in the creation of a table, when we observe it externally several intents can be identified because of the presence of many kinds of relations that can exist among the concepts. Consider for example the case in which the table designer has created a table for representing the fact that a film is "directed by" a director. However, when we observe the table, we might also point out that the director is "the author" of the film, or the director "editing"/"cinematography" of the film. This variability makes more difficult the evaluation of the effectiveness of the proposed approach by only considering the table. For this reason, we decided to take into account the query according to which the table has been generated and to set up the following procedure to compare the different approaches:

1. Different kinds of queries are developed for extracting tables with 10 tuples from the consolidated knowledge graph. These queries have the purpose to identify different kinds of relations that can exist between the instances of the ontology and the specificity of the relations between involved instances.
2. From the queries, concise *SDs* can be extracted that constitute the ground truth descriptions of the tables.
3. The content of the consolidated knowledge graph is reduced by 30% before applying any approach. In this way, the generated tables might contain also instances already occurring in the knowledge graph but also instances and relations that were never seen before.
4. the semantic description inferred by the approach is compared with the corresponding ground truth for determining the precision and the recall of the approach. Precision

and recall are computed as in [154, 65]:

$$precision = \frac{rel(sm) \cap rel(sm')}{rel(sm')} \quad (4.1)$$

$$recall = \frac{rel(sm) \cap rel(sm')}{rel(sm)} \quad (4.2)$$

where sm is the SD obtained through the approach, sm' is the ground truth SD , and $rel(sm)$ is the set of triples contained in sm .

For example, Figure 4.14a contains the SPARQL query for identifying the director and starring of films. Figure 4.14b contains an excerpt of the table obtained by the evaluation of the query on our KG_C and Figure 4.14c the corresponding ground truth SD .

The described procedure has been applied for testing the performance of our approach on the following three categories of experiments with respect to the approach proposed in [65] (the code of this approach is freely available). For each category of experiments, we have generated 20 tables with a variable number of columns and taken the average precision and recall of the considered approaches with respect to the corresponding ground truth SD s. Table 4.7 summarizes the obtained results. The table reports for each category of experiments, the number of meta-instances initially identified in the considered tables, the number of distinct concepts, and the average precision and recall obtained through the SeMi approach and ours.

Identification of relations between two instances. The purpose of these experiments are the identification of the relations that are pointed out in a table when the table contains two concepts (with a variable number of columns presenting concept properties).

In this category of examples, we have considered distinct concepts that can be: directly connected in our ontology (e.g. an Actor and a Film); indirectly connected (e.g. an Actor and a Director); belonging to an inheritance relationship (e.g. a Person and a Film to be contrasted with an Actor and a CreativeWork). Moreover, we have considered tables presenting two instances of the same concept (e.g. two Actors or two Films). As shown in Table 4.7 our approach outperforms the SeMi approach.

Figure 4.15 shows the SD s that we obtained starting from the use case proposed in Figure 4.14. Figure 4.15a shows the result obtained by using only the KG-based weighting system. Therefore, in this case, erroneously, the spouse relation is predicted. Figure 4.15b shows the obtained result when the combined weighting system is used. In this case the spouse relation (which is inherited from Person) is properly weighted and the relations passing through the concept Film are selected. Finally, Figure 4.15c shows the result obtained by

Exp Cat.	initial <i>SDs</i>		Precision (average)		Recall (average)	
	meta-nodes	concepts	HeteroGNN	SeMi	HeteroGNN	SeMi
Exp 1	2	1	0.91	0.16	0.72	0.16
		2	0.66	0.06	0.33	0.33
Exp 2	3	3	0.77	0.13	0.83	0.33
		2	0.83	0.14	0.83	0.5
Exp 3	4	3	0.6	0.16	0.66	0.22
		2	0.36	0.08	0.36	0.11

Table 4.7: Evaluation results of the generated *SDs*.

the SeMi approach. In this case, a more general concept is identified that binds together an Actor and a Director. Note also that the `createdBy` relation is inferred between CreativeWork and Director instances.

Figure 4.16 shows an example in which our approach does not work as expected. The table in Figure 4.16a was conceived with the aim of introducing an instance of the concept Director between them (as shown by the ground truth *SD* in Figure 4.16b). However, according to the current instance of KG_C also the *SD* in Figure 4.16c, which is generated through our method, is plausible. Therefore, our *SD* is an alternative model for the considered scenario.

Identification of relations among three instances. The purpose of this category of experiments is to verify the stability of the approach when the number of meta-instances increases. Different kinds of tables have been considered: tables involving distinct concepts directly related in the ontology (e.g. an Actor, a Film and a ProductionCompany); tables involving distinct concepts indirectly related in the ontology (e.g. two Actors and a

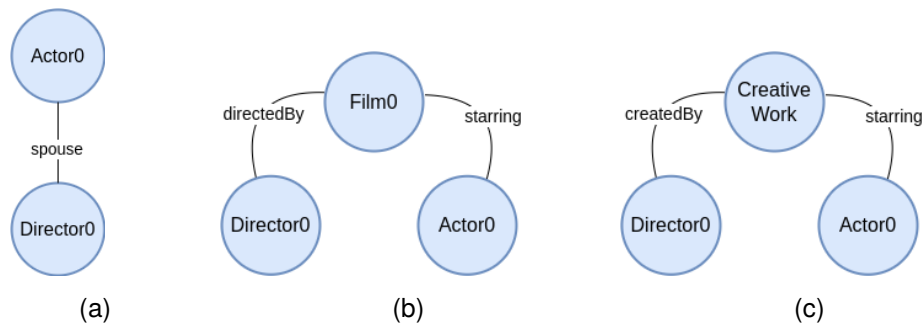


Figure 4.15: *SDs* obtained from the use-case in Figure 4.14 by using (a) our KG-based weighting system; (b) our combined weighted system; (c) the SeMi approach

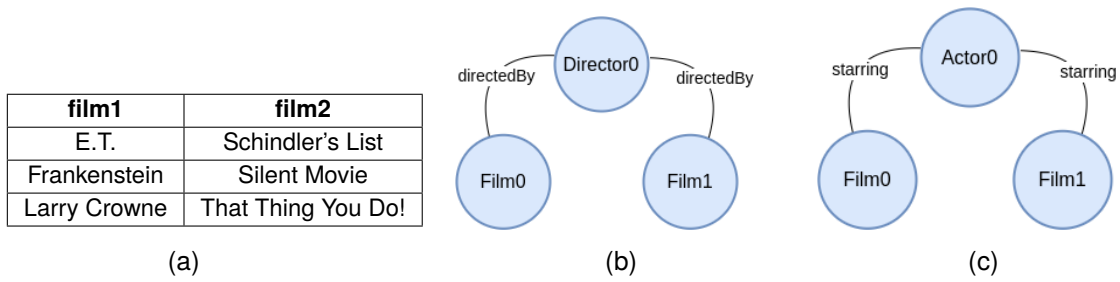


Figure 4.16: (a) Table with two Films; (b) ground truth *SD*; (c) concise *SD*

ProductionCompany); tables involving instances of concepts of the same inheritance hierarchy (e.g. a Person, an Actor, and a Film). As reported in Table 4.7 our approach still outperforms the SeMi approach and also guarantees a precision greater than 0.77.

Figure 4.17a represents the result of our approach on three different instance nodes: TvShow, Director and Person (which actually corresponds to the ground truth). The proposed relations are those that directly connect the three concepts at the ontological level. By contrast, Figure 4.17b shows the *SD* obtained through SeMi. In this case, SeMi introduces the node CreativeWork (which is a superclass of TvShow) and the nodes Film0 and ProductionCompany0 that do not appear to be relevant in this context.

Identification of relations among four instances. This category of experiments has the purpose of further checking the stability of the approach when dealing with much more complex tables. The considered tables extend those used with three concepts by introducing a new concept, an instance of a concept already present in the table, or an instance of a concept that is a specialization/generalization of a concept already included. Even if the performances of this category of experiments reported in Table 4.7 are still better than the

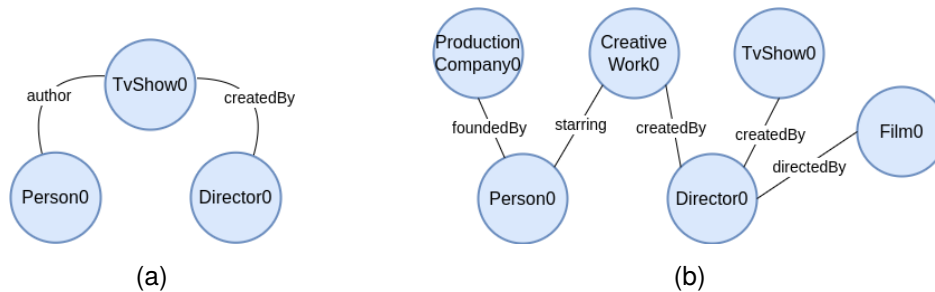


Figure 4.17: Example of *SD* with three concepts: (a) Ground truth/generated *SD*; (b) *SD* obtained by SeMi

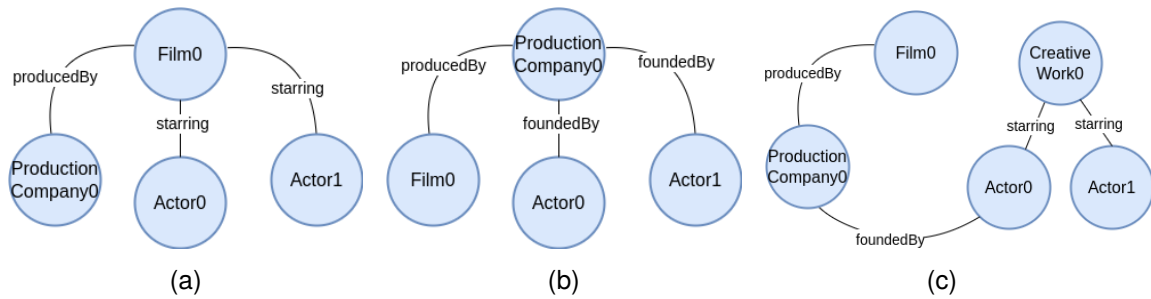


Figure 4.18: Example of SD with four concepts: (a) Ground truth; (b) our generated SD ; (c) the one obtained by SeMi

ones obtained through the SeMi approach, we have to note a decrease in performances. We think that this behaviour is due to the use of the Steiner tree that tends to include in the concise SD shorter paths among those possible. However, this impression should be further investigated.

Our running example is a successful case of identification of the SD for a table presenting four concepts. Figure 4.18 shows an example in which our approach does not work properly. Indeed, starting from the ground truth SD reported in Figure 4.18a, the one in Figure 4.18b is obtained by our approach. The relations to the nodes Actor0 and Actor1 are both named `foundedBy`. Their relations are preferred to the starring ones because that relation was more probable for the GNN. However, also the SD obtained by SeMi (Figure 4.18c) introduces some errors. In our case, the lower weight of the GNN makes the tree algorithm choose the wrong path. In SeMi case, the problem is mainly due to structural problems, indeed, super-concepts are used instead of the ones used in the initial SD .

Discussion. Our experiments pointed out that our approach presents better performances with respect to SeMi. In particular, HeteroGNN gives more satisfying predictions when the initial table contains super-concepts and when the size of the graph increases. In several cases, SeMi retrieves more tuples than required (indeed its recalls are higher than the precision's), it often uses super-concepts instead of the specific ones, and sometimes the final SD loses some concepts that were present in the initial SD .

In HeteroGNN the most problematic use cases are those presenting many instances of the same concept. In these cases, HeteroGNN is not always able to distinguish the relations between the two concepts and chooses the same relation for all the instances (as shown in the example in Figure 4.18).

Exp Cat.	Initial <i>SDs</i>	Properties		Precision
	meta-nodes	total	missing	average
Exp 1	2	6	2	0.76
Exp 2	3	8	3	0.81
Exp 3	4	6	2	0.67

Table 4.8: Evaluation results of the inferred missing semantic properties.

4.7.3 Validation of the prediction of unmatched table columns

We have finally tested the behaviour of our system when the initial *SD* contains unmatched columns and we wish to introduce them through the approach described in Section 4.5.

For this purpose we have considered the same categories of experiments discussed in the previous section and enhanced the queries for extracting tables with:

- columns that contain *extra* properties of the concepts already included in the table (i.e. further details of the concepts that belong to the initial *SD*);
- columns that contain *other* properties, associated with concepts introduced in the complete *SD*.

The newly introduced table columns have been tagged with ? for clarifying that the corresponding ontological property should be predicted. Therefore, the meta-instances of the ground truth *SDs* obtained are the same as the previous experiments but the number of nodes corresponding to table columns is higher and we need to properly associate them with concepts of the ontology.

Example 28 *Figure 4.19 shows an example of extra properties. The initial query is the one reported in Figure 4.14a for creating a table with pairs of actors and directors. The new query has been enhanced for retrieving the date of birth and the children of the director and the awards of the actor. By contrast, Figure 4.20 shows an example of the introduction of other properties starting from the query that associates two films. In this case, the query introduces the name and foundation year of the binding production company. These new introduced columns are marked with ?.* □

Table 4.8 reports the average precision of the experiments that we have conducted that have been classified according to the number of meta-nodes occurring in the initial *SD*. For each category of experiments, we have considered 20 tables containing both extra properties of the concept contained in the initial *SD* and other properties (not contained in the initial *SD*). In the table, we do not report the recall because for each ? property we select only one property and concept. From the reported precision values, we can note that the preci-

```

SELECT ?director ?dirBdate ?children ?actor ?awards
WHERE{
  ?film rdf:type dbo:Film.
  ?film dbo:directedBy ?director.
  ?actor dbo:starring ?film.
  ?director rdf:type dbo:Director.
  ?actor rdf:type dbo:Actor.
  ?actor dbo:awards ?awards.
  ?director dbo:children ?children.
  ?director dbo:birthDate ?dirBdate.}

```

(a)

director	?	?	actor	?
J. Cameron	16/08/1954	2	L. DiCaprio	51
J. Cameron	16/08/1954	2	K. Winslet	59
R. Zemeckis	14/05/1952	4	T. Hanks	53
P. Weir	21/8/1944	2	R. Williams	16

(b)

Figure 4.19: (a) SPARQL query for the generation of extra properties; (b) excerpt of the table generated

sion decreases when we introduce four meta-nodes and this is due to the increase of the possible properties that can be associated with the unmatched columns. The best performance is obtained with three nodes since it is easier to find a path among these classes. The insertion of a fourth node makes the process harder because there are more possible relations to consider. Moreover, from the detailed results emerges that the approach works better when extra properties are considered. Indeed, in this case, some of the properties of instance nodes have already been associated and thus the set of possible properties per concept is reduced. Finally, we can also note that the data type of the properties is also affecting the predictions. Better performances are obtained when trying to predict unmatched columns of type date or decimal, while predictions are worse when the type is integer or string. The reason for this behaviour is related, once again, to the number of possible properties. Usually, the string and integer types are used with a wide variety of data, such as names, addresses, zip codes, years, and so on, while dates and decimals can be used in fewer cases.

Example 29 Consider the situation in Figure 4.19. The first unmatched column is of type date and can be associated with birthDate of the director or birthDate of the actor. The third and fifth unmatched columns are of type int and can be interpreted as the following properties of the complete SD: children of an Actor or a Director,

```

SELECT ?film0 ?film1 ?tName ?year
WHERE{
  ?film0 rdf:type dbo:Film.
  ?film1 rdf:type dbo:Film.
  ?film0 dbo:producedBy ?prodCompany.
  ?film1 dbo:producedBy ?prodCompany.
  ?prodCompany dbo:foundingYear ?year.
  ?prodCompany dbo:tradingName ?tName.
  FILTER(bound(?film0) && ?film0 != ?film1)}

```

(a)

film0	film1	?	?
Mamma Mia!	My Big Fat Greek Wedding 2	Playtone Inc.	1998
Splash	King Arthur (film)	Touchstone Pictures	1984
Gigli	Boyz n the Hood	Columbia Pictures	1924
Kill Bill: Vol1	Kill Bill: Vol2	A band Apart	1991

(b)

Figure 4.20: (a) SPARQL query for the generation of other properties; (b) excerpt of the table generated

year of Film, foundingYear of Company, postalCode of City and numberSeason or numberEpisodes of TelevisionShow. However, the properties year, foundingYear, numberEpisodes and postalCode for the third column are not in the range and therefore are excluded; for the fifth column, the properties year, foundingYear, numberSeason and children are also excluded. The remaining properties associated with the third column are children and numberSeasons. Then, by the application of the Steiner tree, the property children of Director is chosen for this column. Concerning the fifth column, the remaining properties are numberEpisodes (since the years between 1900 and 2000 can be abbreviated to only two digits), awards and foundingYear. The Steiner tree application leads to the selection of the property awards of Film for this column.

Consider now the situation described in Figure 4.20. There are two unmatched columns of two different types. Using the ranges for the last column, only two properties are compatible (i.e. year and foundingYear of Film). Concerning the third column, the prediction is more complex since the ontology contains a high number of properties of type string and the range values are not useful in this specific case. Therefore, the inference is done mainly using the weights of the model. The possible relations obtained using the weighting function are language of Film, knownFor and tradingName for ProductionCompany, birthName and cityName. The application of the Steiner tree leads to the selection of the property language for this column, which is not correct. The reason is that the language is a property

of a `Film` which is already used in `SD` and has a less expensive path. Concerning the fourth column, the compatible properties are `postalCode` and `foundingYear`. In this case, the property `foundingYear` is selected by the Steiner tree which is the correct one. \square

4.8 Concluding remarks

The approach we have proposed in this chapter moves in the same direction as [154] for the generation of a complete `SD` and of [65] for the use of an embedding of the consolidated knowledge graph, however, we introduced innovations from different perspectives. First, we better formalize the problem and introduce requirements for the generation of the complete `SD`. This has the advantage of producing a better characterization of the proposed solution and obtaining more accurate descriptions of the table content. Then, the embedding of the consolidated knowledge graph has been realized by taking into account also the basic properties associated with concepts and adopting a GNN approach designed in order to cope with heterogeneous nodes and edges. Our GNN design introduces an attention mechanism on node-level features and allows recovering the properties of unmatched columns (i.e. columns for which the column type inference approach was not able to associate an ontological concept) by exploiting the embedding of the basic properties of the consolidated knowledge graph. Furthermore, the embedding model is also used for determining properties for unmatched columns (i.e. columns for which the column type inference approach was not able to associate an ontological concept) by exploiting the embedding of the basic properties of the consolidated knowledge graph.

Several experiments have been conducted for assessing the quality of the proposed solution in three different domains. For the evaluation of our GNN approach, we considered [65] and [169] as baseline approaches. By considering the nodes' properties, the AUROC of our approach outperforms the baseline approaches and presents a better execution time. The comparison of our `SDs` with those proposed by [65] suggests that [65] has low precision and recall especially when the table is annotated with more general concepts of the ontology. Moreover, the amount of inferred triples is usually higher and less accurate than ours. Finally, [65] tends to use more general concepts in place of those that should be present, therefore, our approach presents better performances with respect to SeMi in particular when the initial table contains super-concepts and when the size of the graph increases.

Even if the proposed experiments are really promising, we think further experiments should be conducted for validating the proposed methodology. First, we need to consider other datasets presenting a higher number of instances and that strictly follow the constraints imposed by the ontology. At the current stage, many datasets are available but they do not properly adhere to the used ontology. Since we are planning to use the approach in the

biological context, we need to maintain strict adherence to the ontology in order to guarantee the correctness of the predictions that will be realized on the data. Moreover, further experiments should be conducted on the assignment of a property to unmatched columns. At the current stage, the variability of the performances (especially with the *other* properties) requires further investigation.

Chapter 5

Visual management of the semantic description and KG construction

The automatic approach proposed in Chapter 4 infers the SD of a table by considering the consolidated knowledge graph and the domain ontology. The proposed approach might introduce wrong relations among instance nodes or some columns of the table could still be unmatched. In these situations, user intervention is required for concluding and checking the correspondence of SD with the table content and for the translation of the table content in terms of the knowledge graph.

In this chapter, we discuss the user interfaces developed for supporting the user in these two activities. For completing and checking SD , a graphical representation of SD is proposed. The user can thus graphically explore SD and easily check if the labelling is correct, add missing instance nodes, apply modifications if needed, and define relationships among instance nodes. All these operations are executed on the graphical representation of SD . Moreover, specific interfaces are made available for changing columns assignment to the properties of the ontological concepts and for treating unmatched columns.

Once the creation of SD is completed, the knowledge representation of the table content should be generated (what we have named $KG(T)$). However, this operation can be applied only when the required identifying properties are present in SD and values are specified in the table content. Therefore, specific interfaces have been developed for supporting the user in the definition of an identifier for each concept. The aim is to support the user in reducing his/her efforts in this activity.

This chapter is organized as follows. Section 5.1 describes the user interfaces developed

for supporting the user in checking and correcting SD . Section 5.2 shows the transformation functions that can be used, the algorithm for translating the table content in a knowledge graph and the interfaces for fixing and completing the content of the table organized according to SD . Finally, Section 5.3 shows the usability analysis that we have conducted on the interfaces for the management of the graphical representation of SD .

5.1 Interfaces for the management of SD

Starting from the graphical representation of SD , in this section, the interfaces for its manipulation are described. Their purpose is to facilitate the correction of the errors introduced by the automatic techniques discussed in Chapter 4 and to add missing semantic information.

5.1.1 Graphical representation of SD

Figure 5.1 shows an example of the graphical representation that we have developed for $SD = (U_{Cs}, U_T, E_R, E_T)$ (see Section 4.3 in Chapter 4). Green nodes are used for representing instance-nodes (i.e. nodes in U_{Cs}), whereas light blue nodes are used for representing terminal nodes corresponding to the table columns (i.e. nodes in U_T) as illustrated through the orange arrows. Edges between instance-nodes (i.e. E_R) and edges between instance nodes and terminal nodes (i.e. E_T) are represented in the same way (labelled arrows) because their meaning is easily understandable from the context. The label on the edges is the relation/property name.

For each light blue node in the graph, if the column has a single basic type (e.g. the zip column in Figure 5.1), a single incoming edge is present. Multiple incoming edges can be present when the light blue node represents a mixed or union-type column. For example, the Address column is of type mixed and three incoming edges are present (for representing the properties `streetName`, `streetNumber`, and `municipality`). Moreover, the SSN/VAT column is an example of column of type union and two incoming edges are present (one representing the SSN property of the instance-node u_{person}^0 and the other representing the VAT identifier of the instance-node u_{company}^0). We have decided to maintain this simplified representation for keeping simple the illustration. Isolated nodes (i.e. terminal nodes without incoming edges) are not included in our graph representation.

The open-source JavaScript library Cytoscape.js developed by [64] has been used for the generation of an interactive graph representation of SD . The graph is drawn in a canvas where nodes can be added, deleted or moved anywhere. Moreover, operations can be invoked on the nodes/edges through the right button of the mouse. This flexibility and interactivity with the user are crucial in our context for guaranteeing the manipulation of SD by the user.

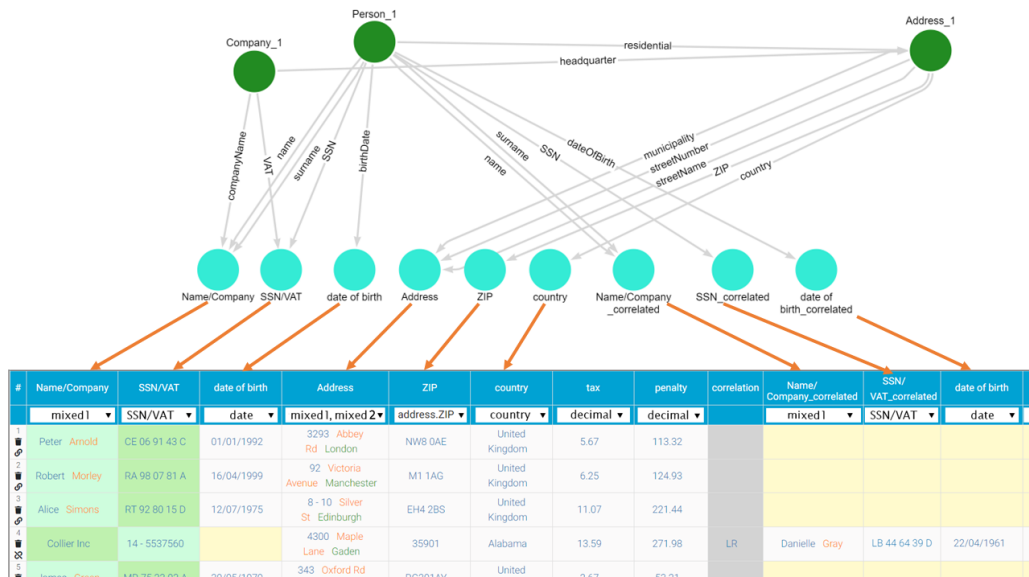


Figure 5.1: Graphical representation of SD and correspondence with table columns

5.1.2 Overview of the main interface

The main interface for the management of SD is shown in Figure 5.2. The graphical representation of SD is reported in the central canvas (2). The green nodes are laid out in the top part of the canvas, whereas, light blue nodes are in the lower part of the canvas. The left panel (1) contains buttons corresponding to the table columns. We exploit a double representation of the table columns (buttons in the left panel and light blue nodes in the central panel) because buttons are used for checking the correctness of the semantic values associated with each column and for adding missing annotations to the unmatched columns. Moreover, the edges in the graphical representation are used for verifying the connections among the components and modifying or adding new ones.

The buttons in the left panel can be coloured in three ways: *green*: it means that the associated column has been already included in SD ; *pink*: it means that the associated column is not yet included in SD (note that unmatched columns are not represented in the graphical representation); *grey*: it is a special placeholder introduced in presence of a correlation among rows in the original table for separating the columns of the main row from those that represent the correlated row. By clicking on the arrow positioned on the left side of the button it is possible to show the data type associated with that column (single type or union of types). By right-clicking on the button itself it is possible to specify a new semantic concept and a property of the domain ontology for each data type of the column.

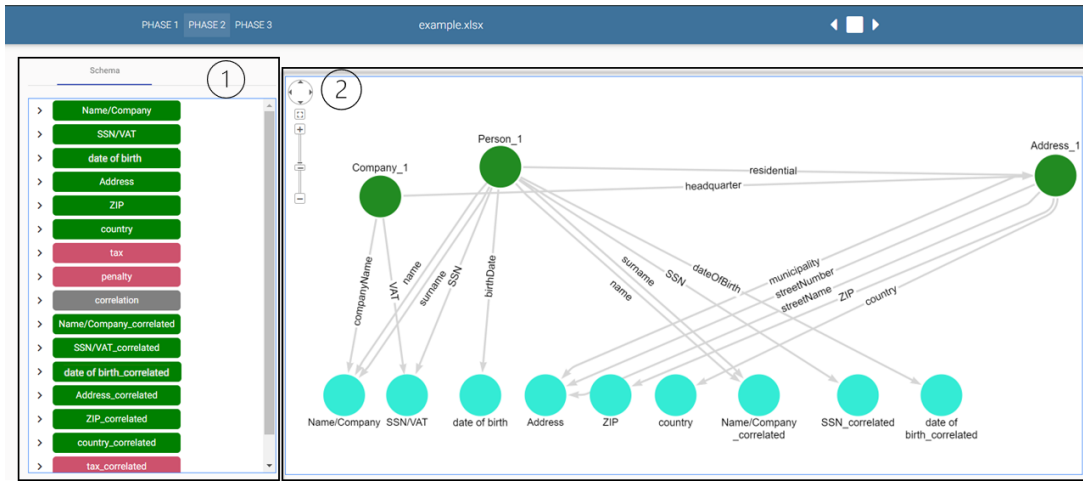


Figure 5.2: Graphical user interface of phase 2

With respect to the graph representation, the position of the nodes in the central canvas can be changed by the user. Moreover, on each node and edge, it is possible to perform different kinds of operations. These operations will be detailed in the next sections.

5.1.3 Visual operations on the table columns

Table 5.1 shows the operations that can be invoked on the table columns reported in the left panel for supporting the user in the correction of errors or in the definition of new nodes.

Operation 1 in Table 5.1 can be invoked on unmatched columns (i.e. pink buttons). This operation is used to include the column content in the SD representation. This operation is specified in two steps. First, we have to identify the properties that represent the column content in the ontology concepts. Then, we have to identify the instance nodes in SD (or new nodes that need to be added in SD) to which the properties can be associated with.

Example 30 Consider the unmatched *tax* column in Figure 5.2. When Operation 1 is invoked on it, the interface in Figure 5.3 is shown to the user. This interface is very similar to the ones presented in Chapter 3 but differs from it because only the data type can be

#	Operation	Description
1	Association of properties	allows the specification of properties to unmatched columns
2	Modification of properties	allows changing the current association of properties for a column
3	Removal of properties	removes the semantic concept associated with the column

Table 5.1: Operations on the table columns

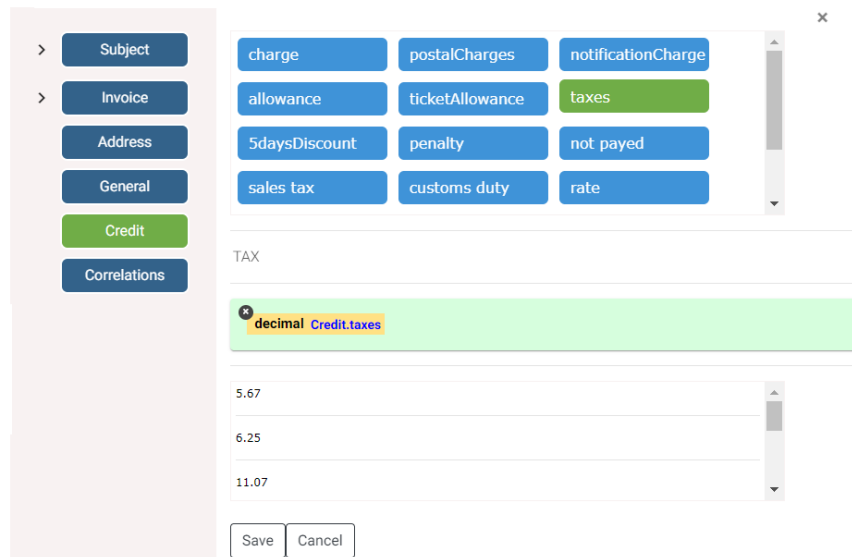


Figure 5.3: Interface for the definition of a semantic type

modified (the definition of a new component of a union type or of a mixed type is not possible anymore). In this case, the concept `Credit` on the left bar and the property `taxes` can be exploited as the semantic annotation of the column. \square

The interface in Figure 5.2 might present a tab above the green area which shows the different types identified in the column that needs to be semantically annotated. Once the specification of the concept and property is terminated, whenever the chosen concept is already present in SD , the interface in Figure 5.4 is shown for determining the instance node to which the properties should be bound. Regardless of the number of instances, after the introduction of a new concept, the system checks the relations existing between the newly inserted element and the other concepts in the ontology. If a single relation is present, it is automatically added to SD , even if the insertion of a third node is required (e.g. between `Person` and `Credit` there is a relation that involves the presence of `Invoice`, therefore the node `Invoice` is added).

Example 31 In the case of the `tax` column, the column `penalty` has been already associated with the instance-node u_{Credit}^0 . So, the user should choose if the same instance-node can be used or if a new one should be defined. Since `taxes` and `penalties` refer the same instance-node, the node already included in SD is selected. At this point, the graphical representation of SD is updated and the button colour becomes green. \square

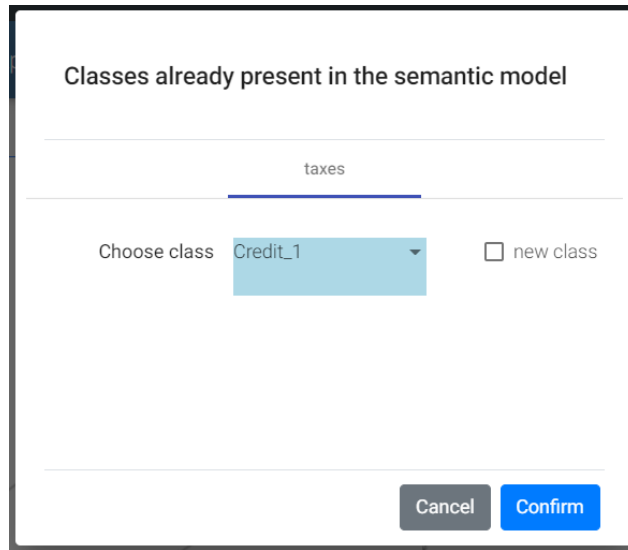


Figure 5.4: Interface for associating the new property to the correct node instance

Operation 2 is used for changing the already associated semantic annotation to a column (i.e. it can be invoked on a green button). Besides changing the semantic annotation through an interface similar to the one discussed for an unmatched column, this operation also allows changing the instance-node to which the properties are associated (if needed).

Example 32 Consider the columns *SSN_correlated*, *Name/Company_correlated* and *date of birth_correlated* in Figure 5.2. They are associated with the instance-node u_{Person}^1 . However, they should not be associated with the invoice subject but with the legal representative of the invoice. Therefore, by using Operation 2, an interface like the one presented in Figure 5.3 is shown to the user where the semantic annotation is already present. The user then directly accesses the interface for the specification of the node instances (Figure 5.4) and requires the introduction of a new instance of the concept *Person*. □

By invoking Operation 3 on a green button, the existing semantic annotation is removed along with the corresponding nodes in the graphical representation.

5.1.4 Visual Operations on the graphical representation of SD

Table 5.2 shows the operations that can be executed on the graphical representation of *SD*. Some of them (1 and 2) can be invoked on light blue nodes and produce the same effect as the corresponding operations that can be applied to the buttons on the left sidebar. Operation 3 can be invoked on an instance-node and allows the introduction of a new link with another instance-node. The inserted links must be coherent with the domain ontology

#	Operation	On	Description
1	edit property	node	the properties of a blue node can be modified
2	delete	node	a node can be removed
3	insert relation	node	a new relation between nodes can be inserted
4	update	edge	the source or the destination of a node can be modified
5	delete	edge	an edge can be removed

Table 5.2: Operations on the graphical representation of SD

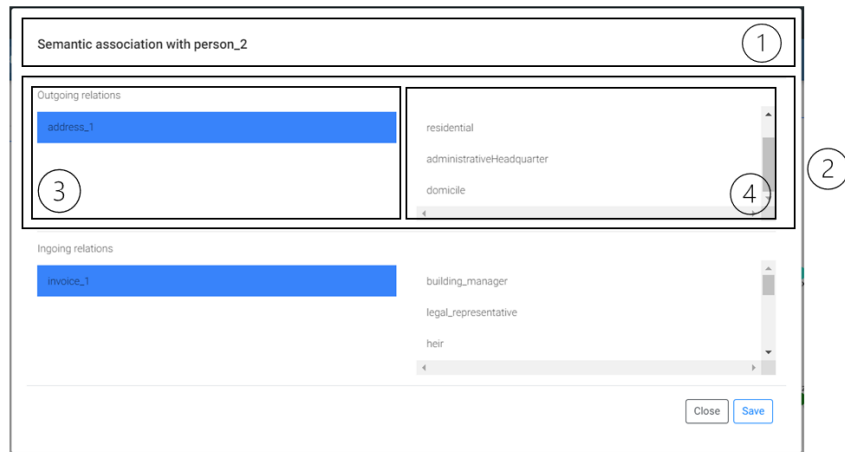


Figure 5.5: Link definition among concepts

so that, for each pair of nodes, only existing relations in the correct direction can be added.

Example 33 Consider the unmatched columns `tax` and `penalty` that have been associated with the instance-node u_{credit}^0 . This instance-node should be linked with the instance-node u_{invoice}^0 . This binding is realized by means of the interface in Figure 5.5. The interface shows the lists of relation names (incoming and outgoing) that can be exploited for the nodes of this concept by taking into account the instance nodes in the current SD and the constraints of the domain ontology. The user can select the correct relation and insert it in the graphical representation of SD that is updated accordingly. \square

Operation 4 of Table 5.2 can be invoked on a node of the graphical representation of SD with the aim of modifying the name of the relation between two nodes or one of the nodes connected by the link. Finally, Operation 5 allows the deletion of an edge occurring in SD .

The following example shows how to complete the semantic description of the table of our running example before starting the translation of the table into a KG representation.

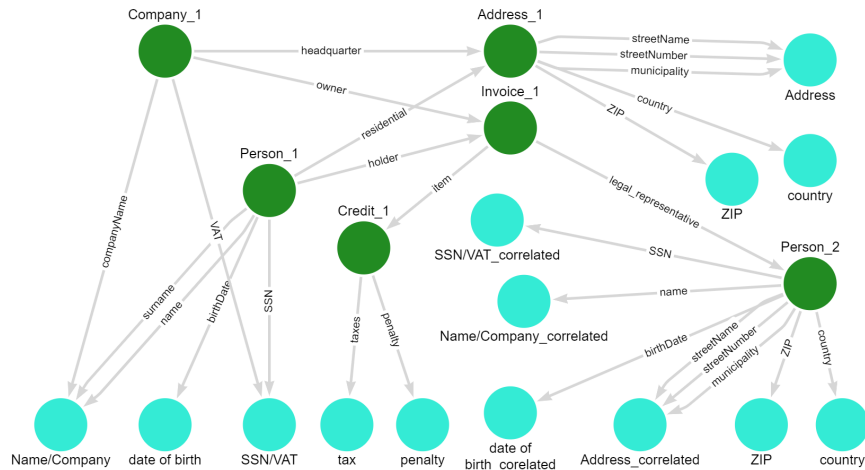


Figure 5.6: Result after the definition of the missing concepts

Example 34 Once the operations described in previous examples are executed, the current *SD* is organized in three separated sub-graphs: *i*) the original one, *ii*) the one having Invoice_1 as the root node; and *iii*) the one having Person_2 as the root node. In order to guarantee the completeness of *SD*, the graph should be connected. By means of Operation 3, the original graph and the graph having Invoice_1 as the root node are connected together through the edge with label holder. Using again Operation 3 starting from Person_2, the subgraph is connected to the graph obtained in the previous step through a link having the correlation name as a label. The obtained result is reported in Figure 5.6. In detail, starting from the node representing Person_2 we open the interface in Figure 5.5 and we choose the incoming link from Invoice_1 to the person through the legal_representative link. This interface can be opened from each node, whose type is reported in the header of the modal (1). For each node, the interface produces the list of the incoming (2) and outgoing links with respect to the node from which the interface has been opened. By querying the domain ontology, we determine all the possible classes (3) and corresponding relations with other classes (4) and we insert them in a list of items. The user can choose both an incoming link and an outgoing link or just one of them. At the end of this process, if the new node does not exist, it is added to the *SD*, and an edge having the chosen label is inserted. When all the edges have been defined, the complete graph shown in Figure 5.6 is obtained. □

5.2 Generation of the knowledge graph

Once the concise SD is completed, the next step is the translation of table T into a knowledge graph $KG(T)$ that contains the content of T represented according to the concepts/properties of the domain ontology. The translation is applied only if mandatory properties and values are contained in the current T .

In this section, we describe the transformation functions that can be exploited for the generation of node identifiers by considering the identifying properties associated with the ontology concepts. Moreover, we describe the translation algorithm that checks the presence of mandatory elements in the table. Whenever they are missing, specific user interfaces have been developed for supporting the user in this activity.

5.2.1 Transformation functions

In order to ensure an identifier for each concept and by taking into account that in many knowledge graphs (e.g. DBpedia, Yago, GeoNames) the instance identifier is realized using identifying properties of the concept, we propose the association of *transformation functions* with the ontology concepts. These functions are specified by the user by taking into account the characteristics of the ontology and allow the definition of human-readable identifiers relying on the values of the identifying properties. The transformation functions are simply python functions that take as input an associative array with the values of the identifying properties and produce a string that is exploited as an identifier. They are invoked each time the instance extracted from a table does not present an identifier.

The transformation functions are usually straightforward to specify and thus we propose a visual language by means of which the user can graphically define and test the behaviour of the function. The language that we propose is inspired by the Scratch language ([143]) with the aim of generating strings that represent identifiers. Our purpose is not the generation of a complicated, complete, and general-purpose visual language, but of a simple approach for creating the identifiers of concept instances. If the user needs to develop more complex logic that cannot be generated by means of the graphical language, he/she can directly write the code in python. This code will be made available and executable directly by the system.

Our visual language is composed of the set of graphical blocks reported in Table 5.3. Four kinds of blocks can be distinguished: the *operation* blocks, which are further divided into *logic* blocks, for the specification of an `if` statement for producing different identifiers depending on the value of the current identifying properties, *binary* blocks, for the manipulation of strings (extracting first/last characters from a string, extracting a substring, generating auto-increment strings) or for generating a terminal string (i.e. a string that should be present in all the identifiers), and the *comparison* blocks for expressing simple and composite logical




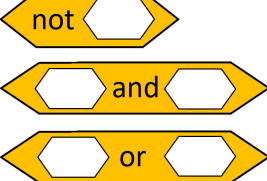
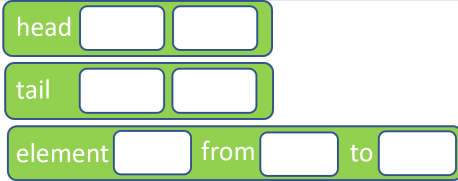
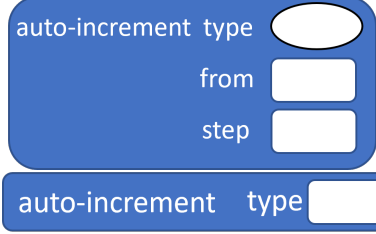

Graphical symbols	Type	Description
	operation	this conditional block returns the string created in one of the two blocks (the then or else branches) depending on the evaluation of the conditional block
	comparison	this block is used to define a compare the two parameters according to the comparison operator
	comparison	comparison operators for the condition block
	logical	logical operators that can be used in a confrontation or in an operation block
	binary	these operators take as input a string and a range and return a substring
	custom	these blocks are used for generating sequences of values of different types (integer, dates) starting from an initial value to the limit
	custom	block for the specification of a string

Table 5.3: Graphical representation of the visual blocks

expressions for the *if* statement; and *variable* blocks for representing the identifying properties of a concept that should be used for the generation of the identifiers. The variable blocks are instantiated with the values of the properties in the consolidated KG.

There are two shapes used for the representation of the blocks: rectangular and hexagonal. The hexagons are used to denote a logical operator or a condition that can be used in a control flow. The rectangles are used for the definition of functions for the manipulation of strings as well as the custom blocks. The input blocks have a rectangular shape, except for the fixed values that are oval (e.g. a function that accepts as input only two possible values).

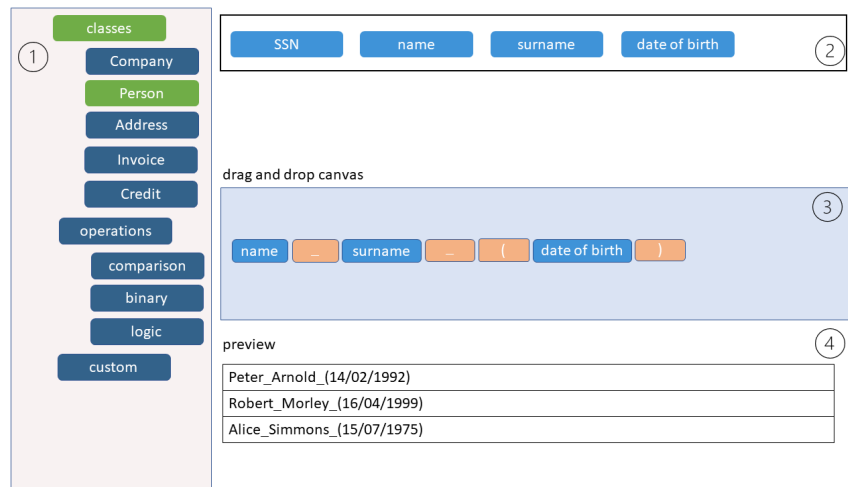


Figure 5.7: Definition of a transformation function for the identifier of a Person

The visual blocks *if-else/if-then-else* are used for defining different identifiers depending on a given condition. The condition can be defined through the comparison blocks and allows three possible inputs: two elements (variables, binary functions or custom values) and a comparison operator (greater than, equal, less than) or logical (and, or) operators. The condition can also be inserted in the not block.

The following data blocks have been developed: $head(str, n)$ returns the first n character of str , $tail(str, n)$ returns the last n character of str and $substring(str, n, m)$ returns the characters between the position n and m of str . Among the custom operators, auto-increment generates identifiers by taking into account the type of the identifier (integer, string) and allows to start from an initial value and to use a step for the generation of the subsequent identifiers. For example, if the type is `integer`, the start value is 3 and the step is 2, the sequence 3, 5, 7, ... will be generated. By contrast, if the type is `char`, the start value is D and the step is 3, the sequence D, G, J, \dots will be generated. Finally, the custom block can be used by the user for the insertion of a custom text.

Through the interface in Figure 5.7 for visual coding, the user can specify how to combine the identifying attributes of a concept to obtain a unique identifier for each entity instance. The interface is organized as follows: the left part (1) contains the following groups:

- *classes*: it is used to contain all the classes occurring in the ontology. When one is selected, the set of its identifying properties is reported in part (2) of the interface.
- *operations*: it contains the visual blocks that can be exploited for defining the logic of

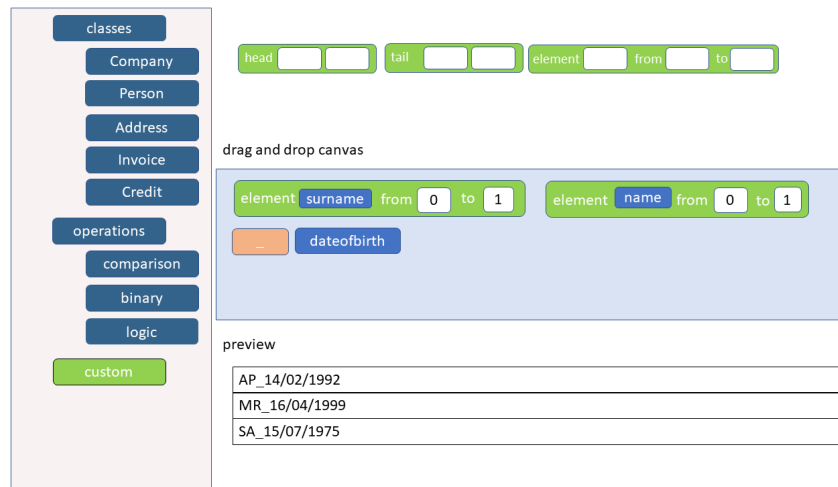


Figure 5.8: Definition of an identifier using a binary operator

the transformation functions organized according to the data and logic operations.

- *custom*: contains a blank block that can be used for specifying strings to be included in all the identifiers, and the auto-increment function that can be used if there is a need for a synthetic identifier.

The content of the top part of the interface in Figure 5.7 (2) varies depending on the group selected from the left panel. If a class is selected, then the corresponding identifying properties are reported. The constructors of the possible operations are reported when the operation group is selected, and finally, the custom block and auto-increment blocks are reported when custom is selected. The blocks contained in (2) can be dragged and dropped in the visual coding area (3) for representing the transformation logic. In this area, all kinds of blocks can be dragged and dropped next to each other and among a block inserted in the area and the next one, the concatenation function is used. A preview of the obtained identifier is shown in (4). In the case of Figure 5.7, the identifier for the concept Person is generated through the concatenation of the name and surname properties separated by an underscore and, within the parentheses, the dateOfBirth property. Further expressions can be realized with our visual language as shown in the next example.

Example 35 *Suppose We wish to create an identifier for the instances of Person. By means of the interface in Figure 5.8, the identifier is defined by using the first letters of the properties name and surname, followed by the dateofbirth property. This result is obtained through the binary function element that is used to extract the first letters.*

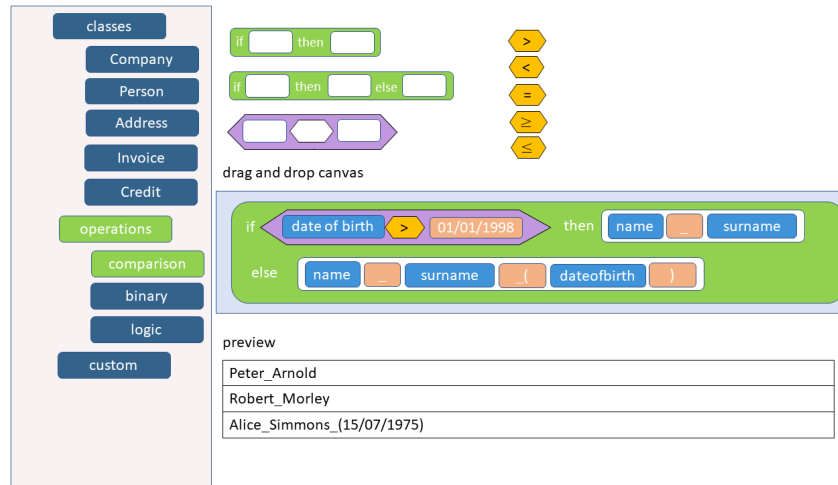


Figure 5.9: Definition of an identifier using a condition

Suppose now that the previous identifier should be applied only to people that were born before the first of January 1998, whereas the identifier is only the concatenation of the name and surname. In this case, the expression reported in Figure 5.9 should be specified that uses the if-then-else block and the concatenation of terminal and variable symbol. □

In some cases, for the identification of a concept, we might wish to exploit serial values as shown in the following example.

Example 36 Suppose we wish to create a synthetic identifier for the Invoice concept. This identifier should be a string starting with Invoice_ followed by an integer number that starts from 2 and has a step of 2. Figure 5.10 shows the specification of this kind of identifier. In the visual coding we have used the auto increment block specific for integer. □

5.2.2 Algorithm for the KG construction

Once SD is completed and the identifiers have been defined, Algorithm 3 is invoked for translating the table T into a knowledge graph $KG(T)$ that contains the content of T represented according to the concepts/properties of the domain ontology. The translation is applied only if mandatory properties and values are contained in the current T . □

Algorithm 3 first checks for each meta-instance u_C^h in SD the occurrence of the "_id" property (line 5). Whenever it is not present, it is included in SD and a transformation function is associated with the node (if specified in the ontology). Then, the algorithm checks the presence of all mandatory properties for u_C^h (line 9). If one of them is missing, u_C^h is enhanced with the mandatory properties and is included in the set of variable nodes

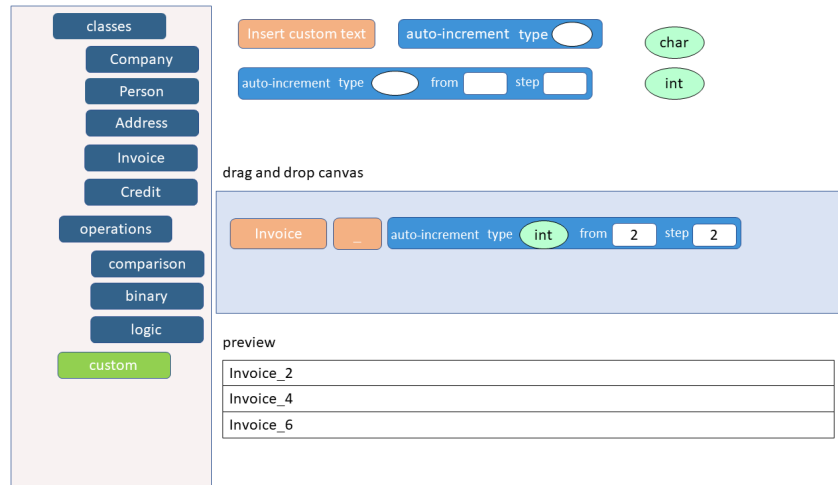


Figure 5.10: Definition of a transformation function for the identifier of an Invoice

VarNodes. The presence of meta-nodes in *VarNodes* does not allow the generation of $KG(T)$ and a message is returned (line 28).

Whenever *VarNodes* is empty (line 15), the second part of the algorithm is activated whose purpose is the generation of the triples for basic properties and relation for each tuple of the table T . In this part of the algorithm, we use the notation $tuple[\langle n, p \rangle]$ for representing the access to the value contained in the current *tuple* that corresponds to the property p associated with the node n in SD . In this second part, for each terminal node u_t in SD that should be included in the knowledge graph (i.e. it is associated with an instance-node of SD), we check the presence of a transformation function for the instance-node u_C^h (to which u_t is associated with) and whenever it is present, it is applied for the generation of the value to be included in $KG(T)$. Otherwise, the value specified in the current tuple is used. Moreover, we check that a value is associated with the property in the tuple if the property is mandatory. If this test fails, it means that a mandatory property is missing and that the generation of $KG(T)$ should be interrupted (line 19), otherwise, a triple, representing the value of the property p on the current row of the table can be included in the set *Triples* (line 20). Once completed the triples for the basic properties, the triples for the relations in SD can be included in *Triples* (lines 22-24). Once all the tuples in T are processed, the set *Triples* can be returned and it contains $KG(T)$ (line 26).

5.2.3 Interface for completing missing information

In case of interruption of the process for the generation of $KG(T)$, the interface in Figure 5.11 is shown to the user. In this interface, the missing mandatory properties are in-

Algorithm 3 Generation of the knowledge graph $KG(T)$

Input: $SD = (U_{C_s}, U_T, E_R, E_T)$ the concise semantic description
 \mathcal{O} the domain ontology
 T the table

Output: $KG(T)$, the representation of T as a knowledge graph

```
1:  $VarNodes := \emptyset$ 
2:  $Triples := \emptyset$ 
3:  $Tran := \emptyset$ 
4: for  $u_C^h \in U_{C_s}$  do
5:   if  $(u_C^h, \_id, u_t) \notin E_T$  then
6:      $E_T := E_T \cup \{(u_C^h, \_id, u_t)\}$ 
7:     if A transformation function  $T_C$  exists for  $C$  then  $Tran := Tran \cup \{(u_C^h, T_C)\}$ 
8:     if Mandatory properties specified for  $C$  are missing in  $u_C^h$  then
9:       Let  $p_1, \dots, p_k$  be the missing identifying properties
10:       $E_T := E_T \cup \bigcup_{i=1}^k \{(u_C^h, p_i, u_t^i)\}$ 
11:       $VarNodes := VarNodes \cup \{u_C^h\}$ 
12: if  $VarNodes = \emptyset$  then
13:   for  $tuple \in T$  do
14:     for  $u_t \in U_t$  s.t.  $(u_C^h, p, u_t) \in E_T$  and  $p \neq \_id$  do
15:       if  $Tran(u_C^h)$  is defined then
16:          $id_c := Tran(u_C^h)(tuple, u_C^h)$ 
17:       else
18:          $id_c := tuple[\langle u_C^h, \_id \rangle]$ 
19:       if ( $p$  is mandatory and  $tuple[\langle u_t, p \rangle]$  is null) then return "not ready"
20:        $Triples := Triples \cup \{(id_c, p, tuple[\langle u_t, p \rangle])\}$ 
21:     for  $(u_{C_s}^h, r, u_{C_d}^h) \in E_R$  do
22:        $Triples := Triples \cup \{(tuple[\langle u_{C_s}^h, \_id \rangle], r, tuple[\langle u_{C_d}^h, \_id \rangle])\}$ 
23:   return  $Triples$ 
24: else
25:   return "not ready"
```

cluded in the table T along with the $_id$ fields with the values eventually generated by the transformation functions (in this case the identifiers are automatically generated).

At this point, the user can check the generated values and complete the missing information. Even if the approach has not yet been included in our approach, the user can exploit some techniques for data augmentation in knowledge graph [39] for the specification of missing information by taking into account the consolidated knowledge graph KG_C or other sources of information. In this way, the user effort can be significantly reduced. Moreover, conditional fixing rules can be specified in the same spirit as those proposed in [23] in the context of IoT data.

credit_1	credit_1	credit_1	person_1	person_1	person_1	person_1	person_1	company_1	company_1	company_1	address_1	address_1	address_1	address_1	address_1	address_1	
_id	taxes	penalty	_id	surname	name	SSN	birthDate	_id	companyName	VAT	_id	streetNumber	municipality	streetName	ZIP	floor	country
0	5.67	113.32	AP_14/02/1992	Peter	Arnold	CE 06 91 43 C	14/01/1992				Abbey_Rd_325(London)	3293	London	Abbey Rd	NW8 0AE		United Kingdom
1	6.25	124.93	MR_16/04/1999	Robert	Mokey	RA 96 07 81 A	16/04/1999				Victoria_Avenue_50(Manchester)	92	Manchester	Victoria Avenue	M1 1AG		United Kingdom
2	1100	220.0	SA_15/07/1975	Alice	Simons	RT 92 80 15 D	15/07/1975				Silver_St_8-10(Edinburgh)	8-10	Edinburgh	Silver	D14 2DS		United Kingdom
3	8.22	164.5				LB 44 64 39 D	22/04/1961	14-5537560	Coller	14-5537560	Stafford_Rd_13A(Newport)	13A	Newport	Stafford Rd	NP19 7DD	39901	Alabama
4	2.67	53.21	GJ_30/05/1979	James	Green	MR 75 33 03 A	30/05/1979				Oxford_Rd_343(Reading)	343	Reading	Oxford Rd	RG30 1AY		United Kingdom
5	3.6	72.11	HB_27/04/1988	Barbara	Howard	NM 00 98 06 B	27/04/1988				Church_Rd_310(Bristol)	310	Bristol	Church Rd	BS5 8AJ		United Kingdom
6	5	100.0						64-8163968	Glover	64-8163968	Green_Road_2918(Alexandria)	2918	Alexandria	Green Road		22304	Vermont

Figure 5.11: Interface for supplying missing information in a table

This interface can also be used also for pointing out semantic errors that can occur in the initial table T or that violate the constraints that can be induced through the domain ontology (besides the structural ones that have been taken into account in the realization of SD). Several are the semantic errors that can occur in the data and should be pointed out. For example, a table row can report that the actress Kate Winslet is married to the film director Edward Abel Smith and another table row that she is married to Sam Mendes. This is a semantic mistake because a person cannot be married to multiple individuals and the error needs to be fixed. Moreover, table rows can violate the constraints that are specified in the ontology like constraints on the number of occurrences of certain relations (e.g. a least one episode for each TelevisionShow series) and constraints on the correspondence among properties (e.g. the total import of an invoice should correspond to the sum of the prices of the items it contains). Last but not least, there can be incorrect or incomplete data, such as outdated information, or incomplete data sources.

The graphical representation of these errors and specific tools that depends on the kind of identified problem can be realized to support the user in fixing the data contained in the initial table and produce data of better quality. In some cases, a cell needs only simple editing, in others, a column should be added or removed from the table. For example, if a person is married to two different people, a possible solution is the insertion of a column containing the divorce date from the first marriage (in the case of Kate Winslet, she is currently married to Edward Abel Smith, after the divorce from Sam Mendes). In case of missing identifying properties, columns should be added for the specification of their values.

Once the missing information has been introduced and the semantic errors have been fixed, the obtained tabular data constitute a new version of the initial table of better quality. In terms of the 5-star deployment scheme for Linked Open Data [12], we can say that the three-phase approach described in this thesis is able to transform a dataset rated with two stars into a

four/five star rating. Indeed, semantic annotations are included in the initial CSV/TSV file and an identifier is assigned to each instance and semantic relationships among the represented concepts. The result is represented in RDF and can be queried through SPARQL. We also remark that the consolidated knowledge graph can always be considered as five stars rating LOD since there is the possibility of inferring new links out of existing facts, and the users can discover more relationships within their linked data.

5.3 Evaluation of usability - phase 2

This section summarizes the results of the usability tests carried out on the interfaces for the management of the semantic description presented in this chapter. The same group of users described in Section 3.5.2 have been involved in conducting the tasks of Table 5.4.

For each task, Table 5.4 reports the main goal, the time required for completing the task, and when the task can be considered successfully completed or when it is completely a failure. All the users have been involved in the first two tasks, while the remaining tasks are executed only by those who had to manage the correlations in Phase 1 (Task 4 of Table 3.2 in Section 3.5.2).

All the individuals who had to deal with mixed types during the first phase were able to complete the first two tasks. The majority of them (90%) completed the job in just 10 minutes while others (10%) had some trouble remembering the procedures to complete Task 2.

For what concerns the last two tasks, some users (20%) needed to watch again the videos to apply the required procedures to complete the assignment correctly. The additional time required for watching videos has not been counted in the total time to complete the tasks. The fact that all users, eventually after watching the introductory videos again, have completed the tasks correctly highlights a possible difficulty for a novice user to learn the various procedures rather than to apply them. The enrichment of contextual help with additional short explanatory videos could improve the user experience of a novice user.

Finally, 25% of the individuals who worked on Task 4 had some problems in finding the correct correlation among the list of possible edges. This suggests that the interface can be improved by adding a search bar on the top of the edge list, increasing the dimension or by grouping the links in categories.

The 85% of the users agreed that the interface is easy-to-use and intuitive whereas the remaining ones expressed a neutral position. The greatest uncertainties concerned the management of the operations that can be executed on the graphical representation of *SD*. In particular, a small part of users has shown difficulty in recognizing or applying operations such as the insertion of new concepts or the insertion or removal of links between concepts.

#	goal	time	success	failure
1	management of unmatched columns	7 min	The user specifies a concept and a property for each unmatched column.	One or more columns have not been associated with a concept and property of the domain ontology
2	connections among concepts	5 min	The user identifies the correctness of the existing links, adds the missing ones and modifies the wrong ones	The final <i>SD</i> is not complete or the links are wrong
3	new instance	4 min	the user is able to define a new instance for a correlated person	the user uses the same instance for both people
4	correlation	6 min	the user specifies the correlation links	the user does not to identify the correlations

Table 5.4: Tasks identified for the usability test of phase 2

Half of the users declared that they had to remove an edge because it was not correct. Only the 16% of the users could not connect all the nodes of the graph because they did not have enough knowledge about the application domain (e.g. they did not know that a company can be the holder of an invoice). Among the users that dealt with the correlations, 60% defined a new instance of the concept *Person*, while the others thought that the single instance was enough for the representation. All the users who worked with the correlations agreed that the process of creating the correlation is easy, while the opinion related to the new instances of a concept was positive for 50% of the users.

In conclusion, the usability test of phase 2 suggests that some aspects of the application can be improved but the overall opinion is that the system is easily usable. We believe that the results were less successful with respect to the results obtained in phase 1 mainly because *i*) the tasks were executed at the end of a long test session that required a high level of attention, we believe that the users' attention was lower with respect to phase 1; *ii*) in this phase, the knowledge of the application domain required to the user is more detailed than the one required in phase 1 (in phase 1 there are simple and well-known data types, such as names and dates, while in the second phase there are links among the concepts, correlations and specific data); and *iii*) the users were not familiar with the ontology and semantic descriptions.

Conclusions and Future Work

The aim of my PhD research activity was the definition of a semi-automatic approach for the construction of a knowledge graph starting from tabular data that adhere to the constraints contained in a domain ontology. The central research questions for this research were:

- How to semi-automatically extract coherent semantic information from heterogeneous spreadsheets?
- How to define a semantic description that characterizes the spreadsheet content?
- How to generate a KG from the extracted (and semantically annotated) data?

To address these research questions, we proposed a three-phase approach and developed a Web application implementing it. The approach is used for identifying a tabular structure within a spreadsheet, correcting syntactic errors, providing a semantic characterization, and transforming the table content into a knowledge graph.

The first phase allows the identification of a table within the spreadsheet, removes the undesired elements (i.e. headers containing descriptions, footers aggregating data), associates a data type to each element of the extracted table and highlights syntactic errors. This is obtained using a machine learning technique that relies on a type system. For each type of the type system, we defined a typing recognizer able to extract values of such type from the table. A multi-label classification approach is then employed for inferring a set of types for each column. Each element whose type is not compliant with the one established for the column is considered an error.

The second phase focuses on the creation of a semantic description SD of the table content that relies on the concepts identified in the first phase to try to identify the relationships existing among them. In the creation of the semantic description we consider all possible relationships that can directly or indirectly exist among the identified concepts and we include them in a graph, named complete SD . Then, the complete SD is enhanced with possible unmatched columns that were not annotated in the first phase. The obtained com-

plete SD is finally weighted according to a weighting system that takes into account the specificity of the relations identified in the ontology and the similarity existing among the nodes of a consolidated Knowledge graph. The similarity is computed in a vectorial representation of the knowledge graph obtained through a relational graph convolutional neural network specifically tailored for inferring relationships among nodes. The approach considers the embedding in the vectorial space also of the basic properties of the instances of the knowledge graph leading to a better performance of the system when predicting the property for unmatched columns. The last step is the extraction of the concise SD that contains the minimal cover tree that can exist among the identified concepts and that minimizes the weights on the paths leading to the table columns. This operation is realized by means of the Steiner tree algorithm. The choice of the minimal cover tree is due to the simplicity of the obtained model that identifies the most likely relationships according to the considered consolidated knowledge graph.

The purpose of the third phase is to create a KG using the table content and the semantic description defined in the previous phase. The translation requires the existence of identifying properties for each meta-node of SD representing a class, therefore constraints have been identified for blocking the translation until all of them are satisfied.

For the realization of the three phases of our approach, we have exploited machine learning techniques that have been coupled with sophisticated user interfaces for supporting the user in checking and adapting the automatic predictions. We believe that interaction with the users is essential in this kind of activity for guaranteeing the development of semantic descriptions of good quality. Both the experiments for checking the precision of the developed approaches and for checking the usability of the graphical interfaces are quite promising and support the idea to create a commercial version of the developed approach.

Several are the directions in which the thesis work can be extended.

The knowledge graph extracted from a table can be integrated into the consolidated knowledge graph. This is an interesting direction and many issues should be faced starting from the identification of similar (or duplicated) instances, the fusion of the entities and the extension of the consolidated knowledge graph with new information inferred from the fusion with the knowledge graph extracted from the table. Even if this is a quite well-known problem in data integration [55], the possibility of using graph embedding techniques (like the ones used in this thesis) appears very promising for obtaining better results.

The semantic description we have proposed in the thesis to characterize the table content could be also exploited for the characterization of a SPARQL query. Therefore, starting from a SPARQL query (eventually presenting basic conditions on properties), the query can be translated into a graph (like the SD graph). Then, by exploiting the domain ontology, the

graph can be enhanced with related concepts and relationships in order to obtain a better formulation of the query that can be used for approximate retrieval in the consolidated knowledge graph. This is an interesting research direction for providing approximate answers to the users' queries that take into account alternative relationships according to which the required data can be bound. In this direction, it would be interesting to consider the issues that might arise when dealing with graph-structured data and how to process the query by considering the embedding of the consolidated knowledge graph.

The graph embedding techniques used in the thesis on knowledge graphs do not take into account the inheritance hierarchy that can be induced by the associated domain ontology. Another interesting research direction is to expand existing approaches for graph embedding to take into account this peculiarity. In this way, instances that represent concepts that belong to the inheritance hierarchy should be represented closer than instances that are not directly related. The development of a more accurate graph embedding approach would produce better predictions in our context.

An interesting research problem that is tightly connected with the issue faced in this thesis is the identification of the ontology to be used for the semantic description of a given table. Indeed, several ontologies are currently available and specifically focused on the representation of a given type of data and the need arises to identify the ontology presenting the highest similarity with the structure and content of tabular data. This problem has gained a lot of attention in the research community as a classification problem and many approaches have been devised especially for semi-structured data (like XML) [13, 6]. The proposed approaches try to extract structural information from the data and develop syntactic and semantic structural similarity measures for identifying the best schema that can be used for the characterization of the considered data. However, the lack of structural information that characterizes tabular data makes this problem particularly challenging and deserves further investigation.

Bibliography

- [1] M. Abadi and et al. TensorFlow: Large-scale machine learning on heterogeneous systems. <https://www.tensorflow.org/>, 2015.
- [2] R. Abraham and M. Erwig. Header and unit inference for spreadsheets through spatial analyses. In *IEEE Symposium on Visual Languages-Human Centric Computing*, pages 165–172. IEEE, 2004.
- [3] R. Abraham and M. Erwig. Ucheck: A spreadsheet type checker for end users. *Journal of Visual Languages and Computing*, 18:71–95, 02 2007.
- [4] M. Adelfio and H. Samet. Schema extraction for tabular data on the web. *Proc. of the VLDB Endowment*, 6:421–432, 04 2013.
- [5] A. F. Agarap. Deep learning using rectified linear units (relu). *CoRR*, 2073, 2018.
- [6] A. Algergawy, M. Mesiti, R. Nayak, and G. Saake. XML data clustering: An overview. *ACM Comput. Surv.*, 43(4):25:1–25:41, 2011.
- [7] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87 – 106, 1987.
- [8] M. Arenas, P. Barcelo, L. Libkin, and F. Murlak. *Relational and XML Data Exchange*. Morgan and Claypool Publishers, 2010.
- [9] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *The Semantic Web*, pages 722–735. Springer, 2007.
- [10] D. W. Barowy, S. Gulwani, T. Hart, and B. Zorn. Flashrelate: Extracting relational data from semi-structured spreadsheets using examples. *SIGPLAN Not.*, 50(6):218–228, 2015.

- [11] Z. Bellahsene, A. Bonifati, and E. Rahm. *Schema Matching and Mapping*. Springer, 2011.
- [12] T. Berners-Lee. 5 star linked data. W3C rec.: https://www.w3.org/2011/gld/wiki/5_Star_Linked_Data, 2006.
- [13] E. Bertino, G. Guerrini, and M. Mesiti. Measuring the structural similarity among XML documents and dtDs. *J. Intell. Inf. Syst.*, 30(1):55–92, 2008.
- [14] C. S. Bhagavatula, T. Noraset, and D. Downey. Tabel: Entity linking in web tables. In *Int'l Semantic Web Conf.*, pages 425–441. Springer, 2015.
- [15] C. Bizer. The emerging web of linked data. *IEEE Intelligent Systems*, 24(5):87–92, 2009.
- [16] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Int'l Conf. on Management of Data*, page 1247–1250. ACM, 2008.
- [17] S. Bonfitto. Semantic integration of heterogeneous and complex spreadsheet tables. In *Database Systems for Advanced Applications: 26th Int'l Int'l, DASFAA 2021, Taipei, Taiwan, April 11–14, 2021, Proceedings, Part III*, page 643–646, Berlin, Heidelberg, 2021. Springer-Verlag.
- [18] S. Bonfitto, L. Cappelletti, E. Casiraghi, P. Perlasca, F. Trovato, G. Valentini, and M. Mesiti. A web tool for the semantic integration of heterogeneous and complex spreadsheet tables. In *Proc. of the 29th Italian Symposium on Advanced Database Systems*, volume 2994 of *CEUR Workshop Proceedings*, pages 116–127, SEBD 2021, Pizzo Calabro (VV), Italy, September 5-9, 2021, 2021. CEUR-WS.org.
- [19] S. Bonfitto, L. Cappelletti, F. Trovato, G. Valentini, and M. Mesiti. Semi-automatic column type inference for CSV table understanding. In *Proc. of 47th Int'l Conf. on Current Trends in Theory and Practice of Computer Science, SOFSEM*, volume 12607 of *Lecture Notes in Computer Science*, pages 535–549, Bolzano, Italy, 2021. Springer.
- [20] S. Bonfitto, E. Casiraghi, and M. Mesiti. Table understanding approaches for extracting knowledge from heterogeneous tables. *WIREs Data Mining Knowl. Discov.*, 11(4), 2021.
- [21] S. Bonfitto, M. Dileo, E. Casiraghi, S. Gaito, G. Valentini, and M. Mesiti. A semi-automatic approach for feeding bio-medical kgs. In *Poster in 5th Advanced School in Computer Science and Engineering: AI for Better Medicine*, 2023.

- [22] S. Bonfitto, M. Dileo, S. Gaito, E. Casiraghi, and M. Mesiti. A semantic approach for constructing knowledge graphs extracted from tables. *Submitted for journal publication, 2023*.
- [23] S. Bonfitto, F. Hachem, E. G. Belay, S. Valtolina, and M. Mesiti. On the bulk ingestion of iot devices from heterogeneous iot brokers. In *IEEE Int'l Congress on Internet of Things (ICIOT)*, pages 189–195, 2019.
- [24] S. Bonfitto, P. Perlasca, and M. Mesiti. Easy-to-use interfaces for supporting the user in the semantic annotation of web tables. In *EDBT/ICDT 2023, 2023*.
- [25] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26:2787–2795, 2013.
- [26] L. Breiman. Random forests. *Mach. Learn.*, 45:5–32, 2001.
- [27] L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and regression trees. *statistics/probability series*, 1984.
- [28] D. Brickley and R. Guha. Rdf schema 1.1, 2014. <https://www.w3.org/TR/rdf-schema/>.
- [29] M. Cafarella, A. Halevy, H. Lee, J. Madhavan, C. Yu, D. Z. Wang, and E. Wu. Ten years of webtables. *Proc. VLDB Endow.*, 11(12):2140–2149, 2018.
- [30] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: Exploring the power of tables on the web. *Proc. VLDB Endow.*, 1(1):538–549, Aug. 2008.
- [31] T. Ceritli, C. K. I. Williams, and J. Geddes. ptype: probabilistic type inference. *Data Mining and Knowledge Discovery*, 34(3):870–904, Mar 2020.
- [32] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view, 2019.
- [33] J. Chen, E. Jimenez-Ruiz, I. Horrocks, and C. Sutton. Learning semantic annotations for tabular data. In *Proc. of the 28 Int'l Joint Conf. on Artificial Intelligence, IJCAI-19*, pages 2088–2094, Macao, China, 7 2019. Int'l Joint Conf. on Artificial Intelligence Organization.
- [34] J. Chen, E. Jiménez-Ruiz, I. Horrocks, and C. Sutton. Colnet: Embedding the semantics of web tables for column type prediction. In *AAAI Conf. on Artificial Intelligence*, volume 33, pages 29–36, 2019.
- [35] X. Chen, L. Chiticariu, M. Danilevsky, A. Evfimievski, and P. Sen. A rectangle mining

- method for understanding the semantics of financial tables. In *Int'l Conf. on Document Analysis and Recognition (ICDAR)*, volume 1, pages 268–273. IEEE, 2017.
- [36] Z. Chen and M. Cafarella. Automatic web spreadsheet data extraction. In *Int'l Workshop on Semantic Search Over the Web*. ACM, 2013.
- [37] Z. Chen, M. Cafarella, J. Chen, D. Prevo, and J. Zhuang. Senbazuru: A prototype spreadsheet database management system. *Proc. VLDB Endow.*, 6(12):1202–1205, 2013.
- [38] Z. Chen, S. Dadiomov, R. Wesley, G. Xiao, D. Cory, M. Cafarella, and J. Mackinlay. Spreadsheet property detection with rule-assisted active learning. In *ACM Conf. on Information and Knowledge Management*, pages 999–1008, 11 2017.
- [39] Z. Chen, Y. Wang, B. Zhao, J. Cheng, X. Zhao, and Z. Duan. Knowledge graph completion: A review. *IEEE Access*, 8:192435–192456, 2020.
- [40] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD Int'l Conf. on Management of Data*, page 1247–1261. ACM, 2015.
- [41] P. Cimiano and H. Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semant. Web*, 8(3):489–508, jan 2017.
- [42] D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Conf. on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.
- [43] Y. Coadou. Boosted decision trees and applications. In *EPJ Web of conferences*, volume 55, page 02004. EDP Sciences, 2013.
- [44] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.
- [45] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- [46] A. "Costa e Silva". *Parts that add up to a whole: a framework for the analysis of tables*. PhD thesis, University of Edinburgh, 2010.
- [47] A. Costa e Silva, A. Jorge, and L. Torgo. Automatic selection of table areas in documents for information extraction. In *Progress in Artificial Intelligence*, volume 2902, pages 460–465, 12 2003.

- [48] A. Costa e Silva, A. M. Jorge, and L. Torgo. Design of an end-to-end method to extract information from tables. *Int'l Journal of Document Analysis and Recognition (IJ DAR)*, 8:144–171, 2006.
- [49] J. Cunha, M. Erwig, J. Mendes, and J. Saraiva. Model inference for spreadsheets. *Automated Software Engineering*, 23:361–392, 09 2014.
- [50] S. Das, S. Sundara, and R. Cyganiak. R2rml: Rdb to rdf mapping language. W3C rec.: www.w3.org/TR/r2rml/, 2012.
- [51] N. Dershowitz and D. A. Plaisted. Chapter 9 - rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, Handbook of Automated Reasoning, pages 535–610. North-Holland, Amsterdam, 2001.
- [52] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. Imap: Discovering complex semantic matches between database schemas. In *Int'l Conf. on Management of Data*, page 383–394. ACM, 2004.
- [53] N. Di Mauro, S. Ferilli, and F. Esposito. Learning to recognize critical cells in document tables. In *Digital Libraries and Archives*, pages 105–116. Springer, 2013.
- [54] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. RML: a generic language for integrated RDF mappings of heterogeneous data. In *Proc. of the 7th Workshop on Linked Data on the Web*, volume 1184 of *CEUR Workshop Proc.*, 2014.
- [55] A. Doan, A. Halevy, and Z. Ives. *Principles of Data Integration*. Morgan Kaufmann Publishers, 2012.
- [56] H. Dong, S. Liu, S. Han, Z. Fu, and D. Zhang. Tablesense: Spreadsheet table detection with convolutional neural networks. In *AAAI Int'l on Artificial Intelligence*, volume 33, pages 69–76, 2019.
- [57] V. Efthymiou, O. Hassanzadeh, M. Rodriguez-Muro, and V. Christophides. Matching web tables with knowledge base entities: From entity lookups to entity embeddings. In *Int'l Semantic Web Conf.*, pages 260–277. Springer, 2017.
- [58] L. Ehrlinger and W. Wöb. Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)*, 48(1-4):2, 2016.
- [59] D. W. Embley, M. Hurst, D. Lopresti, and G. Nagy. Table-processing paradigms: a research survey. *Int'l Journal of Document Analysis and Recognition (IJ DAR)*, 8(2-3):66–86, 2006.

- [60] D. W. Embley, M. S. Krishnamoorthy, G. Nagy, and S. C. Seth. Converting heterogeneous statistical tables on the web to searchable databases. *Int'l Journal on Document Analysis and Recognition (IJ DAR)*, 19:119–138, 2016.
- [61] I. Ermilov and A.-C. N. Ngomo. Taipan: Automatic property mapping for tabular data. In *Int. Conf. Knowledge Engineering and Knowledge Management*, page 163–179, 2016.
- [62] M. Erwig and M. Burnett. Adding apples and oranges. In *Int'l Symposium on Practical Aspects of Declarative Languages*, pages 173–191. Springer, 2002.
- [63] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, New Orleans, Louisiana, USA, 2019. ICLR.
- [64] M. Franz, C. T. Lopes, G. Huck, Y. Dong, O. Sumer, and G. D. Bader. Cytoscape.js: a graph theory library for visualisation and analysis. *Bioinformatics*, 32(2):309–311, 09 2015.
- [65] G. Futia, A. Vetrò, and J. C. De Martin. Semi: A semantic modeling machine to build knowledge graphs with graph neural networks. *SoftwareX*, 12:100516, 2020.
- [66] M. Galkin, D. Mouromtsev, and S. Auer. Identifying web tables: Supporting a neglected type of content on the web. In *Int'l Conf. Knowledge Engineering and Semantic Web*, pages 48–62, 10 2015.
- [67] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak. Towards domain-independent information extraction from web tables. In *Int'l Conf. on World Wide Web*, page 71–80. ACM, 2007.
- [68] C. Gini. On the measure of concentration with special reference to income and statistics. *Colorado College Publication, General Series*, 208:73–79, 1936.
- [69] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Conf. on computer vision and pattern recognition*, pages 580–587. IEEE, 2014.
- [70] J. Gonsior, J. Rehak, M. Thiele, E. Koci, M. Günther, and W. Lehner. Active learning for spreadsheet cell classification. In *Workshops of the EDBT/ICDT Joint Int'l*, volume 2578 of *CEUR Workshop Proc.*, 2020.
- [71] R. C. Gonzales and R. E. Woods. Digital image processing, 2002.

- [72] Google. Openrefine: A free, open source, powerful tool for working with messy data, 2020. <https://openrefine.org/>.
- [73] H. Gulwani. Spreadsheet table transformations from examples. *Commun. ACM*, 2011.
- [74] S. Gulwani, W. R. Harris, and R. Singh. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8):97–105, Aug. 2012.
- [75] W. L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2020.
- [76] L. Han, T. Finin, C. Parr, J. Sachs, and A. Joshi. Rdf123: from spreadsheets to rdf. *Semantic Web*, 5318:451–466, 10 2008.
- [77] J. Handley. Document recognition. *Electronic Imaging Technology*, pages 289–316, 1999.
- [78] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
- [79] M. A. Hasan and M. J. Zaki. A survey of link prediction in social networks. *Social network data analytics*, pages 243–275, 2011.
- [80] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.
- [81] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *IEEE int'l Conf. on computer vision*, pages 2961–2969, 2017.
- [82] P. Heyvaert, B. De Meester, A. Dimou, and R. Verborgh. Declarative rules for linked data generation at your fingertips! In *The Semantic Web: ESWC 2018 Satellite Events*, pages 213–217. Springer, 2018.
- [83] M. Hofmann. Support vector machines-kernels and the kernel trick. *Notes*, 26(3), 2006.
- [84] A. Hogan, E. Blomqvist, M. Cochez, C. D’amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4), jul 2021.
- [85] M. Holeček, A. Hoskovec, P. Baudiš, and P. Klinger. Table understanding in struc-

- tured documents. In *Int'l Conf. on Document Analysis and Recognition Workshops*, volume 5, pages 158–164, 2019.
- [86] A. Holzinger. Interactive machine learning for health informatics: when do we need the human-in-the-loop? *Brain Informatics*, 3(2):119–131, 2016.
- [87] H. H. Hoos and T. Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.
- [88] K. Hu, N. Gaikwad, M. Bakker, M. Hulsebos, E. Zraggen, C. Hidalgo, T. Kraska, G. Li, A. Satyanarayan, and Ç. Demiralp. Viznet: Towards a large-scale visualization learning and benchmarking repository. In *Int'l Conf. on Human Factors in Computing Systems (CHI)*. ACM, 2019.
- [89] M. Hulsebos, K. Hu, M. Bakker, E. Zraggen, A. Satyanarayan, T. Kraska, c. Demiralp, and C. Hidalgo. Sherlock: A deep learning approach to semantic data type detection. In *SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*. ACM, 2019.
- [90] V. Hung, B. Benatallah, and R. Saint-Paul. Spreadsheet-based complex data transformation. In *ACM Int'l Conf. on Information and Knowledge Management*, page 1749–1754. ACM, 2011.
- [91] A. Hur, N. K. Janjua, and M. Ahmed. A survey on state-of-the-art techniques for knowledge graphs construction and challenges ahead. In *Proc. of Int'l Conf. on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 99–103, California, USA, 2021. IEEE.
- [92] M. Hurst. *The Interpretation of Tables in Texts*. PhD thesis, Uni. of Edinburgh, 2000.
- [93] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. Yu. A survey on knowledge graphs: representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33:494–514, Feb. 2022.
- [94] E. Jiménez-Ruiz, O. Hassanzadeh, V. Efthymiou, J. Chen, and K. Srinivas. Semtab 2019: Resources to benchmark tabular data to knowledge graph matching systems. In *The Semantic Web*, pages 514–530. Springer, 2020.
- [95] Z. Jin, M. R. Anderson, M. Cafarella, and H. V. Jagadish. Foofah: Transforming data by example. In *Int'l Conf. on Management of Data*, page 683–698. ACM, 2017.
- [96] E. Kacprzak, J. M. Giménez-García, A. Piscopo, L. Koesten, L.-D. Ibáñez, J. Tension, and E. Simperl. Making sense of numerical data - semantic labelling of web tables. In *Knowledge Engineering and Knowledge Management*, pages 163–178. Springer, 2018.

- [97] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *ACM Human Factors in Computing Systems (CHI)*, page 3363–3372, 2011.
- [98] U. Khurana and S. Galhotra. Semantic concept annotation for tabular data. In *Proc. of the 30th ACM Int'l Conf. on Information and Knowledge Management*, page 844–853, New York, NY, USA, 2021. ACM.
- [99] Y.-S. Kim and K.-H. Lee. Extracting logical structures from html tables. *Computer Standards and Interfaces*, 30(5):296 – 308, 2008.
- [100] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.
- [101] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks, 2017.
- [102] G. Klyne and J. J. Carroll. Resource description framework (rdf): Concepts and abstract syntax, 2004. <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>.
- [103] E. Koci, D. Kuban, N. Luettig, D. Olwig, M. Thiele, J. Gonsior, W. Lehner, and O. Romero. Xlindy: Interactive recognition and information extraction in spreadsheets. In *Symposium on Document Engineering*. ACM, 2019.
- [104] E. Koci, M. Thiele, O. Romero, and W. Lehner. Cell classification for layout recognition in spreadsheets. In *Int'l Conf. on Knowledge Discovery, Knowledge Engineering, and Knowledge Management*, pages 78–100. Springer, 2016.
- [105] E. Koci, M. Thiele, O. Romero, and W. Lehner. A machine learning approach for layout inference in spreadsheets. In *Int'l Joint Conf. on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, page 77–88. SCITEPRESS, 2016.
- [106] E. Koci, M. Thiele, O. Romero, and W. Lehner. Table identification and reconstruction in spreadsheets. In *Int'l Conf. on Advanced Information Systems Engineering*, pages 527–541. Springer, 2017.
- [107] E. Koci, M. Thiele, O. Romero, and W. Lehner. A genetic-based search for adaptive table recognition in spreadsheets. In *Int'l Conf. on Document Analysis and Recognition*, pages 1274–1279, 2019.
- [108] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
- [109] A. Kristiadi, M. A. Khan, D. Lukovnikov, J. Lehmann, and A. Fischer. Incorporating literals into knowledge graph embeddings, 2018.

- [110] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.
- [111] A. Kumar, S. S. Singh, K. Singh, and B. Biswas. Link prediction techniques, applications, and performance: A survey. *Physica A-statistical Mechanics and Its Applications*, 553:124289, 2020.
- [112] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Int'l Conf. on Machine Learning*, page 282–289. Morgan Kaufmann Publishers, 2001.
- [113] A. Langeegger and W. Wöß. Xlwrap – querying and integrating arbitrary spreadsheets with sparql. In *The Semantic Web*, pages 359–374. Springer, 2009.
- [114] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- [115] M. Lefrançois, A. Zimmermann, and N. Bakerally. A sparql extension for generating rdf from heterogeneous formats. In *The Semantic Web*, pages 35–50. Springer, 2017.
- [116] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proc. VLDB Endow.*, 3(1–2):1338–1347, 2010.
- [117] Q. Liu, S. Tang, X. Zhang, X. Zhao, B. Y. Zhao, and H. Zheng. Network growth and link prediction through an empirical lens. *Proc. of the 2016 Internet Measurement Conf.*, 1:1–15, 2016.
- [118] W.-Y. Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23, 2011.
- [119] D. Lopresti and G. Nagy. Automated table processing: An (opinionated) survey. In *IAPR Workshop on Graphics Recognition*, pages 109–134, 1999.
- [120] D. Lopresti and G. Nagy. A tabular survey of automated table processing. In *Int'l Workshop on Graphics Recognition*, pages 93–120. Springer, 1999.
- [121] H. Masuda, S. Tsukamoto, S. Yasutomi, and H. Nakagawa. Recognition of html table structure. In *Int'l Joint Conf. on Natural Language Processing*, pages 183–188, 2004.
- [122] N. D. Mauro, F. Esposito, and S. Ferilli. Finding critical cells in web tables with srl: Trying to uncover the devil's tease. In *12th Int'l Conf. on Document Analysis and Recognition*, pages 882–886, 2013.

- [123] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 10 2013.
- [124] N. Milosevic, C. Gregson, R. Hernandez, and G. Nenadic. Disentangling the structure of tables in scientific literature. In *Int'l Conf. on Applications of Natural Language to Information Systems*, volume 9612, pages 162–174, 06 2016.
- [125] N. Milosevic, C. Gregson, R. Hernandez, and G. Nenadic. A framework for information extraction from tables in biomedical literature. *Int'l Journal on Document Analysis and Recognition (IJ DAR)*, 02 2019.
- [126] V. Mulwad, T. Finin, and A. Joshi. *A Domain Independent Framework for Extracting Linked Semantic Data from Tables*, pages 16–33. Springer, 07 2012.
- [127] V. Mulwad, T. Finin, and A. Joshi. Semantic message passing for generating linked data from tables. In *The Semantic Web Conference*, pages 363–378, Berlin, Heidelberg, 2013. Springer.
- [128] G. Nagy, D. Embley, M. Krishnamoorthy, and S. Seth. Clustering header categories extracted from web tables. *Proc. Int'l Society for Optical Engineering*, 9402, 02 2015.
- [129] N. Novelli and R. Cicchetti. Fun: An efficient algorithm for mining functional and embedded dependencies. In *Int'l Conf. on Database Theory*, pages 189–203, 2001.
- [130] L. Obrst. Ontologies for semantically interoperable systems. In *Proc. of Int'l Conf. on Information and Knowledge Management*, page 366–369, New Orleans, Louisiana, USA, 2003. ACM.
- [131] J. Pearl, M. Glymour, and N. Jewell. *Causal Inference in Statistics: A Primer*. Wiley, 2016.
- [132] F. Pedregosa and et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [133] C. Peterman, C. H. Chang, and H. Alam. A system for table understanding. In *Symposium on document image understanding technology*, pages 55–62, 1997.
- [134] M. Pham, S. Alse, C. A. Knoblock, and P. Szekely. Semantic labeling: A domain-independent approach. In *The Semantic Web Conf.*, pages 446–462, Cham, Germany, 2016. Springer.
- [135] S. Y. Philip, J. Han, and C. Faloutsos. *Link mining: Models, algorithms, and applications*. Springer, 2010.

- [136] D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table extraction using conditional random fields. In *Int'l ACM Conf. on Research and development in information retrieval*, pages 235–242, 2003.
- [137] A. Pivk, P. Cimiano, Y. Sure, M. Gams, V. Rajkovič, and R. Studer. Transforming arbitrary tables into logical form with tartar. *Data and Knowledge Engineering*, 60(3):567–595, 2007.
- [138] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
- [139] V. Raman and J. Hellerstein. Potter’s wheel: An interactive data cleaning system. In *Int'l Conf. Very Large Data Bases*, page 381–390, 09 2001.
- [140] S. K. Ramnandan, A. Mittal, C. A. Knoblock, and P. Szekely. Assigning semantic labels to data sources. In *European Semantic Web Conf.*, pages 403–417. Springer, 2015.
- [141] T. Rebele, F. Suchanek, J. Hoffart, J. Biega, E. Kuzey, and G. Weikum. Yago: A multilingual knowledge base from wikipedia, wordnet, and geonames. In *Int'l Conf. on Semantic Web*, pages 177–185, 10 2016.
- [142] RedHat. Drools: A business rules management system (brms) solution., 2020. [https://https://www.drools.org/](https://www.drools.org/).
- [143] M. Resnik. Scratch, 2006. <https://scratch.mit.edu/>.
- [144] D. Ritze, O. Lehmberg, and C. Bizer. Matching html tables to dbpedia. In *Int'l Conf. on Web Intelligence, Mining and Semantics*. ACM, 2015.
- [145] N. Rümmele, Y. Tyshetskiy, and A. Collins. Evaluating approaches for supervised semantic labeling. In *Workshop on Linked Data on the Web co-located with The Web Conf.*, volume 2073 of *CEUR*, Lyon, France, 2018. TheWebConf Workshop.
- [146] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks, 2017.
- [147] S. Schreiber, S. Agne, I. Wolf, A. Dengel, and S. Ahmed. Deepdesrt: Deep learning for detection and structure recognition of tables in document images. In *IAPR Int'l Conf. on document analysis and recognition (ICDAR)*, volume 1, pages 1162–1167. IEEE, 2017.
- [148] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

- [149] A. Shigarov. Table understanding using a rule engine. *Expert Systems with Applications*, 42:929–937, 02 2015.
- [150] A. Shigarov, V. Khristyuk, A. Mikhailov, and V. Paramonov. Tabbyxl: Rule-based spreadsheet data extraction and transformation. *SoftwareX*, pages 59–75, 10 2019.
- [151] M. A. Shigarov A. Rule-based spreadsheet data transformation from arbitrary to relational tables. *Information Systems*, pages 123–136, 8 2017.
- [152] M. K. Smith, C. Welty, and D. L. McGuinness. Owl web ontology language, 2004. <https://www.w3.org/TR/owl-guide/>.
- [153] J. Son, J. Lee, S. Park, H. Song, S. Lee, and S. Park. Discriminating meaningful web tables from decorative tables using a composite kernel. In *Int'l Conf. on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 368–371, 2008.
- [154] M. Taheriyani, C. A. Knoblock, P. Szekely, and J. L. Ambite. Learning the semantics of structured data sources. *Journal of Web Semantics*, 37-38:152 – 169, 2016.
- [155] K. Takeoka, M. Oyamada, S. Nakadai, and T. Okadome. Meimei: An efficient probabilistic approach for semantically annotating tables. In *AAAI Conf. on Artificial Intelligence*, volume 33, pages 281–288, 2019.
- [156] D. Taniar and J. Rahayu. *Web Semantics & Ontology*. IGI Global research collection. Idea Group Pub., 2006.
- [157] Trifacta. Trifacta wrangler, 2020. <https://www.trifacta.com/>.
- [158] I. Valera and Z. Ghahramani. Automatic discovery of the statistical types of variables in a dataset. In *Proc. of Machine Learning Research*, volume 70, pages 3521–3529, 2017.
- [159] G. J. J. van den Burg, A. Nazábal, and C. Sutton. Wrangling messy csv files by detecting row and type patterns. *Data Mining and Knowledge Discovery*, 33(6):1799–1820, 2019.
- [160] K. N. Vavliakis, T. K. Grollios, and P. A. Mitkas. Rdote - transforming relational databases into semantic web data. In *Int'l Conf. on ISWC - Posters and Demonstrations Track - Volume 658*, page 121–124. CEUR-WS.org, 2010.
- [161] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks, 2017.
- [162] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. *Proc. VLDB Endow.*, 4(9):528–538, 2011.

- [163] B. Vu, C. Knoblock, and J. Pujara. Learning semantic models of data sources using probabilistic graphical models. In *The World Wide Web Conference*, page 1944–1953, San Francisco, California, USA, 2019. ACM.
- [164] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, page 2022–2032, New York, NY, USA, 2019. ACM.
- [165] Y. Wang and J. Hu. A machine learning based approach for table detection on the web. In *Int'l Conf. on World Wide Web*, page 242–250. ACM, 2002.
- [166] D. J. Watts and S. H. Strogatz. *Collective dynamics of 'small-world' networks*, volume 393, pages 440–442. Nature Publishing Group, New York, USA, 1998.
- [167] X. Wei, B. Croft, and A. McCallum. Table extraction for answer retrieval. *Information retrieval*, 9(5):589–611, 2006.
- [168] G. Weikum, X. L. Dong, S. Razniewski, F. Suchanek, et al. Machine knowledge: Creation and curation of comprehensive knowledge bases. *Foundations and Trends® in Databases*, 10(2-4):108–490, 2021.
- [169] W. Wilcke, P. Bloem, V. de Boer, and R. van't Veer. End-to-end learning on multimodal knowledge graphs. *Under Submission*, 2021.
- [170] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.
- [171] R. Xie, Z. Liu, J. Jia, H. Luan, and M. Sun. Representation learning of knowledge graphs with entity descriptions. In *Proc. of the AAAI Conf. on Artificial Intelligence*, volume 30, Phoenix, USA, 2016. PKP.
- [172] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng. Embedding entities and relations for learning and inference in knowledge bases, 2014.
- [173] Y. Yang, R. N. Lichtenwalter, and N. V. Chawla. Evaluating link prediction methods. *Knowledge and Information Systems*, 45(3):751–782, oct 2014.
- [174] R. Zanibbi, D. Blostein, and J. R. Cordy. A survey of table recognition. *Document Analysis and Recognition*, 7(1):1–16, 2004.
- [175] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD Int'l Int'l on Knowledge*

Discovery and Data Mining, KDD '19, page 793–803, New York, NY, USA, 2019. Association for Computing Machinery.

- [176] D. Zhang, M. Hulsebos, Y. Suhara, c. Demiralp, J. Li, and W.-C. Tan. Sato: Contextual semantic type detection in tables. *Proc. VLDB Endow.*, 13(12):1835–1848, jul 2020.
- [177] M. Zhang and Z. Zhou. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837, 2014.
- [178] S. Zhang and K. Balog. Web table extraction, retrieval, and augmentation: A survey. *ACM Trans. Intell. Syst. Technol.*, 11(2), jan 2020.
- [179] Z. Zhang. Effective and efficient semantic table interpretation using tableminer⁺. *Semantic Web*, 8(6):921–957, 2017.